w

Harrison Kinsley & Daniel Kukieła

Acknowledgements

Harrison Kinsley:

My wife, Stephanie, for her unfailing support and faith in me throughout the years. You've never doubted me.
Each and every viewer and person who supported this book and project. Without my audience, none of this would have been possible.
The Python programming community in general for being awesome!

Daniel Kukieła for your unwavering effort with this massive project that Neural Networks from Scratch became. From learning C++ to make mods in GTA V, to Python for various projects, to the calculus behind neural networks, there doesn't seem to be any problem you cannot solve and it is a pleasure to do this for a living with you. I look forward to seeing what's next!

Daniel Kukieła:

My son, Oskar, for his patience and understanding during the busy days. My wife, Katarzyna, for the boundless love, faith and support in all the things I do, have ever done, and plan to do, the sunlight during most stormy days and the morning coffee every single day.
Harrison for challenging me to learn Python then pushing me towards learning neural networks. For showing me that things do not have to be perfectly done, all the support, and making me a part of so many interesting projects including “let’s make a tutorial on neural networks from scratch,” which turned into one the biggest challenges of my life — this book. I wouldn’t be at where I am now if all of that didn’t happen.
The Python community for making me a better programmer and for helping me to improve my language skills.

Copyright

Copyright © 2020 Harrison Kinsley
Cover Design copyright © 2020 Harrison Kinsley

No part of this book may be reproduced in any form or by any electronic or mechanical means, with the following exceptions:

1. Brief quotations from the book.
2. Python Code/software (strings interpreted as logic with Python), which is housed under the MIT license, described on the next page.

License for Code

The Python code/software in this book is contained under the following MIT License:

Copyright © 2020 Sentdex, Kinsley Enterprises Inc., https://nnfs.io

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Readme

The objective of this book is to break down an extremely complex topic, neural networks, into small pieces, consumable by anyone wishing to embark on this journey. Beyond breaking down this topic, the hope is to dramatically demystify neural networks. As you will soon see, this subject, when explored from scratch, can be an educational and engaging experience. This book is for anyone willing to put in the time to sit down and work through it. In return, you will gain a far deeper understanding than most when it comes to neural networks and deep learning.

This book will be easier to understand if you already have an understanding of Python or another programming language. Python is one of the most clear and understandable programming languages; we have no real interest in padding page counts and exhausting an entire first chapter with a basics of Python tutorial. If you need one, we suggest you start here: *https://pythonprogramming.net/python-fundamental-tutorials/* To cite this material:

Harrison Kinsley & Daniel Kukieła Neural Networks from Scratch (NNFS) https://nnfs.io

Chapter 1

Introducing Neural Networks

We begin with a general idea of what **neural networks** are and why you might be interested in them. Neural networks, also called **Artificial Neural Networks** (though it seems, in recent years, we've dropped the "artificial" part), are a type of machine learning often conflated with deep learning. The defining characteristic of a *deep* neural network is having two or more **hidden layers** — a concept that will be explained shortly, but these hidden layers are ones that the neural network controls. It's reasonably safe to say that most neural networks in use are a form of deep learning.

Fig 1.01: Depicting the various fields of artificial intelligence and where they fit in overall.

A Brief History

Since the advent of computers, scientists have been formulating ways to enable machines to take input and produce desired output for tasks like **classification** and **regression**. Additionally, in general, there's **supervised** and **unsupervised** machine learning. Supervised machine learning is used when you have pre-established and labeled data that can be used for training. Let's say you have sensor data for a server with metrics such as upload/download rates, temperature, and humidity, all organized by time for every 10 minutes. Normally, this server operates as intended and has no outages, but sometimes parts fail and cause an outage. We might collect data and then divide it into two classes: one class for times/observations when the server is operating normally, and another class for times/observations when the server is experiencing an outage. When the server is failing, we want to label that sensor data leading up to failure as data that preceded a failure. When the server is operating normally, we simply label that data as "normal."

What each sensor measures in this example is called a feature. A group of features makes up a feature set (represented as vectors/arrays), and the values of a feature set can be referred to as a sample. Samples are fed into neural network models to train them to fit desired outputs from these inputs or to predict based on them during the inference phase.

The "normal" and "failure" labels are **classifications** or **labels.** You may also see these referred to as **targets** or **ground-truths** while we fit a machine learning algorithm. These targets are the classifications that are the *goal* or *target*, known to be *true and correct*, for the algorithm to learn. For this example, the aim is to eventually train an algorithm to read sensor data and accurately predict when a failure is imminent. This is just one example of supervised learning in the form of classification. In addition to classification, there's also regression, which is used to predict numerical values, like stock prices. There's also unsupervised machine learning, where the machine finds structure in data without knowing the labels/classes ahead of time. There are additional concepts (e.g., reinforcement learning and semi-supervised machine learning) that fall under the umbrella of neural networks. For this book, we will focus on classification and regression with neural networks, but what we cover here leads to other use-cases.

Neural networks were conceived in the 1940s, but figuring out how to train them remained a mystery for 20 years. The concept of **backpropagation** (explained later) came in the 1960s, but neural networks still did not receive much attention until they started winning competitions in 2010. Since then, neural networks have been on a meteoric rise due to their sometimes seemingly

magical ability to solve problems previously deemed unsolvable, such as image captioning, language translation, audio and video synthesis, and more.

Currently, neural networks are the primary solution to most competitions and challenging technological problems like self-driving cars, calculating risk, detecting fraud, and early cancer detection, to name a few.

What is a Neural Network?

"Artificial" neural networks are inspired by the organic brain, translated to the computer. It's not a perfect comparison, but there are neurons, activations, and lots of interconnectivity, even if the underlying processes are quite different.

Fig 1.02: Comparing a biological neuron to an artificial neuron.

A single neuron by itself is relatively useless, but, when combined with hundreds or thousands (or many more) of other neurons, the interconnectivity produces relationships and results that frequently outperform any other machine learning methods.

Fig 1.03: Example of a neural network with 3 hidden layers of 16 neurons each.

Anim 1.03: https://nnfs.io/ntr

The above animation shows the examples of the model structures and the numbers of parameters the model has to learn to adjust in order to produce the desired outputs. The details of what is seen here are the subjects of future chapters.

It might seem rather complicated when you look at it this way. Neural networks are considered to be "black boxes" in that we often have no idea *why* they reach the conclusions they do. We do understand *how* they do this, though.

Dense layers, the most common layers, consist of interconnected neurons. In a dense layer, each neuron of a given layer is connected to every neuron of the next layer, which means that its output value becomes an input for the next neurons. Each connection between neurons has a weight associated with it, which is a trainable factor of how much of this input to use, and this weight gets multiplied by the input value. Once all of the *inputs·weights* flow into our neuron, they are

summed, and a bias, another trainable parameter, is added. The purpose of the bias is to offset the output positively or negatively, which can further help us map more real-world types of dynamic data. In chapter 4, we will show some examples of how this works.

The concept of weights and biases can be thought of as "knobs" that we can tune to fit our model to data. In a neural network, we often have thousands or even millions of these parameters tuned by the optimizer during training. Some may ask, "why not just have biases or just weights?" Biases and weights are both tunable parameters, and both will impact the neurons' outputs, but they do so in different ways. Since weights are multiplied, they will only change the magnitude or even completely flip the sign from positive to negative, or vice versa. *Output = weight·input+bias* is not unlike the equation for a line $y = mx+b$. We can visualize this with:

Fig 1.04: Graph of a single-input neuron's output with a weight of 1, bias of 0 and input *x*.

Adjusting the weight will impact the slope of the function:

Fig 1.05: Graph of a single-input neuron's output with a weight of 2, bias of 0 and input *x*.

As we increase the value of the weight, the slope will get steeper. If we decrease the weight, the slope will decrease. If we negate the weight, the slope turns to a negative:

Fig 1.06: Graph of a single-input neuron's output with a weight of -0.70, bias of 0 and input *x*.

This should give you an idea of how the weight impacts the neuron's output value that we get from *inputs·weights+bias.* Now, how about the bias parameter? The bias offsets the overall function. For example, with a weight of 1.0 and a bias of 2.0:

Fig 1.07: Graph of a single-input neuron's output with a weight of 1, bias of 2 and input *x*.

As we increase the bias, the function output overall shifts upward. If we decrease the bias, then the overall function output will move downward. For example, with a negative bias:

Fig 1.08: Graph of a single-input neuron's output with a weight of 1.0, bias of -0.70 and input *x*.

Anim 1.04-1.08: https://nnfs.io/bru

As you can see, weights and biases help to impact the outputs of neurons, but they do so in slightly different ways. This will make even more sense when we cover **activation functions** in chapter 4. Still, you can hopefully already see the differences between weights and biases and how they might individually help to influence output. Why this matters will be conveyed shortly.

As a very general overview, the step function meant to mimic a neuron in the brain, either “firing” or not — like an on-off switch. In programming, an on-off switch as a function would be called a **step function** because it looks like a step if we graph it.

Fig 1.09: Graph of a step function.

For a step function, if the neuron’s output value, which is calculated by *sum(inputs · weights) + bias*, is greater than 0, the neuron fires (so it would output a 1). Otherwise, it does not fire and would pass along a 0. The formula for a single neuron might look something like:

```
IJŁĻĴŁĻ ł ĽŁİâĨĮĴŁĻĽ ắ ŃĦĨÍÎĻĽã к ĠĨĞĽ
```

We then usually apply an activation function to this output, noted by *activation()*:

```
IJŁĻĴŁĻ ł ĞĢĻĨĿĞĻĨIJĮâIJŁĻĴŁĻã
```

While you can use a step function for your activation function, we tend to use something slightly more advanced. Neural networks of today tend to use more informative activation functions (rather than a step function), such as the **Rectified Linear** (ReLU) activation function, which we will cover in-depth in Chapter 4. Each neuron’s output could be a part of the ending output layer, as well as the input to another layer of neurons. While the full function of a neural network can get very large, let’s start with a simple example with 2 hidden layers of 4 neurons each.

Fig 1.10: Example basic neural network.

Along with these 2 hidden layers, there are also two more layers here — the input and output layers. The input layer represents your actual input data, for example, pixel values from an image or data from a temperature sensor. While this data can be "raw" in the exact form it was collected, you will typically **preprocess** your data through functions like **normalization** and **scaling**, and your input needs to be in numeric form. Concepts like scaling and normalization will be covered later in this book. However, it is common to preprocess data while retaining its features and having the values in similar ranges between 0 and 1 or -1 and 1. To achieve this, you will use either or both scaling and normalization functions. The output layer is whatever the neural network returns. With classification, where we aim to predict the class of the input, the output layer often has as many neurons as the training dataset has classes, but can also have a single output neuron for binary (two classes) classification. We'll discuss this type of model later and, for now, focus on a classifier that uses a separate output neuron per each class. For example, if our goal is to classify a collection of pictures as a "dog" or "cat," then there are two classes in total. This means our output layer will consist of two neurons; one neuron associated with "dog" and the other with "cat." You could also have just a single output neuron that is "dog" or "not dog."

Fig 1.11: Visual depiction of passing image data through a neural network, getting a classification

For each image passed through this neural network, the final output will have a calculated value in the "cat" output neuron, and a calculated value in the "dog" output neuron. The output neuron that received the highest score becomes the class prediction for the image used as input.

Fig 1.12: Visual depiction of passing image data through a neural network, getting a classification

Anim 1.11-1.12: https://nnfs.io/qtb

The thing that makes neural networks appear challenging is the math involved and how scary it can sometimes look. For example, let's imagine a neural network, and take a journey through what's going on during a simple forward pass of data, and the math behind it. Neural networks are really only a bunch of math equations that we, programmers, can turn into code. For this, do not worry about understanding everything. The idea here is to give you a high-level impression of what's going on overall. Then, this book's purpose is to break down each of these elements into painfully simple explanations, which will cover both forward and backward passes involved in training neural networks.

When represented as one giant function, an example of a neural network's forward pass would be computed with:

$$L = -\sum_{l=1}^{N} y_l log(\forall_{j=1}^{n_3} \frac{e^{\sum_{i=1}^{n_2}(\forall_{j=1}^{n_2} max(0,\sum_{i=1}^{n_1}(\forall_{j=1}^{n_1} max(0,\sum_{i=1}^{n_0} X_i w_{1,i,j}+b_{1,j}))_i w_{2,i,j}+b_{2,j}))_i w_{3,i,j}+b_{3,j}}}{\sum_{k=1}^{n_3} e^{\sum_{i=1}^{n_2}(\forall_{j=1}^{n_2} max(0,\sum_{i=1}^{n_1}(\forall_{j=1}^{n_1} max(0,\sum_{i=1}^{n_0} X_i w_{1,i,j}+b_{1,k}))_i w_{2,i,j}+b_{2,k}))_i w_{3,i,k}+b_{3,k}}})$$

https://nnfs.io

Fig 1.13: Full formula for the forward pass of an example neural network model.

Anim 1.13: https://nnfs.io/vkt

Naturally, that looks extremely confusing, and the above is actually the easy part of neural networks. This turns people away, understandably. In this book, however, we're going to be coding everything from scratch, and, when doing this, you should find that there's no step along the way to producing the above function that is very challenging to understand. For example, the above function can also be represented in nested python functions like:

```
loss = -np.log(
    np.sum(
        y * np.exp(
            np.dot(
                np.maximum(
                    0,
                    np.dot(
                        np.maximum(
                            0,
                            np.dot(
                                X,
                                w1.T
                            ) + b1
                        ),
                        w2.T
                    ) + b2
                ),
                w3.T
            ) + b3
        ) /
        np.sum(
            np.exp(
                np.dot(
                    np.maximum(
                        0,
                        np.dot(
                            np.maximum(
                                0,
                                np.dot(
                                    X,
                                    w1.T
                                ) + b1
                            ),
                            w2.T
                        ) + b2
                    ),
                    w3.T
                ) + b3
            ),
            axis=1,
            keepdims=True
        )
    )
)
```

https://nnfs.io

Fig 1.14: Python code for the forward pass of an example neural network model.

There may be some functions there that you don't understand yet. For example, maybe you do not know what a log function is, but this is something simple that we'll cover. Then we have a sum operation, an exponentiating operation (again, you may not exactly know what this does, but it's nothing hard). Then we have a dot product, which is still just about understanding how it works, there's nothing there that is over your head if you know how multiplication works! Finally, we have some transposes, noted as .T, which, again, once you learn what that operation does, is not a challenging concept. Once we've separated each of these elements, learning what they do and how they work, suddenly, things will not appear to be as daunting or foreign. Nothing in this forward pass requires education beyond basic high school algebra! For an animation that depicts how all of this works in Python, you can check out the following animation, but it's certainly not expected that you'd immediately understand what's going on. The point is that this seemingly complex topic can be broken down into small, easy to understand parts, which is the purpose of the coming chapters!

Anim 1.14: https://nnfs.io/vkr

A typical neural network has thousands or even up to millions of adjustable **parameters** (weights and biases). In this way, neural networks act as enormous functions with vast numbers of **parameters**. The concept of a long function with millions of variables that could be used to solve a problem isn't all too difficult. With that many variables related to neurons, arranged as interconnected layers, we can imagine there exist some combinations of values for these variables that will yield desired outputs. Finding that combination of parameter (weight and bias) values is the challenging part.

The end goal for neural networks is to adjust their weights and biases (the parameters), so when applied to a yet-unseen example in the input, they produce the desired output. When supervised machine learning algorithms are trained, we show the algorithm examples of inputs and their associated desired outputs. One major issue with this concept is **overfitting** — when the algorithm only learns to fit the training data but doesn't actually "understand" anything about underlying input-output dependencies. The network basically just "memorizes" the training data.

Thus, we tend to use "in-sample" data to train a model and then use "out-of-sample" data to validate an algorithm (or a neural network model in our case). Certain percentages are set aside for both datasets to partition the data. For example, if there is a dataset of 100,000 samples of data and labels, you will immediately take 10,000 and set them aside to be your "out-of-sample" or "validation" data. You will then train your model with the other 90,000 in-sample or "training" data and finally validate your model with the 10,000 out-of-sample data that the model hasn't yet seen. The goal is for the model to not only accurately predict on the training data, but also to be similarly accurate while predicting on the withheld out-of-sample validation data.

This is called **generalization**, which means learning to fit the data instead of memorizing it. The idea is that we "train" (slowly adjusting weights and biases) a neural network on many examples of data. We then take out-of-sample data that the neural network has never been presented with and hope it can accurately predict on these data too.

You should now have a general understanding of what neural networks are, or at least what the objective is, and how we plan to meet this objective. To train these neural networks, we calculate

how “wrong” they are using algorithms to calculate the error (called **loss**), and attempt to slowly adjust their parameters (weights and biases) so that, over many iterations, the network gradually becomes less wrong. The goal of all neural networks is to generalize, meaning the network can see many examples of never-before-seen data, and accurately output the values we hope to achieve. Neural networks can be used for more than just classification. They can perform regression (predict a scalar, singular, value), clustering (assign unstructured data into groups), and many other tasks. Classification is just a common task for neural networks.

Supplementary Material: https://nnfs.io/ch1
Chapter code, further resources, and errata for this chapter.

Chapter 2

Coding Our First Neurons

While we assume that we're all beyond beginner programmers here, we will still try to start slowly and explain things the first time we see them. To begin, we will be using **Python 3.7** (although any version of Python 3+ will likely work). We will also be using **NumPy** after showing the pure-Python methods and Matplotlib for some visualizations. It should be the case that a huge variety of versions should work, but you may wish to match ours exactly to rule out any version issues. Specifically, we are using:

Python 3.7.5
NumPy 1.15.0
Matplotlib 3.1.1

Since this is a *Neural Networks from Scratch in Python* book, we will demonstrate how to do things without NumPy as well, but NumPy is Python's all-things-numbers package. Building from scratch is the point of this book though ignoring NumPy would be a disservice since it is among the most, if not the most, important and useful packages for data science in Python.

A Single Neuron

Let's say we have a single neuron, and there are three inputs to this neuron. As in most cases, when you initialize parameters in neural networks, our network will have weights initialized randomly, and biases set as zero to start. Why we do this will become apparent later on. The input will be either actual training data or the outputs of neurons from the previous layer in the neural network. We're just going to make up values to start with as input for now:

```
ĨĮĴŁĻĹ ł äĝd ğd ġā
```

Each input also needs a weight associated with it. Inputs are the data that we pass into the model to get desired outputs, while the weights are the parameters that we'll tune later on to get these results. Weights are one of the types of values that change inside the model during the training phase, along with biases that also change during training. The values for weights and biases are what get "trained," and they are what make a model actually work (or not work). We'll start by making up weights for now. Let's say the first input, at index 0, which is a 1, has a weight of 0.2, the second input has a weight of 0.8, and the third input has a weight of -0.5. Our input and weights lists should now be:

```
ĨĮĴŁĻĹ ł äĝd ğd ġā
ŃĦĨÍÎĻĹ ł äflgğd flgíd hflgĥā
```

Next, we need the bias. At the moment, we're modeling a single neuron with three inputs. Since we're modeling a single neuron, we only have one bias, as there's just one bias value per neuron. The bias is an additional tunable value but is not associated with any input in contrast to the weights. We'll randomly select a value of 2 as the bias for this example:

```
ĨĮĴŁĻĹ ł äĝd ğd ġā
ŃĦĨÍÎĻĹ ł äflgğd flgíd hflgĥā
ĠĨĞĠĹ ł ğ
```

This neuron sums each input multiplied by that input's weight, then adds the bias. All the neuron does is take the fractions of inputs, where these fractions (weights) are the adjustable parameters, and adds another adjustable parameter — the bias — then outputs the result. Our output would be calculated up to this point like:

```
IJŁĻĴŁĻ ł âĨĮĴŁĻĽäfläǻŃĦĨÍÎĻĽäfläǻ κ
          ĨĮĴŁĻĽäĝäǻŃĦĨÍÎĻĽäĝäǻ κ
          ĨĮĴŁĻĽäğäǻŃĦĨÍÎĻĽäğäǻ κ ĠĨĞĽã

ĴĹĨĮĻâIJŁĻĴŁĻã

ńńń
ğgġ
```

The output here should be **2.3**. We will use ńńń to denote output in this book.

```
inputs = [1.0, 2.0, 3.0]
weights = [0.2, 0.8, -0.5]
bias = 2.0

output = inputs[0]*weights[0] + inputs[1]*weights[1] + inputs[2]*weights[2] + bias
print(output)

>>> 2.3
```

Fig 2.01: Visualizing the code that makes up the math of a basic neuron.

Anim 2.01: https://nnfs.io/bkr

What might we need to change if we have 4 inputs, rather than the 3 we've just shown? Next to the additional input, we need to add an associated weight, which this new input will be multiplied with. We'll make up a value for this new weight as well. Code for this data could be:

```
ĨĮĴŁĻĽ ł äĝgfld ğgfld ġgfld ğgĥā
ŃĦĨÍÎĻĽ ł äflgğd flgíd hflgĥd ĝgflā
ĠĨĞĽ ł ğgfl
```

Which could be depicted visually as:

Fig 2.02: Visualizing how the inputs, weights, and biases from the code interact with the neuron.

Anim 2.02: https://nnfs.io/djp

All together in code, including the new input and weight, to produce output:

```
ĨĮĴŁĻĽ ł äĝgfld ğgfld ġgfld ğgĥā
ŃĦĨÍÎĻĽ ł äflgğd flgíd hflgĥd ĝgflā
ĠĨĞĽ ł ğgfl

IJŁĻĴŁĻ ł âĨĮĴŁĻĽäflāǻŃĦĨÍÎĻĽäflā ĸ
          ĨĮĴŁĻĽäĝāǻŃĦĨÍÎĻĽäĝā ĸ
          ĨĮĴŁĻĽäğāǻŃĦĨÍÎĻĽäğā ĸ
          ĨĮĴŁĻĽäġāǻŃĦĨÍÎĻĽäġā ĸ ĠĨĞĽā
```

ĴĹĨĮĻâIJŁĻĴŁĻã

ńńń
ǵgí

Visually:

```
inputs = [1.0, 2.0, 3.0, 2.5]
weights = [0.2, 0.8, -0.5, 1.0]
bias = 2.0

output = inputs[0]*weights[0] + inputs[1]*weights[1] + inputs[2]*weights[2] + inputs[3]*weights[3] + bias
print(output)

>>> 4.8
```

Fig 2.03: Visualizing the code that makes up a basic neuron, with 4 inputs this time.

Anim 2.03: https://nnfs.io/djp

A Layer of Neurons

Neural networks typically have layers that consist of more than one neuron. Layers are nothing more than groups of neurons. Each neuron in a layer takes exactly the same input — the input given to the layer (which can be either the training data or the output from the previous layer), but contains its own set of weights and its own bias, producing its own unique output. The layer's output is a set of each of these outputs — one per each neuron. Let's say we have a scenario with 3 neurons in a layer and 4 inputs:

Fig 2.04: Visualizing a layer of neurons with common input.

Anim 2.04: https://nnfs.io/mxo

We'll keep the initial 4 inputs and set of weights for the first neuron the same as we've been using so far. We'll add 2 additional, made up, sets of weights and 2 additional biases to form 2 new neurons for a total of 3 in the layer. The layer's output is going to be a list of 3 values, not just a single value like for a single neuron.

```
ĨĮĴŁĻĽ ł äĝd ğd ġd ğgĥā

ŃĦĨÍÎĻĽĝ ł äflgğd flgíd hflgĥd ĝā
ŃĦĨÍÎĻĽğ ł äflgĥd hflgîĝd flgğħd hflgĥā
ŃĦĨÍÎĻĽġ ł ähflgğħd hflgğìd flgĝìd flgíìā

ĠĨĞĽĝ ł ğ
ĠĨĞĽğ ł ġ
ĠĨĞĽġ ł flgĥ

IJŁĻĴŁĻĽ ł ä
        ſi NĦŁĹIJĮ ĝſ
        ĨĮĴŁĻĽäflāǻŃĦĨÍÎĻĽĝäflā ĸ
        ĨĮĴŁĻĽäĝāǻŃĦĨÍÎĻĽĝäĝā ĸ
        ĨĮĴŁĻĽäğāǻŃĦĨÍÎĻĽĝäğā ĸ
        ĨĮĴŁĻĽäġāǻŃĦĨÍÎĻĽĝäġā ĸ ĠĨĞĽĝd

        ſi NĦŁĹIJĮ ğſ
        ĨĮĴŁĻĽäflāǻŃĦĨÍÎĻĽğäflā ĸ
        ĨĮĴŁĻĽäĝāǻŃĦĨÍÎĻĽğäĝā ĸ
        ĨĮĴŁĻĽäğāǻŃĦĨÍÎĻĽğäğā ĸ
        ĨĮĴŁĻĽäġāǻŃĦĨÍÎĻĽğäġā ĸ ĠĨĞĽğd

        ſi NĦŁĹIJĮ ġſ
        ĨĮĴŁĻĽäflāǻŃĦĨÍÎĻĽġäflā ĸ
        ĨĮĴŁĻĽäĝāǻŃĦĨÍÎĻĽġäĝā ĸ
        ĨĮĴŁĻĽäğāǻŃĦĨÍÎĻĽġäğā ĸ
        ĨĮĴŁĻĽäġāǻŃĦĨÍÎĻĽġäġā ĸ ĠĨĞĽġā

ĴĹĨĮĻâIJŁĻĴŁĻĽã

ńńń
äǵgíd ĝgğĝd ğgġíĥā
```

```
inputs = [1.0, 2.0, 3.0, 2.5]
weights1 = [0.2, 0.8, -0.5, 1.0]
weights2 = [0.5, -0.91, 0.26, -0.5]
weights3 = [-0.26, -0.27, 0.17, 0.87]
bias1 = 2.0
bias2 = 3.0
bias3 = 0.5

outputs = [
    inputs[0]*weights1[0] + inputs[1]*weights1[1] + inputs[2]*weights1[2] + inputs[3]*weights1[3] + bias1,
    inputs[0]*weights2[0] + inputs[1]*weights2[1] + inputs[2]*weights2[2] + inputs[3]*weights2[3] + bias2,
    inputs[0]*weights3[0] + inputs[1]*weights3[1] + inputs[2]*weights3[2] + inputs[3]*weights3[3] + bias3
]
print(outputs)

>>> [4.8, 1.21, 2.385]
```

https://nnfs.io

Fig 2.04.2: Code, math and visuals behind a layer of neurons.

Anim 2.04: https://nnfs.io/mxo

In this code, we have three sets of weights and three biases, which define three neurons. Each neuron is "connected" to the same inputs. The difference is in the separate weights and bias that each neuron applies to the input. This is called a **fully connected** neural network — every neuron in the current layer has connections to every neuron from the previous layer. This is a very common type of neural network, but it should be noted that there is no requirement to fully connect everything like this. At this point, we have only shown code for a single layer with very few neurons. Imagine coding many more layers and more neurons. This would get very challenging to code using our current methods. Instead, we could use a loop to scale and handle dynamically-sized inputs and layers. We've turned the separate weight variables into a list of weights so we can iterate over them, and we changed the code to use loops instead of the hardcoded operations.

```
ĨĮĴŁĻĽ ł äĝd ğd ġd ğgĥā
ŃĦĨÍÎĻĽ ł ääflgğd flgíd hflgĥd ĝād
           äflgĥd hflgîĝd flgğħd hflgĥād
           ähflgğħd hflgğìd flgĝìd flgíìāā
ĠĨĞĽĦĽ ł äğd ġd flgĥā

fi OŁĻĴŁĻ IJÌ ĢŁĹĹĦĮĻ ĬĞÑĦĹ
ĬĞÑĦĹçIJŁĻĴŁĻĽ ł āā
fi FIJĹ ĦĞĢÎ ĮĦŁĹIJĮ
ÌIJĹ ĮĦŁĹIJĮçŃĦĨÍÎĻĽd ĮĦŁĹIJĮçĠĨĞĽ ĨĮ ĻĨĴâŃĦĨÍÎĻĽd ĠĨĞĽĦĽãf
    fi ZĦĹIJĦĤ IJŁĻĴŁĻ IJÌ ÍĨĿĦĮ ĮĦŁĹIJĮ
    ĮĦŁĹIJĮçIJŁĻĴŁĻ ł fl
    fi FIJĹ ĦĞĢÎ ĨĮĴŁĻ ĞĮĤ ŃĦĨÍÎĻ ĻIJ ĻÎĦ ĮĦŁĹIJĮ
    ÌIJĹ ĮçĨĮĴŁĻd ŃĦĨÍÎĻ ĨĮ ĻĨĴâĨĮĴŁĻĽd ĮĦŁĹIJĮçŃĦĨÍÎĻĽãf
        fi MŁĬĻĨĴĬÑ ĻÎĨĽ ĨĮĴŁĻ ĠÑ ĞĽĹIJĢĨĞĻĦĤ ŃĦĨÍÎĻ
        fi ĞĮĤ ĞĤĤ ĻIJ ĻÎĦ ĮĦŁĹIJĮnĽ IJŁĻĴŁĻ ĿĞĹĨĞĠĬĦ
        ĮĦŁĹIJĮçIJŁĻĴŁĻ κł ĮçĨĮĴŁĻńŃĦĨÍÎĻ
    fi AĤĤ ĠĨĞĽ
    ĮĦŁĹIJĮçIJŁĻĴŁĻ κł ĮĦŁĹIJĮçĠĨĞĽ
    fi PŁĻ ĮĦŁĹIJĮnĽ ĹĦĽŁĬĻ ĻIJ ĻÎĦ ĬĞÑĦĹnĽ IJŁĻĴŁĻ ĬĨĹĽĻ
    ĬĞÑĦĹçIJŁĻĴŁĻĽgĞĴĴĦĮĤâĮĦŁĹIJĮçIJŁĻĴŁĻã

ĴĹĨĮĻâĬĞÑĦĹçIJŁĻĴŁĻĽã


ńńń
ägíd ĝgĝĝd ĝgġíĥā
```

This does the same thing as before, just in a more dynamic and scalable way. If you find yourself confused at one of the steps, ĴĹĨĮĻâã out the objects to see what they are and what's happening. The ĻĨĴâã function lets us iterate over multiple iterables (lists in this case) simultaneously. Again, all we're doing is, for each neuron (the outer loop in the code above, over neuron weights and biases), taking each input value multiplied by the associated weight for that input (the inner loop in the code above, over inputs and weights), adding all of these together, then adding a bias at the end. Finally, sending the neuron's output to the layer's output list.

That's it! How do we know we have three neurons? Why do we have three? We can tell we have three neurons because there are 3 sets of weights and 3 biases. When you make a neural network of your own, you also get to decide how many neurons you want for each of the layers. You can combine however many inputs you are given with however many neurons that you desire. As you progress through this book, you will gain some intuition of how many neurons to try using. We will start by using trivial numbers of neurons to aid in understanding how neural networks work at their core.

With our above code that uses loops, we could modify our number of inputs or neurons in our layer to be whatever we wanted, and our loop would handle it. As we said earlier, it would be a disservice not to show NumPy here since Python alone doesn't do matrix/tensor/array math very efficiently. But first, the reason the most popular deep learning library in Python is called "TensorFlow" is that it's all about doing operations on **tensors**.

Tensors, Arrays and Vectors

What are "tensors?"

Tensors are *closely-related to* arrays. If you interchange tensor/array/matrix when it comes to machine learning, people probably won't give you too hard of a time. But there are subtle differences, and they are primarily either the context or attributes of the tensor object. To understand a tensor, let's compare and describe some of the other data containers in Python (things that hold data). Let's start with a list. A Python list is defined by comma-separated objects contained in brackets. So far, we've been using lists.

This is an example of a simple list:

```
Ĭ ł äĝdĥdħdğā
```

A list of lists:

```
ĬIJĬ ł ääĝdĥdħdğād
       äġdğdĝdġāā
```

A list of lists of lists!

```
ĬIJĬIJĬ ł äääĝdĥdħdğād
          äġdğdĝdġāād
         ääĥdğdĝdğād
          äħdġdídġāād
         äägdídĥdġād
          äĝdĝdîdġāāā
```

Everything shown so far could also be an array or an array representation of a tensor. A list is just a list, and it can do pretty much whatever it wants, including:

```
ĞĮIJĻÎĦĹçĬĨĽĻçIJÌçĬĨĽĻĽ ł äägdğdġād
                          äĥdĝāā
```

The above list of lists cannot be an array because it is not **homologous**. A list of lists is homologous if each list along a dimension is identically long, and this must be true for each dimension. In the case of the list shown above, it's a 2-dimensional list. The first dimension's length is the number of sublists in the total list (2). The second dimension is the length of each of those sublists (3, then 2). In the above example, when reading across the "row" dimension (also called the second dimension), the first list is 3 elements long, and the second list is 2 elements long — this is not homologous and, therefore, cannot be an array. While failing to be consistent in one dimension is enough to show that this example is not homologous, we could also read down the "column" dimension (the first dimension); the first two columns are 2 elements long while the third column only contains 1 element. Note that every dimension does not necessarily need to be the same length; it is perfectly acceptable to have an array with 4 rows and 3 columns (i.e., 4x3).

A matrix is pretty simple. It's a rectangular array. It has columns and rows. It is two dimensional. So a matrix can be an array (a 2D array). Can all arrays be matrices? No. An array can be far more than just columns and rows, as it could have four dimensions, twenty dimensions, and so on.

```
ĬĨĽĻçİĞĻĹĨÑçĞĹĹĞÑ ł äägdğād
                    äĥdĝād
                    äídğāā
```

The above list could also be a valid matrix (because of its columns and rows), which automatically means it could also be an array. The "shape" of this array would be 3x2, or more formally described as a shape of *(3, 2)* as it has 3 rows and 2 columns.

To denote a shape, we need to check every dimension. As we've already learned, a matrix is a 2-dimensional array. The first dimension is what's inside the most outer brackets, and if we look at the above matrix, we can see 3 lists there: ägdğā, äĥdĝā, and äídğā; thus, the size in this dimension is 3 and each of those lists has to be the same shape to form an array (and matrix in this case). The next dimension's size is the number of elements inside this more inner pair of brackets, and we see that it's 2 as all of them contain 2 elements.

With 3-dimensional arrays, like in *lolol* below, we'll have a 3rd level of brackets:

```
ǏIJǏIJǏ ł äääĝdĥdħdǧād
          äġdǧdĝdġāād
         äähdǧdĝdǧād
          äħdǵdídǵāād
         äägdídĥdġād
          äĝdĝdîdǵāāā
```

The first level of this array contains 3 matrices:

```
äägdĥdħdǧād
 äġdǧdĝdġāā

äähdǧdĝdǧād
 äħdǵdídǵāā
```

And

```
äägdídĥdġād
 äĝdĝdîdǵāā
```

That's what's inside the most outer brackets and the size of this dimension is then 3. If we look at the first matrix, we can see that it contains 2 lists — äĝdĥdħdǧā and äġdǧdĝdġā so the size of this dimension is 2 — while each list of this inner matrix includes 4 elements. These 4 elements make up the 3rd and last dimension of this matrix since there are no more inner brackets. Therefore, the shape of this array is *(3, 2, 4)* and it's a 3-dimensional array, since the shape contains 3 dimensions.

Fig 2.05: Example of a 3-dimensional array.

Anim 2.05: https://nnfs.io/jps

Finally, what's a tensor? When it comes to the discussion of tensors versus arrays in the context of computer science, pages and pages of debate have ensued. This intense debate appears to be caused by the fact that people are arguing from entirely different places. There's no question that a tensor is not just an array, but the real question is: "What is a tensor, to a computer scientist, in the context of deep learning?" We believe that we can solve the debate in one line:

A tensor object is an object that can be represented as an array.

What this means is, as programmers, we can (and will) treat tensors as arrays in the context of deep learning, and that's really all the thought we have to put into it. Are all tensors *just* arrays? No, but they are represented as arrays in our code, so, to us, they're only arrays, and this is why there's so much argument and confusion.

Now, what is an array? In this book, we define an array as an ordered homologous container for numbers, and mostly use this term when working with the NumPy package since that's what the main data structure is called within it. A linear array, also called a 1-dimensional array, is the simplest example of an array, and in plain Python, this would be a list. Arrays can also consist of multi-dimensional data, and one of the best-known examples is what we call a matrix in mathematics, which we'll represent as a 2-dimensional array. Each element of the array can be accessed using a tuple of indices as a key, which means that we can retrieve any array element.

We need to learn one more notion — a vector. Put simply, a vector in math is what we call a list in Python or a 1-dimensional array in NumPy. Of course, lists and NumPy arrays do not have the same properties as a vector, but, just as we can write a matrix as a list of lists in Python, we can also write a vector as a list or an array! Additionally, we'll look at the vector algebraically (mathematically) as a set of numbers in brackets. This is in contrast to the physics perspective, where the vector's representation is usually seen as an arrow, characterized by a magnitude and a direction.

Dot Product and Vector Addition

Let's now address vector multiplication, as that's one of the most important operations we'll perform on vectors. We can achieve the same result as in our pure Python implementation of multiplying each element in our inputs and weights vectors element-wise by using a **dot product**, which we'll explain shortly. Traditionally, we use dot products for **vectors** (yet another name for a container), and we can certainly refer to what we're doing here as working with vectors just as we can call them "tensors." Nevertheless, this seems to add to the mysticism of neural networks — like they're these objects out in a complex multi-dimensional vector space that we'll never understand. Keep thinking of vectors as arrays — a 1-dimensional array is just a vector (or a list in Python).

Because of the sheer number of variables and interconnections made, we can model very complex and non-linear relationships with non-linear activation functions, and truly feel like wizards, but this might do more harm than good. Yes, we will be using the "dot product," but we're doing this because it results in a clean way to perform the necessary calculations. It's nothing more in-depth than that — as you've already seen, we can do this math with far more rudimentary-sounding words. When multiplying vectors, you either perform a dot product or a cross product. A cross product results in a vector while a dot product results in a scalar (a single value/number).

First, let's explain what a dot product of two vectors is. Mathematicians would say:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

A dot product of two vectors is a sum of products of consecutive vector elements. Both vectors must be of the same size (have an equal number of elements).

Let's write out how a dot product is calculated in Python. For it, you have two vectors, which we can represent as lists in Python. We then multiply their elements from the same index values and then add all of the resulting products. Say we have two lists acting as our vectors:

```
Ǧ ł äĝdǧdġā
Ġ ł äǧdġdǵā
```

To obtain the dot product:

```
ĤIJĻçĴĹIJĤŁĢĻ ł ĞäflāǻĠäflā κ ĞäĝāǻĠäĝā κ ĞäğāǻĠäğā
ĴĹĨĮĻâĤIJĻçĴĹIJĤŁĢĻã

ńńń
ğfl
```

```
a = [1, 2, 3]
b = [2, 3, 4]

dot_product = a[0]*b[0] + a[1]*b[1] + a[2]*b[2]
>>> 20
```

$$\vec{a} \cdot \vec{b} = [1, 2, 3] \cdot [2, 3, 4] = 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 = 20$$

https://nnfs.io

Fig 2.06: Math behind the dot product example.

Anim 2.06: https://nnfs.io/xpo

Now, what if we called *a* "inputs" and *b* "weights?" Suddenly, this dot product looks like a succinct way to perform the operations we need and have already performed in plain Python. We need to multiply our weights and inputs of the same index values and add the resulting values together. The dot product performs this exact type of operation; thus, it makes lots of sense to use here. Returning to the neural network code, let's make use of this dot product. Plain Python does not contain methods or functions to perform such an operation, so we'll use the NumPy package, which is capable of this, and many more operations that we'll use in the future.

We'll also need to perform a vector addition operation in the not-too-distant future. Fortunately, NumPy lets us perform this in a natural way — using the plus sign with the variables containing vectors of the data. The addition of the two vectors is an operation performed element-wise, which means that both vectors have to be of the same size, and the result will become a vector of this

size as well. The result is a vector calculated as a sum of the consecutive vector elements:
$\vec{a} + \vec{b} = [a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n]$

A Single Neuron with NumPy

Let's code the solution, for a single neuron to start, using the dot product and the addition of the vectors with NumPy. This makes the code much simpler to read and write (and faster to run):

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ĨĮĴŁĻĽ ł äĝgfld ğgfld ġgfld ğgĥā
ŃĦĨÍÎĻĽ ł äflgğd flgíd hflgĥd ĝgflā
ĠĨĞĽ ł ğgfl


IJŁĻĴŁĻĽ ł ĮĴgĤIJĻâŃĦĨÍÎĻĽd ĨĮĴŁĻĽã κ ĠĨĞĽ

ĴĹĨĮĻâIJŁĻĴŁĻĽã

ńńń
ǵgí
```

```
inputs = [1.0, 2.0, 3.0, 2.5]
weights = [0.2, 0.8, -0.5, 1.0]
bias = 2.0

outputs = np.dot(weights, inputs) + bias
```

https://nnfs.io

```
np.dot([0.2, 0.8, -0.5, 1.0], [1.0, 2.0, 3.0, 2.5]) =
= 0.2*1.0 + 0.8*2.0 + -0.5*3.0 + 1.0*2.5 = 2.8
```

Fig 2.07: Visualizing the math of the dot product of inputs and weights for a single neuron.

```
inputs = [1.0, 2.0, 3.0, 2.5]
weights = [0.2, 0.8, -0.5, 1.0]
bias = 2.0

outputs = np.dot(weights, inputs) + bias
>>> 4.8


np.dot(weights, inputs) + bias = 2.8 + 2.0 = 4.8
```

Fig 2.08: Visualizing the math summing the dot product and bias.

Anim 2.07-2.08: https://nnfs.io/blq

A Layer of Neurons with NumPy

Now we're back to the point where we'd like to calculate the output of a layer of 3 neurons, which means the weights will be a matrix or list of weight vectors. In plain Python, we wrote this as a list of lists. With NumPy, this will be a 2-dimensional array, which we'll call a matrix. Previously with the 3-neuron example, we performed a multiplication of those weights with a list containing inputs, which resulted in a list of output values — one per neuron.

We also described the dot product of two vectors, but the weights are now a matrix, and we need to perform a dot product of them and the input vector. NumPy makes this very easy for us — treating this matrix as a list of vectors and performing the dot product one by one with the vector of inputs, returning a list of dot products.

The dot product's result, in our case, is a vector (or a list) of sums of the weight and input products for each of the neurons. From here, we still need to add corresponding biases to them. The biases can be easily added to the result of the dot product operation as they are a vector of the same size. We can also use the plain Python list directly here, as NumPy will convert it to an array internally.

Previously, we had calculated outputs of each neuron by performing a dot product and adding a bias, one by one. Now we have changed the order of those operations — we're performing dot product first as one operation on all neurons and inputs, and then we are adding a bias in the next operation. When we add two vectors using NumPy, each i-th element is added together, resulting in a new vector of the same size. This is both a simplification and an optimization, giving us simpler and faster code.

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ĨĮĴŁĻĽ ł äĝgfld ğgfld ġgfld ğgĥā
ŃĦĨÍÎĻĽ ł ääflgğd flgíd hflgĥd ĝād
           äflgĥd hflgîĝd flgğħd hflgĥād
           ähflgğħd hflgğìd flgĝìd flgííāā
ĠĨĞĽĦĽ ł äğgfld ġgfld flgĥā


ĬĞÑĦĹçIJŁĻĴŁĻĽ ł ĮĴgĤIJĻâŃĦĨÍÎĻĽd ĨĮĴŁĻĽã κ ĠĨĞĽĦĽ

ĴĹĨĮĻâĬĞÑĦĹçIJŁĻĴŁĻĽã

ńńń
ĞĹĹĞÑâäġgí   ĝgğĝ  ğgġíĥāã
```

```
inputs = [1.0, 2.0, 3.0, 2.5]
weights = [[0.2, 0.8, -0.5, 1.0],
           [0.5, -0.91, 0.26, -0.5],
           [-0.26, -0.27, 0.17, 0.87]]
biases = [2.0, 3.0, 0.5]

outputs = np.dot(weights, inputs) + biases


np.dot(weights, inputs) = [np.dot(weights[0], inputs),
np.dot(weights[1], inputs), np.dot(weights[2], inputs)]
= [2.8, -1.79, 1.885]
```

https://nnfs.io

Fig 2.09: Code and visuals for the dot product applied to the layer of neurons.

```
inputs = [1.0, 2.0, 3.0, 2.5]
weights = [[0.2, 0.8, -0.5, 1.0],
           [0.5, -0.91, 0.26, -0.5],
           [-0.26, -0.27, 0.17, 0.87]]
biases = [2.0, 3.0, 0.5]

outputs = np.dot(weights, inputs) + biases
>>> array([4.8   1.21  2.385])
```

Fig 2.10: Code and visuals for the sum of the dot product and bias with a layer of neurons.

Anim 2.09-2.10: https://nnfs.io/cyx

This syntax involving the dot product of weights and inputs followed by the vector addition of bias is the most commonly used way to represent this calculation of *inputs·weights+bias*. To explain the order of parameters we are passing into *np.dot()*, we should think of it as whatever comes first will decide the output shape. In our case, we are passing a list of neuron weights first and then the inputs, as our goal is to get a list of neuron outputs. As we mentioned, a dot product of a matrix and a vector results in a list of dot products. The *np.dot()* method treats the matrix as a list of vectors and performs a dot product of each of those vectors with the other vector. In this example, we used that property to pass a matrix, which was a list of neuron weight vectors and a vector of inputs and get a list of dot products — neuron outputs.

A Batch of Data

To train, neural networks tend to receive data in **batches.** So far, the example input data have been only one sample (or **observation**) of various features called a feature set:

```
ĨĮĴŁĻĽ ł äĝd ğd ġd ğgĥā
```

Here, the `äĝd ğd ġd ğgĥā` data are somehow meaningful and descriptive to the output we desire. Imagine each number as a value from a different sensor, from the example in chapter 1, all simultaneously. Each of these values is a feature observation datum, and together they form a **feature set instance**, also called an **observation**, or most commonly, a **sample**.

Fig 2.11: Visualizing a 1D array.

Anim 2.11: https://nnfs.io/lqw

Often, neural networks expect to take in many **samples** at a time for two reasons. One reason is that it's faster to train in batches in parallel processing, and the other reason is that batches

help with generalization during training. If you fit (perform a step of a training process) on one sample at a time, you're highly likely to keep fitting to that individual sample, rather than slowly producing general tweaks to weights and biases that fit the entire dataset. Fitting or training in batches gives you a higher chance of making more meaningful changes to weights and biases. For the concept of fitment in batches rather than one sample at a time, the following animation can help:

Fig 2.12: Example of a linear equation fitting batches of 32 chosen samples. See animation below for other sizes of samples at a time to see how much of a difference batch size can make.

Anim 2.12: https://nnfs.io/vyu

An example of a batch of data could look like:

Input data :

```
batch = [[1,5,6,2],
         [3,2,1,3],
         [5,2,1,2],
         [6,4,8,4],
         [2,8,5,3],
         [1,1,9,4],
         [6,6,0,4],
         [8,7,6,4]]
```

Shape :

```
(8, 4)
```

Type :

2D Array, Matrix

https://nnfs.io

Fig 2.13: Example of a batch, its shape, and type.

Anim 2.13: https://nnfs.io/lqw

Recall that in Python, and in our case, lists are useful containers for holding a sample as well as multiple samples that make up a batch of observations. Such an example of a batch of observations, each with its own sample, looks like:

```
ĨĮĴŁĻĽ ł äägd ğd ġd ğghad äğd hd hgd gad ähgghd ggid ggġd hflgíaa
```

This list of lists could be made into an array since it is homologous. Note that each "list" in this larger list is a sample representing a feature set. `äĝd ğd ġd ğgĥā`, `äğd ĥd hĝd ğā`, and `ähĝgĥd ğgìd ġgġd hflgíā` are all **samples**, and are also referred to as **feature set instances** or **observations**.

We have a matrix of inputs and a matrix of weights now, and we need to perform the dot product on them somehow, but how and what will the result be? Similarly, as we performed a dot product on a matrix and a vector, we treated the matrix as a list of vectors, resulting in a list of dot products. In this example, we need to manage both matrices as lists of vectors and perform dot products on all of them in all combinations, resulting in a list of lists of outputs, or a matrix; this operation is called the **matrix product**.

Matrix Product

The **matrix product** is an operation in which we have 2 matrices, and we are performing dot products of all combinations of rows from the first matrix and the columns of the 2nd matrix, resulting in a matrix of those atomic **dot products**:

Fig 2.14: Visualizing how a single element in the resulting matrix from matrix product is calculated. See animation for the full calculation of each element.

Anim 2.14: https://nnfs.io/jei

To perform a matrix product, the size of the second dimension of the left matrix must match the size of the first dimension of the right matrix. For example, if the left matrix has a shape of *(5, 4)* then the right matrix must match this 4 within the first shape value *(4, 7)*. The shape of the resulting array is always the first dimension of the left array and the second dimension of the right array, *(5, 7)*. In the above example, the left matrix has a shape of *(5, 4)*, and the upper-right matrix has a shape of *(4, 5)*. The second dimension of the left array and the first dimension of the second array are both *4*, they match, and the resulting array has a shape of *(5, 5)*.

To elaborate, we can also show that we can perform the matrix product on vectors. In mathematics, we can have something called a column vector and row vector, which we'll explain better shortly. They're vectors, but represented as matrices with one of the dimensions having a size of 1:

$$a = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

$$b = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

a is a row vector. It looks very similar to a vector *a* (with an arrow above it) described earlier along with the vector product. The difference in notation between a row vector and vector are commas between values and the arrow above symbol *a* is missing on a row vector. It's called a row vector as it's a vector of a row of a matrix. *b*, on the other hand, is called a column vector because it's a column of a matrix. As row and column vectors are technically matrices, we do not denote them with vector arrows anymore.

When we perform the matrix product on them, the result becomes a matrix as well, like in the previous example, but containing just a single value, the same value as in the dot product example we have discussed previously:

$$ab = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 20 \end{bmatrix}$$

Fig 2.15: Product of row and column vectors.

Anim 2.15: https://nnfs.io/bkw

In other words, row and column vectors are matrices with one of their dimensions being of a size of 1; and, we perform the **matrix product** on them instead of the **dot product**, which results in a matrix containing a single value. In this case, we performed a matrix multiplication of matrices with shapes *(1, 3)* and *(3, 1)*, then the resulting array has the shape *(1, 1)* or a size of *1x1*.

Transposition for the Matrix Product

How did we suddenly go from 2 vectors to row and column vectors? We used the relation of the dot product and matrix product saying that a dot product of two vectors equals a matrix product of a row and column vector (the arrows above the letters signify that they are vectors):

$$\vec{a} \cdot \vec{b} = ab^T$$

We also have temporarily used some simplification, not showing that column vector *b* is actually a **transposed** vector *b*. The proper equation, matching the dot product of vectors *a* and *b* written as matrix product should look like:

$$ab^T = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 20 \end{bmatrix}$$

Here we introduced one more new operation — **transposition**. Transposition simply modifies a matrix in a way that its rows become columns and columns become rows:

Fig 2.16: Example of an array transposition.

Anim 2.16: https://nnfs.io/qut

$$\begin{bmatrix} 0.49 & 0.97 & 0.53 & 0.05 & 0.33 \\ 0.65 & 0.62 & 0.51 & 1.00 & 0.38 \\ 0.61 & 0.45 & 0.74 & 0.27 & 0.64 \\ 0.17 & 0.36 & 0.17 & 0.96 & 0.12 \\ 0.79 & 0.32 & 0.68 & 0.90 & 0.77 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0.49 & 0.65 & 0.61 & 0.17 & 0.79 \\ 0.97 & 0.62 & 0.45 & 0.36 & 0.32 \\ 0.53 & 0.51 & 0.74 & 0.17 & 0.68 \\ 0.05 & 1.00 & 0.27 & 0.96 & 0.90 \\ 0.33 & 0.38 & 0.64 & 0.12 & 0.77 \end{bmatrix}$$

https://nnfs.io

Fig 2.17: Another example of an array transposition.

Anim 2.17: https://nnfs.io/pnq

Now we need to get back to row and column vector definitions and update them with what we have just learned.

A row vector is a matrix whose first dimension's size (the number of rows) equals 1 and the second dimension's size (the number of columns) equals *n* — the vector size. In other words, it's a 1×n array or array of shape (1, n):

$$a = \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_n \end{bmatrix}$$

With NumPy and with 3 values, we would define it as:

```
ĮĴgĞĹĹĞÑâäägd ğd ġāāã
```

Note the use of double brackets here. To transform a list into a matrix containing a single row (perform an equivalent operation of turning a vector into row vector), we can put it into a list and create numpy array:

```
Ğ ł äĝd ğd ġā
ĮĴgĞĹĹĞÑâäĞāã

ńńń
ĞĹĹĞÑâäägd ğd ġāāã
```

Again, note that we encase Ğ in brackets before converting to an array in this case.
Or we can turn it into a 1D array and expand dimensions using one of the NumPy abilities:

```
Ğ ł äĝd ğd ġā
ĮĴgĦŇĴĞĮĤçĤĨİĹâĮĴgĞĹĹĞÑâĞãd ĞŇĨĹłflã

ńńń
ĞĹĹĞÑâäägd ğd ġāāã
```

Where *np.expand_dims()* adds a new dimension at the index of the *axis*.
A column vector is a matrix where the second dimension's size equals 1, in other words, it's an array of shape (n, 1):

$$b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{bmatrix}$$

With NumPy it can be created the same way as a row vector, but needs to be additionally transposed — transposition turns rows into columns and columns into rows:

$$\begin{bmatrix} b_1 & b_2 & b_3 & \dots & b_n \end{bmatrix}^T = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{bmatrix}$$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{bmatrix}^T = \begin{bmatrix} b_1 & b_2 & b_3 & \dots & b_n \end{bmatrix}$$

To turn vector *b* into row vector *b*, we'll use the same method that we used to turn vector *a* into row vector *a*, then we can perform a transposition on it to make it a column vector *b*:

$$b = \begin{bmatrix} 2 & 3 & 4 \end{bmatrix}$$

$$b^T = \begin{bmatrix} 2 & 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

With NumPy code:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

Ğ ł äĝd ğd ġā
Ġ ł äğd ġd ǵā

Ğ ł ĮĴgĞĹĹĞÑâäĞāã
Ġ ł ĮĴgĞĹĹĞÑâäĠāãgT


ĮĴgĤIJĻâĞd Ġã

ńńń
ĞĹĹĞÑâäägflāāã
```

We have achieved the same result as the dot product of two vectors, but performed on matrices and returning a matrix — exactly what we expected and wanted. It's worth mentioning that NumPy does not have a dedicated method for performing matrix product — the dot product and matrix product are both implemented in a single method: *np.dot()*.

As we can see, to perform a matrix product on two vectors, we took one as is, transforming it into a row vector, and the second one using transposition on it to turn it into a column vector. That allowed us to perform a matrix product that returned a matrix containing a single value. We also performed the matrix product on two example arrays to learn how a matrix product works — it creates a matrix of dot products of all combinations of row and column vectors.

A Layer of Neurons & Batch of Data w/ NumPy

Let's get back to our inputs and weights — when covering them, we mentioned that we need to perform dot products on all of the vectors that consist of both input and weight matrices. As we have just learned, that's the operation that the matrix product performs. We just need to perform transposition on its second argument, which is the weights matrix in our case, to turn the row vectors it currently consists of into column vectors.

Initially, we were able to perform the dot product on the inputs and the weights without a transposition because the weights were a matrix, but the inputs were just a vector. In this case, the dot product results in a vector of atomic dot products performed on each row from the matrix and this single vector. When inputs become a batch of inputs (a matrix), we need to perform the matrix product. It takes all of the combinations of rows from the left matrix and columns from the right matrix, performing the dot product on them and placing the results in an output array. Both arrays have the same shape, but, to perform the matrix product, the shape's value from the index 1 of the first matrix and the index 0 of the second matrix must match — they don't right now.

```
inputs = [[1.0, 2.0, 3.0, 2.5],
          [2.0, 5.0, -1.0, 2.0],
          [-1.5, 2.7, 3.3, -0.8]]
weights = [[0.2, 0.8, -0.5, 1.0],
           [0.5, -0.91, 0.26, -0.5],
           [-0.26, -0.27, 0.17, 0.87]]
```

Fig 2.18: Depiction of why we need to transpose to perform the matrix product.

If we transpose the second array, values of its shape swap their positions.

Fig 2.19: After transposition, we can perform the matrix product.

Anim 2.18-2.19: https://nnfs.io/crq

If we look at this from the perspective of the input and weights, we need to perform the dot product of each input and each weight set in all of their combinations. The dot product takes the row from the first array and the column from the second one, but currently the data in both arrays are row-aligned. Transposing the second array shapes the data to be column-aligned. The matrix product of inputs and transposed weights will result in a matrix containing all atomic dot products that we need to calculate. The resulting matrix consists of outputs of all neurons after operations performed on each input sample:

```
inputs = [[1.0, 2.0, 3.0, 2.5],
          [2.0, 5.0, -1.0, 2.0],
          [-1.5, 2.7, 3.3, -0.8]]
weights = [[0.2, 0.8, -0.5, 1.0],
           [0.5, -0.91, 0.26, -0.5],
           [-0.26, -0.27, 0.17, 0.87]]
biases = [2.0, 3.0, 0.5]

outputs = np.dot(inputs, np.array(weights).T) + biases

np.dot(inputs, np.array(weights).T)
```

https://nnfs.io

$$\begin{bmatrix} 1.0 & 2.0 & 3.0 & 2.5 \\ 2.0 & 5.0 & -1.0 & 2.0 \\ -1.5 & 2.7 & 3.3 & -0.8 \end{bmatrix} \begin{bmatrix} 0.2 & 0.5 & -0.26 \\ 0.8 & -0.91 & -0.27 \\ -0.5 & 0.26 & 0.17 \\ 1.0 & -0.5 & 0.87 \end{bmatrix} = \begin{bmatrix} 2.8 & -1.79 & 1.885 \\ 6.9 & -4.81 & -0.3 \\ -0.59 & -1.949 & -0.474 \end{bmatrix}$$

Fig 2.20: Code and visuals depicting the dot product of inputs and transposed weights.

Anim 2.20: https://nnfs.io/gjw

We mentioned that the second argument for *np.dot()* is going to be our transposed weights, so first will be inputs, but previously weights were the first parameter. We changed that here. Before, we were modeling neuron output using a single sample of data, a vector, but now we are a step forward when we model layer behavior on a batch of data. We could retain the current parameter order, but, as we'll soon learn, it's more useful to have a result consisting of a list of layer outputs per each sample than a list of neurons and their outputs sample-wise. We want the resulting array to be sample-related and not neuron-related as we'll pass those samples further through the network, and the next layer will expect a batch of inputs.

We can code this solution using NumPy now. We can perform *np.dot()* on a plain Python list of lists as NumPy will convert them to matrices internally. We are converting weights ourselves though to perform transposition operation first, *T* in the code, as plain Python list of lists does not support it. Speaking of biases, we do not need to make it a NumPy array for the same reason — NumPy is going to do that internally.

Biases are a list, though, so they are a 1D array as a NumPy array. The addition of this bias vector to a matrix (of the dot products in this case) works similarly to the dot product of a matrix and vector that we described earlier; The bias vector will be added to each row vector of the matrix. Since each column of the matrix product result is an output of one neuron, and the vector is going to be added to each row vector, the first bias is going to be added to each first element of those vectors, second to second, etc. That's what we need — the bias of each neuron needs to be added to all of the results of this neuron performed on all input vectors (samples).

```
inputs = [[1.0, 2.0, 3.0, 2.5],
          [2.0, 5.0, -1.0, 2.0],
          [-1.5, 2.7, 3.3, -0.8]]
weights = [[0.2, 0.8, -0.5, 1.0],
           [0.5, -0.91, 0.26, -0.5],
           [-0.26, -0.27, 0.17, 0.87]]
biases = [2.0, 3.0, 0.5]

outputs = np.dot(inputs, np.array(weights).T) + biases
>>> array([[ 4.8    1.21   2.385],
           [ 8.9   -1.81   0.2  ],
           [ 1.41   1.051  0.026]])
```

https://nnfs.io

$$\begin{bmatrix} 2.8 & -1.79 & 1.885 \\ 6.9 & -4.81 & -0.3 \\ -0.59 & -1.949 & -0.474 \end{bmatrix} + \begin{bmatrix} 2.0 & 3.0 & 0.5 \end{bmatrix} = \begin{bmatrix} 4.8 & 1.21 & 2.385 \\ 8.9 & -1.81 & 0.2 \\ 1.41 & 1.051 & 0.026 \end{bmatrix}$$

Fig 2.21: Code and visuals for inputs multiplied by the weights, plus the bias.

Anim 2.21: https://nnfs.io/qty

Now we can implement what we have learned into code:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ


ĨĮĴŁĻĽ ł äägĝgfld ğgfld ġgfld ğgĥād
          äğgfld ĥgfld hĝgfld ğgflād
          ähĝgĥd ğgìd ġgġd hflgíāā
ŃĦĨÍÎĻĽ ł äāflgğd flgíd hflgĥd ĝgflād
           äflgĥd hflgîĝd flgğħd hflgĥād
           āhflgğħd hflgğìd flgĝìd flgíìāā
ĠĨĞĽĦĽ ł äğgfld ġgfld flgĥā

ĬĞÑĦĹçIJŁĻĴŁĻĽ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĮĴgĞĹĹĞÑâŃĦĨÍÎĻĽãgTã κ ĠĨĞĽĦĽ

ĴĹĨĮĻâĬĞÑĦĹçIJŁĻĴŁĻĽã

ńńń
ĞĹĹĞÑâää ġgí    ĝgğĝ   ğgġíĥād
       ä ígî   hĝgíĝ   flgğ  ād
       ä ĝgġĝ   ĝgflĥĝ  flgflğħāāã
```

As you can see, our neural network takes in a group of samples (inputs) and outputs a group of predictions. If you've used any of the deep learning libraries, this is why you pass in a list of inputs (even if it's just one feature set) and are returned a list of predictions, even if there's only one prediction.

Supplementary Material: https://nnfs.io/ch2
Chapter code, further resources, and errata for this chapter.

Chapter 3

Adding Layers

The neural network we've built is becoming more respectable, but at the moment, we have only one layer. Neural networks become "deep" when they have 2 or more **hidden layers**. At the moment, we have just one layer, which is effectively an output layer. Why we want two or more **hidden** layers will become apparent in a later chapter. Currently, we have no hidden layers. A hidden layer isn't an input or output layer; as the scientist, you see data as they are handed to the input layer and the resulting data from the output layer. Layers between these endpoints have values that we don't necessarily deal with, hence the name "hidden." Don't let this name convince you that you can't access these values, though. You will often use them to diagnose issues or improve your neural network. To explore this concept, let's add another layer to this neural network, and, for now, let's assume these two layers that we're going to have will be the hidden layers, and we just have not coded our output layer yet.

Before we add another layer, let's think about what will be coming. In the case of the first layer, we can see that we have an input with 4 features.

Fig 3.01: Input layer with 4 features into a hidden layer with 3 neurons.

Samples (feature set data) get fed through the input, which does not change it in any way, to our first hidden layer, which we can see has 3 sets of weights, with 4 values each.

Each of those 3 unique weight sets is associated with its distinct neuron. Thus, since we have 3 weight sets, we have 3 neurons in this first hidden layer. Each neuron has a unique set of weights, of which we have 4 (as there are 4 inputs to this layer), which is why our initial weights have a shape of *(3,4)*.

Now, we wish to add another layer. To do that, we must make sure that the expected input to that layer matches the previous layer's output. We have set the number of neurons in a layer by setting how many weight sets and biases we have. The previous layer's influence on weight sets for the current layer is that each weight set needs to have a separate weight per input. This means a distinct weight per neuron from the previous layer (or feature if we're talking the input). The previous layer has 3 weight sets and 3 biases, so we know it has 3 neurons. This then means, for the next layer, we can have as many weight sets as we want (because this is how many neurons this new layer will have), but each of those weight sets must have 3 discrete weights.

To create this new layer, we are going to copy and paste our ŃĦĨÍÎĻĽ and ĠĨĞĽĦĽ to ŃĦĨÍÎĻĽğ and ĠĨĞĽĦĽğ, and change their values to new made up sets. Here's an example:

```
ĨĮĴŁĻĽ ł ääĝd ğd ġd ğgĥād
          äğgd ĥgd hĝgd ğād
          ähĝgĥd ğgìd ġgġd hflgíāā
ŃĦĨÍÎĻĽ ł ääflgğd flgíd hflgĥd ĝād
           äflgĥd hflgîĝd flgğħd hflgĥād
           ähflgğħd hflgğìd flgĝìd flgíìāā
ĠĨĞĽĦĽ ł äğd ġd flgĥā
```

```
ŃĦĨÍÎĻĽğ ł ääflgĝd hflgĝǵd flgĥād
           ähflgĥd flgĝğd hflgġġād
           ähflgǵǵd flgìġd hflgĝġāā
ĠĨĞĽĦĽğ ł ähĝd ğd hflgĥā
```

Next, we will now call *outputs* "*layer1_ouputs*"*:*

```
ĬĞÑĦĹĝçIJŁĻĴŁĻĽ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĮĴgĞĹĹĞÑâŃĦĨÍÎĻĽãgTã κ ĠĨĞĽĦĽ
```

As previously stated, inputs to layers are either inputs from the actual dataset you're training with or outputs from a previous layer. That's why we defined 2 versions of *weights* and *biases* but only 1 of *inputs* — because the inputs for layer 2 will be the outputs from the previous layer:

```
ĬĞÑĦĹğçIJŁĻĴŁĻĽ ł ĮĴgĤIJĻâĬĞÑĦĹĝçIJŁĻĴŁĻĽd ĮĴgĞĹĹĞÑâŃĦĨÍÎĻĽğãgTã κ ċ
                  ĠĨĞĽĦĽğ
```

All together now:

```
ĨĬĴIJĹĻ ĮŁĬĴÑ ĞĽ ĮĴ

ĨĮĴŁĻĽ ł äägd ğd ġd ğghād äğgd ĥgd hĝgd ğād ähĝghd ğgìd ġgġd hflgíāā
ŃĦĨÍÎĻĽ ł äägğd flgíd hflghd ĝād
           äflghd hflgîĝd flgğhd hflghād
           ähflgğhd hflgğìd flgĝìd flgíìāā
ĠĨĞĽĦĽ ł äğd ġd flghā
ŃĦĨÍÎĻĽğ ł äägĝd hflgĝǵd flghād
            ähflghd flgĝğd hflgġġād
            ähflgǵǵd flgìġd hflgĝġāā
ĠĨĞĽĦĽğ ł ähĝd ğd hflghā

ĬĞÑĦĹĝçIJŁĻĴŁĻĽ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĮĴgĞĹĹĞÑâŃĦĨÍÎĻĽãgTã κ ĠĨĞĽĦĽ
ĬĞÑĦĹğçIJŁĻĴŁĻĽ ł ĮĴgĤIJĻâĬĞÑĦĹĝçIJŁĻĴŁĻĽd ĮĴgĞĹĹĞÑâŃĦĨÍÎĻĽğãgTã κ ĠĨĞĽĦĽğ

ĴĹĨĮĻâĬĞÑĦĹğçIJŁĻĴŁĻĽã


ńńń
ĞĹĹĞÑâää flghflġĝ  hĝgflġĝíĥ hğgflġíìĥād
       ä flgğġġ  hğgìġğ  hĥgìħġġ ād
       ähflgîîġĝġ  ĝgġĝğĥġ hflgġĥħĥĥāāã
```

At this point, our neural network could be visually represented as:

Fig 3.02: 4 features input into 2 hidden layers of 3 neurons each.

Training Data

Next, rather than hand-typing in random data, we'll use a function that can create non-linear data. What do we mean by non-linear? Linear data can be fit with or represented by a straight line.

Fig 3.03: Example of data (orange dots) that can be represented (fit) by a straight line (green line).

Non-linear data cannot be represented well by a straight line.

Fig 3.04: Example of data (orange dots) that is not well fit by a straight line.

If you were to graph data points of the form *(x, y)* where *y = f(x)*, and it looks to be a line with a clear trend or slope, then chances are, they're linear data! Linear data are very easily approximated by far simpler machine learning models than neural networks. What other machine learning algorithms cannot approximate so easily are non-linear datasets. To simplify this, we've created a Python package that you can install with pip, called *nnfs*:

```
ĴĨĴ ĨĮĽĻĞĬĬ ĮĮÌĽ
```

The nnfs package contains functions that we can use to create data. For example:

```
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĨİĴIJĹĻ ĽĴĨĹĞĬçĤĞĻĞ
```

The *spiral_data* function was slightly modified from https://cs231n.github.io/neural-networks-case-study/, which is a great supplementary resource for this topic.

You will typically not be generating training data from a function for your neural networks. You will have an actual dataset. Generating a dataset this way is purely for convenience at this stage. We will also use this package to ensure repeatability for everyone, using `ĮĮÌĽgĨĮĨĻâã`, after importing NumPy:

```
ĨİĴIJĹĻ ĮłİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ

ĮĮÌĽgĨĮĨĻâã
```

The ĮĮÌĽgĨĮĨĮâã does three things: it sets the random seed to 0 (by the default), creates a *float32* dtype default, and overrides the original dot product from NumPy. All of these are meant to ensure repeatable results for following along.

The ĽĴĨĹĞĬçĤĞĻĞ function allows us to create a dataset with as many classes as we want. The function has parameters to choose the number of classes and the number of points/observations per class in the resulting non-linear dataset. For example:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ

Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

ĴĬĻgĽĢĞĻĻĦĹâXäfdflād Xäfdĝāā
ĴĬĻgĽÎIJŃâã
```

Fig 3.05: Uncolored spiral dataset.

If you trace from the center, you can determine all 3 classes separately, but this is a very challenging problem for a machine learning classifier to solve. Adding color to the chart makes this more clear:

```
ĴĬĻgĽĢĞĻĻĦĹâXäfd flād Xäfd ĝād ĢłÑd ĢİĞĴłďĠĹÍďã
ĴĬĻgĽÎIJŃâã
```

Fig 3.06: Spiral dataset colored by class.

Keep in mind that the neural network will not be aware of the color differences as the data have no class encodings. This is only made as an instruction for the reader. In the data above, each dot is the feature, and its coordinates are the samples that form the dataset. The “classification” for that dot has to do with which spiral it is a part of, depicted by blue, green, or red color in the previous image. These colors would then be assigned a class number for the model to fit to, like 0, 1, and 2.

Dense Layer Class

Now that we no longer need to hand-type our data, we should create something similar for our various types of neural network layers. So far, we've only used what's called a **dense** or **fully-connected** layer. These layers are commonly referred to as "dense" layers in papers, literature, and code, but you will occasionally see them called fully-connected or "fc" for short in code. Our dense layer class will begin with two methods.

```
class LayerDense:

    def __init__(ccinit, inputs, neurons):
        self.weights = np.random.randn(inputs, neurons)
        self.biases = np.zero(neurons)

    # Forward pass
    def forward(ccself, inputs):
        self.output = np.dot(inputs, self.weights) + self.biases
```

As previously stated, weights are often initialized randomly for a model, but not always. If you wish to load a pre-trained model, you will initialize the parameters to whatever that pretrained model finished with. It's also possible that, even for a new model, you have some other initialization rules besides random. For now, we'll stick with random initialization. Next, we have the *forward* method. When we pass data through a model from beginning to end, this is called a **forward pass**. Just like everything else, however, this is not the only way to do things. You can have the data loop back around and do other interesting things. We'll keep it usual and perform a regular forward pass.

To continue the `LayerDense` class' code let's add the random initialization of weights and biases:

```
# Layer initialization
def __init__(ccself, inputs, neurons):
    self.weights = np.random.randn(inputs, neurons)
    self.biases = np.zero(neurons)
```

Here, we're setting weights to be random and biases to be 0. Note that we're initializing weights to be *(inputs, neurons),* rather than (*neurons, inputs)*. We're doing this ahead instead of transposing every time we perform a forward pass, as explained in the previous chapter. Why zero biases? In specific scenarios, like with many samples containing values of 0, a bias can ensure that a neuron fires initially. It sometimes may be appropriate to initialize the biases to some non-zero number, but the most common initialization for biases is 0. However, in these scenarios, you may find success in doing things another way. This will vary depending on your use-case and is just one of many things you can tweak when trying to improve results. One situation where you might want to try something else is with what's called **dead neurons**. We haven't yet covered activation functions in practice, but imagine our step function again.

Fig 3.07: Graph of a step function.

It's possible for *weights · inputs + biases* not to meet the threshold of the step function, which means the neuron will output a 0. Alone, this is not a big issue, but it becomes a problem if this happens to this neuron for every one of the input samples (it'll become clear why once we cover backpropagation). So then this neuron's 0 output is the input to another neuron. Any weight multiplied by zero will be zero. With an increasing number of neurons outputting 0, more inputs to the next neurons will receive these 0s rendering the network essentially non-trainable, or "dead."

Next, let's explore `ĮĴgĹĞĮĤIJİgĹĞĮĤĮ` and `ĮĴgŅĦĹIJĽ` in more detail. These methods are convenient ways to initialize arrays. `ĮĴgĹĞĮĤIJİgĹĞĮĤĮ` produces a Gaussian distribution with a mean of 0 and a variance of 1, which means that it'll generate random numbers, positive and negative, centered at 0 and with the mean value close to 0. In general, neural networks work best with values between -1 and +1, which we'll discuss in an upcoming chapter. So this `ĮĴgĹĞĮĤIJİgĹĞĮĤĮ` generates values around those numbers. We're going to multiply this Gaussian distribution for the weights by *0.01* to generate numbers that are a couple of magnitudes smaller. Otherwise, the model will take more time to fit the data during the training process as starting values will be disproportionately large compared to the updates being made

during training. The idea here is to start a model with non-zero values small enough that they won't affect training. This way, we have a bunch of values to begin working with, but hopefully none too large or as zeros. You can experiment with values other than *0.01* if you like.

Finally, the `ĮĴgĹĞĮĤIJİgĹĞĮĤĮ` function takes dimension sizes as parameters and creates the output array with this shape. The weights here will be the number of inputs for the first dimension and the number of neurons for the 2nd dimension. This is similar to our previous made up array of weights, just randomly generated. Whenever there's a function or block of code that you're not sure about, you can always print it out. For example:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ

ĮĮÌĽgĨĮĨĮâã

ĴĹĨĮĻâĮĴgĹĞĮĤIJİgĹĞĮĤĮâğdĥãã


ńńń
ää ĝgìħǵflĥğǵ   flǵǵflflĝĥìğ   flgîìíìġí    ğgğǵflíîġĝ   ĝgíħìĥĥí  ā
 ähflgîììğììî   flgîĥflflííǵǵ hflgĝĥĝġĥìğ  hflgĝflġğĝííĥ  flgǵĝflĥîíĥğāā
```

The example function call has returned a 2x5 array (which we can also say is "*with a shape of (2,5)*") with data randomly sampled from a Gaussian distribution with a mean of 0.

Next, the `ĮĴgŅĦĹIJĽ` function takes a desired array shape as an argument and returns an array of that shape filled with zeros.

```
ĴĹĨĮĻâĮĴgŅĦĹIJĽââğdĥāāā


ńńń
ääflg  flg  flg  flg  flgā
 äflg  flg  flg  flg  flgāā
```

We'll initialize the biases with the shape of *(1, n_neurons)*, as a row vector, which will let us easily add it to the result of the dot product later, without additional operations like transposition.

To see an example of how our method initializes weights and biases:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ

ĮĮÌĽgĨĮĨĻâã

ĮçĨĮĴŁĻĽ ł ğ
ĮçĮĦŁĹIJĮĽ ł ǵ

ŃĦĨÍÎĻĽ ł flgflĝ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
ĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã

ĴĹĨĮĻâŃĦĨÍÎĻĽã
ĴĹĨĮĻâĠĨĞĽĦĽã


ńńń
ää flgflĝìħǵflĥğ  flgflflǵflflĝĥì  flgflflîìíìġí  flgflğğǵflííġā
 ä flgflĝíħìĥĥí hflgflflîììğìí  flgflflîĥflflíí hflgflflĝĥĝǵĥìāā
ääflg flg flg flgāā
```

On to our forward method — we need to update it with the dot product+biases calculation:

```
ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
    ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĽĦĬÌgŃĦĨÍÎĻĽã к ĽĦĬÌgĠĨĞĽĦĽ
```

Nothing new here, just turning the previous code into a method. Our full *Layer_Dense* class so far:

```
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽãf
        ĽĦĬÌgŃĦĨÍÎĻĽ ł flgflĝ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã

    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĽĦĬÌgŃĦĨÍÎĻĽã к ĽĦĬÌgĠĨĞĽĦĽ
```

We're ready to make use of this new class instead of hardcoded calculations, so let's generate some data using the discussed dataset creation method and use our new layer to perform a forward pass:

```
# CĹĦĞĻĦ ĤĞĻĞĹĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflld ĢĬĞĽĽĦĽłġã

# CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴłĻ ÌĦĞĻłĹĦĽ ĞĮĤ ġ IJłĻĴłĻ ŁĞĬłĦĽ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ġã

# PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ IJÌ IJłĹ ĻĹĞĨĮĨĮÍ ĤĞĻĞ ĻÎĹIJłÍÎ ĻÎĨĽ ĬĞÑĦĹ
ĤĦĮĽĦĝgÌIJĹŃĞĹĤâXã

# LĦĻďĽ ĽĦĦ IJłĻĴłĻ IJÌ ĻÎĦ ÌĨĹĽĻ ÌĦŃ ĽĞİĴĬĦĽf
```

ĴĹĨĮĻâĤĦĮĽĦĝgIJłĻĴłĻäfĥãã Go ahead and run everything.

Full code up to this point:

```
ĨİĴIJĹĻ ĮłİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĨİĴIJĹĻ ĽĴĨĹĞĬçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


# DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    # LĞÑĦĹ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴłĻĽd ĮçĮĦłĹIJĮĽãf
        # IĮĨĻĨĞĬĨŅĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍÎĻĽ ł flgflĝ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĮçĨĮĴłĻĽd ĮçĮĦłĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦłĹIJĮĽãã
```

```
    fi FIJĹŃĞĹÎ ĴĞĽĽ
    ÎHÌ ÌIJĹŃĞĹÎâĽHĬÌd ĨĮĴŁĻĽãf
        fi CĞĬĢŁĬĞĻH IJŁĻĴŁĻ ŁĠĬŁHĽ ÌĹIJİ ĨĮĴŁĻĽd ŃHĨÍÎĻĽ ĞĮÎ ĠĨĞĽHĽ
        ĽHĬÌgIJŁĻĴŁĻ ł ĮĴgÎIJĻâĨĮĴŁĻĽd ĽHĬÌgŃHĨÍÎĻĽã κ ĽHĬÌgĠĨĞĽHĽ


fi CĹHĞĻH ÎĞĻĞĽHĻ
Xd Ñ ł ĽĴĨĹĞĬçÎĞĻĞâĽĞĬĴĬHĽłĝflfld ĢĬĞĽĽHĽłġã

fi CĹHĞĻH DHĮĽH ĬĞÑHĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌHĞĻŁĹHĽ ĞĮÎ ġ IJŁĻĴŁĻ ŁĠĬŁHĽ
ÎHĮĽHĝ ł LĞÑHĹçDHĮĽHâğd ġã

fi PHĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ÎĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑHĹ
ÎHĮĽHĝgÌIJĹŃĞĹÎâXã

fi LHĻďĽ ĽHH IJŁĻĴŁĻ IJÌ ĻÎH ÌĨĹĽĻ ÌHŃ ĽĞĬĴĬHĽf
ĴĹĨĮĻâÎHĮĽHĝgIJŁĻĴŁĻäfĥãã


ńńń
ää flgflflflflflflflHκflfl  flgflflflflflflflHκflfl  flgflflflflflflflHκflflã
 ähĝgflġìĥĝííHhflġ  ĝgĝġîĥġĥĝHhflġ hġgìîíġĥflflHhflĥã
 ähğgìġĝġíġğHhflġ  ġgĝìğîĝĥflHhflġ hígĥîğĝìîíHhflĥã
 ähġgğĝííġĥĥHhflġ  ĥgğĥĥĥğĥìHhflġ hĥgĥîĝğĥíğHhflĥã
 ähĥgììflìĥíflHhflġ  ìgĝġflĝġflĥHhflġ hígîġġflġġîHhflĥãã
```

In the output, you can see we have 5 rows of data that have 3 values each. Each of those 3 values is the value from the 3 neurons in the *dense1* layer after passing in each of the samples. Great! We have a network of neurons, so our neural network model is almost deserving of its name, but we're still missing the activation functions, so let's do those next!

Supplementary Material: https://nnfs.io/ch3
Chapter code, further resources, and errata for this chapter.

Chapter 4

Activation Functions

In this chapter, we will tackle a few of the activation functions and discuss their roles. We use different activation functions for different cases, and understanding how they work can help you properly pick which of them is best for your task. The activation function is applied to the output of a neuron (or layer of neurons), which modifies outputs. We use activation functions because if the activation function itself is nonlinear, it allows for neural networks with usually two or more hidden layers to map nonlinear functions. We'll be showing how this works in this chapter.

In general, your neural network will have two types of activation functions. The first will be the activation function used in hidden layers, and the second will be used in the output layer. Usually, the activation function used for hidden neurons will be the same for all of them, but it doesn't have to.

The Step Activation Function

Recall the purpose this activation function serves is to mimic a neuron "firing" or "not firing" based on input information. The simplest version of this is a step function. In a single neuron, if the *weights* · *inputs* + *bias* results in a value greater than 0, the neuron will fire and output a 1; otherwise, it will output a 0.

Fig 4.01: Step function graph.

This activation function has been used historically in hidden layers, but nowadays, it is rarely a choice.

The Linear Activation Function

A linear function is simply the equation of a line. It will appear as a straight line when graphed, where y=x and the output value equals the input.

Fig 4.02: Linear function graph.

This activation function is usually applied to the last layer's output in the case of a regression model — a model that outputs a scalar value instead of a classification. We'll cover regression in chapter 17 and soon in an example in this chapter.

The Sigmoid Activation Function

The problem with the step function is it's not very informative. When we get to training and network optimizers, you will see that the way an optimizer works is by assessing individual impacts that weights and biases have on a network's output. The problem with a step function is that it's less clear to the optimizer what these impacts are because there's very little information gathered from this function. It's either on (1) or off (0). It's hard to tell how "close" this step function was to activating or deactivating. Maybe it was very close, or maybe it was very far. In terms of the final output value from the network, it doesn't matter if it was *close* to outputting something else. Thus, when it comes time to optimize weights and biases, it's easier for the optimizer if we have activation functions that are more granular and informative.

The original, more granular, activation function used for neural networks was the **Sigmoid** activation function, which looks like:

Fig 4.03: Sigmoid function graph.

This function returns a value in the range of 0 for negative infinity, through 0.5 for the input of 0, and to 1 for positive infinity. We'll talk about this function more in chapter 16.

As mentioned earlier, with "dead neurons," it's usually better to have a more granular approach for the hidden neuron activation functions. In this case, we're getting a value that can be reversed to its original value; the returned value contains all the information from the input, contrary to a function like the step function, where an input of 3 will output the same value as an input of 300,000. The output from the Sigmoid function, being in the range of 0 to 1, also works better with neural networks — especially compared to the range of the negative to the positive infinity — and adds nonlinearity. The importance of nonlinearity will become more clear shortly in this chapter. The Sigmoid function, historically used in hidden layers, was eventually replaced by the **Rectified Linear Units** activation function (or **ReLU**). That said, we will be using the Sigmoid function as the output layer's activation function in chapter 16.

The Rectified Linear Activation Function

Fig 4.04: Graph of the ReLU activation function.

The rectified linear activation function is simpler than the sigmoid. It's quite literally $y=x$, clipped at 0 from the negative side. If x is less than or equal to 0, then y is 0 — otherwise, y is equal to x.

$$y = \begin{cases} x & x > 0 \\ 0 & x \leqslant 0 \end{cases}$$

This simple yet powerful activation function is the most widely used activation function at the time of writing for various reasons — mainly speed and efficiency. While the sigmoid activation function isn't the most complicated, it's still much more challenging to compute than the ReLU activation function. The ReLU activation function is extremely close to being a linear activation function while remaining nonlinear, due to that bend after 0. This simple property is, however, very effective.

Why Use Activation Functions?

Now that we understand what activation functions represent, how some of them look, and what they return, let's discuss *why* we use activation functions in the first place. In most cases, for a neural network to fit a nonlinear function, we need it to contain two or more hidden layers, and we need those hidden layers to use a nonlinear activation function.

First off, what's a nonlinear function? A nonlinear function cannot be represented well by a straight line, such as a sine function:

Fig 4.05: Graph of y=sin(x)

While there are certainly problems in life that are linear in nature, for example, trying to figure out the cost of some number of shirts, and we know the cost of an individual shirt, and that there are no bulk discounts, then the equation to calculate the price of any number of those products is a linear equation. Other problems in life are not so simple, like the price of a home. The number of factors that come into play, such as size, location, time of year attempting to sell, number of rooms, yard, neighborhood, and so on, makes the pricing of a home a nonlinear equation. Many of the more interesting and hard problems of our time are nonlinear. The main attraction for neural networks has to do with their ability to solve nonlinear problems. First, let's consider a situation where neurons have no activation function, which would be the same as having an activation function of $y=x$. With this linear activation function in a neural network with 2 hidden layers of 8 neurons each, the result of training this model will look like:

Fig 4.06: Neural network with linear activation functions in hidden layers attempting to fit y=sin(x)

When using the same 2 hidden layers of 8 neurons each with the rectified linear activation function, we see the following result after training:

Fig 4.07: ReLU activation functions in hidden layers attempting to fit y=sin(x)

Linear Activation in the Hidden Layers

Now that you can see that this is the case, we still should consider *why* this is the case. To begin, let's revisit the linear activation function of $y=x$, and let's consider this on a singular neuron level. Given values for weights and biases, what will the output be for a neuron with a $y=x$ activation function? Let's look at some examples — first, let's try to update the first weight with a positive value:

Fig 4.08: Example of output with a neuron using a linear activation function.

As we continue to tweak with weights, updating with a negative number this time:

Fig 4.09: Example of output with a neuron using a linear activation function, updated weight.

And updating weights and additionally a bias:

Fig 4.10: Example of output with a neuron using a linear activation function, updated another weight.

No matter what we do with this neuron's weights and biases, the output of this neuron will be perfectly linear to $y=x$ of the activation function. This linear nature will continue throughout the entire network:

Fig 4.11: A neural network with all linear activation functions.

No matter what we do, however many layers we have, this network can only depict linear relationships if we use linear activation functions. It should be fairly obvious that this will be the case as each neuron in each layer acts linearly, so the entire network is a linear function as well.

ReLU Activation in a Pair of Neurons

We believe it is less obvious how, with a barely nonlinear activation function, like the rectified linear activation function, we can suddenly map nonlinear relationships and functions, so now let's cover that. Let's start again with a single neuron. We'll begin with both a weight of 0 and a bias of 0:

Fig 4.12: Single neuron with single input (zeroed weight) and ReLU activation function.

In this case, no matter what input we pass, the output of this neuron will always be a 0, because the weight is 0, and there's no bias. Let's set the weight to be 1:

Fig 4.13: Single neuron with single input and ReLU activation function, weight set to 1.0.

Now it looks just like the basic rectified linear function, no surprises yet! Now let's set the bias to 0.50:

Fig 4.14: Single neuron with single input and ReLU activation function, bias applied.

We can see that, in this case, with a single neuron, the bias offsets the overall function's activation point *horizontally*. By increasing bias, we're making this neuron activate earlier. What happens when we negate the weight to -1.0?

Fig 4.15: Single neuron with single input and ReLU activation function, negative weight.

With a negative weight and this single neuron, the function has become a question of when this neuron *deactivates*. Up to this point, you've seen how we can use the bias to offset the function horizontally, and the weight to influence the slope of the activation. Moreover, we're also able to control whether the function is one for determining where the neuron activates or deactivates.

What happens when we have, rather than just the one neuron, a pair of neurons? For example, let's pretend that we have 2 hidden layers of 1 neuron each. Thinking back to the $y=x$ activation function, we unsurprisingly discovered that a linear activation function produced linear results no matter what chain of neurons we made. Let's see what happens with the rectified linear function for the activation. We'll begin with the last values for the 1st neuron and a weight of 1, with a bias of 0, for the 2nd neuron:

Fig 4.16: Pair of neurons with single inputs and ReLU activation functions.

As we can see so far, there's no change. This is because the 2nd neuron's bias is doing no offsetting, and the 2nd neuron's weight is just multiplying output by 1, so there's no change. Let's try to adjust the 2nd neuron's bias now:

Fig 4.17: Pair of neurons with single inputs and ReLU activation functions, other bias applied.

Now we see some fairly interesting behavior. The bias of the second neuron indeed shifted the overall function, but, rather than shifting it *horizontally*, it shifted the function *vertically*. What then might happen if we make that 2nd neuron's weight -2 rather than 1?

Fig 4.18: Pair of neurons with single inputs and ReLU activation functions, other negative weight.

Something exciting has occurred! What we have here is a neuron that has both an activation and a deactivation point. When *both* neurons are activated, when their "area of effect" comes into play, they produce values in the range of the granular, variable, and output. If any neuron in the pair is inactive, the pair will produce non-variable output:

Fig 4.19: Pair of neurons with single inputs and ReLU activation functions, area of effect.

ReLU Activation in the Hidden Layers

Let's now take this concept and use it to fit to the sine wave function using 2 hidden layers of 8 neurons each, and we can hand-tune the values to fit the curve. We'll do this by working with 1 pair of neurons at a time, which means 1 neuron from each layer individually. For simplicity, we are also going to assume that the layers are not densely connected, and each neuron from the first hidden layer connects to only one neuron from the second hidden layer. That's usually not the case with the real models, but we want this simplification for the purpose of this demo. Additionally, this example model takes a single value as an input, the input to the sine function, and outputs a single value like the sine function. The output layer uses the Linear activation function, and the hidden layers will use the rectified linear activation function.

To start, we'll set all weights to 0 and work with the first pair of neurons:

Fig 4.20: Hand-tuning a neural network starting with the first pair of neurons.

Next, we can set the weight for the hidden layer neurons and the output neuron to 1, and we can see how this impacts the output:

Fig 4.21: Adjusting weights for the first/top pair of neurons all to 1.

In this case, we can see that the slope of the overall function is impacted. We can further increase this slope by adjusting the weight for the first neuron of the first layer to 6.0:

Fig 4.22: Setting weight for first hidden neuron to 6.

We can now see, for example, that the initial slope of this function is what we'd like, but we have a problem. Currently, this function never ends because this neuron pair never *deactivates*. We can visually see where we'd like the deactivation to occur. It's where the red fitment line (our current neural network's output) diverges initially from the green sine wave. So now, while we have the correct slope, we need to set this spot as our deactivation point. To do that, we start by increasing

the bias for the 2nd neuron of the hidden layer pair to 0.70. Recall that this offsets the overall function *vertically*:

Fig 4.23: Using the bias for the 2nd hidden neuron in the top pair to offset function vertically.

Now we can set the weight for the 2nd neuron to -1, causing a deactivation point to occur, at least horizontally, where we want it:

Fig 4.24: Setting the weight for the 2nd neuron in the top pair to -1.

Now we'd like to flip this slope back. How might we flip the output of these two neurons? It seems like we can take the weight of the connection to the output neuron, which is currently a 1.0, and just flip it to a -1, and that flips the function:

Fig 4.25: Setting the weight to the output neuron to -1.

We're certainly getting closer to making this first section fit how we want. Now, all we need to do is offset this up a bit. For this hand-optimized example, we're going to use the first 7 pairs of neurons in the hidden layers to create the sine wave's shape, then the bottom pair to offset everything vertically. If we set the bias of the 2nd neuron in the bottom pair to 1.0 and the weight to the output neuron as 0.7, we can vertically shift the line like so:

Fig 4.26: Using the bottom pair of neurons to offset the entire neural network function.

At this point, we have completed the first section with an “area of effect” being the first upward section of the sine wave. We can start on the next section that we wish to do. We can start by setting all weights for this 2nd pair of neurons to 1, including the output neuron:

Fig 4.27: Starting to adjust the 2nd pair of neurons (from the top) for the next segment of the overall function.

At this point, this 2nd pair of neurons’ activation is beginning too soon, which is impacting the “area of effect” of the top pair that we already aligned. To fix this, we want this second pair to start influencing the output where the first pair deactivates, so we want to adjust the function horizontally. As you can recall from earlier, we adjust the first neuron’s bias in this neuron pair to achieve this. Also, to modify the slope, we’ll set the weight coming into that first neuron for the 2nd pair, setting it to 3.5. This is the same method we used to set the slope for the first section, which is controlled by the top pair of neurons in the hidden layer. After these adjustments:

Fig 4.28: Adjusting the weight and bias into the first neuron of the 2nd pair.

We will now use the same methodology as we did with the first pair to set the deactivation point. We set the weight for the 2nd neuron in the hidden layer pair to -1 and the bias to 0.27.

Fig 4.29: Adjusting the bias of the 2nd neuron in the 2nd pair.

Then we can flip this section's function, again the same way we did with the first one, by setting the weight to the output neuron from 1.0 to -1.0:

Fig 4.30: Flipping the 2nd pair's function segment, flipping the weight to the output neuron.

And again, just like the first pair, we will use the bottom pair to fix the vertical offset:

Fig 4.31: Using the bottom pair of neurons to adjust the network's overall function.

We then just continue with this methodology. We'll leave it flat for the top section, which means we will only begin the activation for the 3rd pair of hidden layer neurons when we wish for the slope to start going down:

Fig 4.32: Adjusting the 3rd pair of neurons for the next segment.

This process is simply repeated for each section, giving us a final result:

Fig 4.33: The completed process (see anim for all values).

We can then begin to pass data through to see how these neuron's areas of effect come into play — only when both neurons are activated based on input:

Fig 4.34: Example of data passing through this hand-crafted model.

In this case, given an input of 0.08, we can see the only pairs activated are the top ones, as this is their area of effect. Continuing with another example:

Fig 4.35: Example of data passing through this hand-crafted model.

In this case, only the fourth pair of neurons is activated. As you can see, even without any of the other weights, we've used some crude properties of a pair of neurons with rectified linear activation functions to fit this sine wave pretty well. If we enable all of the weights now and allow a mathematical optimizer to train, we can see even better fitment:

Fig 4.36: Example of fitment after fully-connecting the neurons and using an optimizer.

Animation for the entirety of the concept of ReLU fitment:

Anim 4.12-4.36: https://nnfs.io/mvp

It should begin to make more sense to you now how more neurons can enable more unique areas of effect, why we need two or more hidden layers, and why we need nonlinear activation functions to map nonlinear problems. For further example, we can take the above example with 2 hidden layers of 8 neurons each, and instead use 64 neurons per hidden layer, seeing the even further continued improvement:

Fig 4.37: Fitment with 2 hidden layers of 64 neurons each, fully connected, with optimizer.

Anim 4.37: https://nnfs.io/moo

ReLU Activation Function Code

Despite the fancy sounding name, the rectified linear activation function is straightforward to code. Most closely to its definition:

```
ÕĮĴŁĻĽ ł äfld ğd hĝd ġgġd hğgìd ĝgĝd ğgğd hĝflflā

IJŁĻĴŁĻ ł äā
ÌIJĹ Ĩ ĨĮ ĨĮĴŁĻĽf
    ĨÌ Ĩ ń flf
        IJŁĻĴŁĻgĞĴĴĦĮĤâĨã
    ĦĬĽĦf
        IJŁĻĴŁĻgĞĴĴĦĮĤâflã

ĴĹĨĮĻâIJŁĻĴŁĻã


ńńń
äfld ğd fld ġgġd fld ĝgĝd ğgğd flā
```

We made up a list of values to start. The ReLU in this code is a loop where we're checking if the current value is greater than 0. If it is, we're appending it to the output list, and if it's not, we're appending 0. This can be written more simply, as we just need to take the largest of two values: 0 or neuron value. For example:

```
ÕĮĴŁĻĽ ł äfld ğd hĝd ġgġd hğgìd ĝgĝd ğgğd hĝflflā

IJŁĻĴŁĻ ł äā
ÌIJĹ Ĩ ĨĮ ĨĮĴŁĻĽf
    IJŁĻĴŁĻgĞĴĴĦĮĤâİĞŇâfld Ĩãã

ĴĹĨĮĻâIJŁĻĴŁĻã


ńńń
äfld ğd fld ġgġd fld ĝgĝd ğgğd flā
```

NumPy contains an equivalent — `ĮĴgİĞŃĨİŁİâã`:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ĨĮĴŁĻĽ ł äfld ğd hĝd ġgġd hğgìd ĝgĝd ğgğd hĝflflā
IJŁĻĴŁĻ ł ĮĴgİĞŃĨİŁİâfld ĨĮĴŁĻĽã
ĴĹĨĮĻâIJŁĻĴŁĻã


ńńń
äflg  ğg  flg  ġgġ flg  ĝgĝ ğgğ flg ā
```

This method compares each element of the input list (or an array) and returns an object of the same shape filled with new values. We will use it in our new rectified linear activation class:

```
ſi RHLU ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçRHLUf

    ſi FIJĹŃĞĹĤ ĴĞĽĽ
    ĤHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽãf
        ſi CĞĬĢŁĬĞĻH IJŁĻĴŁĻ ĿĞĬŁHĽ ÌĹIJİ ĨĮĴŁĻ
        ĽHĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĨİŁİâfld ĨĮĴŁĻĽã
```

Let's apply this activation function to the dense layer's outputs in our code:

```
ſi CĹHĞĻH ĤĞĻĞĽHĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬHĽłĝflfld ĢĬĞĽĽHĽłġã

ſi CĹHĞĻH DHĮĽH ĬĞÑHĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌHĞĻŁĹHĽ ĞĮĤ ġ IJŁĻĴŁĻ ĿĞĬŁHĽ
ĤHĮĽHĝ ł LĞÑHĹçDHĮĽHâğd ġã

ſi CĹHĞĻH RHLU ĞĢĻĨĿĞĻĨIJĮ âĻIJ ĠH ŁĽHĤ ŃĨĻÎ DHĮĽH ĬĞÑHĹãf
ĞĢĻĨĿĞĻĨIJĮĝ ł AĢĻĨĿĞĻĨIJĮçRHLUâã

ſi MĞĪH Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮĬ ĤĞĻĞ ĻÎĹIJŁĬÎ ĻÎĨĽ ĬĞÑHĹ
ĤHĮĽHĝgÌIJĹŃĞĹĤâXã

ſi FIJĹŃĞĹĤ ĴĞĽĽ ĻÎĹIJŁĬÎ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢg
ſi TĞĪHĽ ĨĮ IJŁĻĴŁĻ ÌĹIJİ ĴĹHĿĨIJŁĽ ĬĞÑHĹ
ĞĢĻĨĿĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤHĮĽHĝgIJŁĻĴŁĻã
```

```
ﬁ LĦĻďĽ ĽĦĦ IJŁĻĴŁĻ IJÌ ĻÎĦ ÌĨĹĽĻ ÌĦŃ ĽĞİĴĬĦĽf
ĴĹĨĮĻâĞĢĻĨĿĞĻĨIJIĢgIJŁĻĴŁĻäfĥāã


ńńń
ääflg          flg          flg         ā
 äflg          flgflflflĝĝġîĥ flg        ā
 äflg          flgflflflġĝìğî flg        ā
 äflg          flgflflflĥğħħħ flg        ā
 äflg          flgflflflìĝġflĝ flg       āā
```

As you can see, negative values have been **clipped** (modified to be zero). That's all there is to the rectified linear activation function used in the hidden layer. Let's talk about the activation function that we are going to use on the output of the last layer.

The Softmax Activation Function

In our case, we're looking to get this model to be a classifier, so we want an activation function meant for classification. One of these is the Softmax activation function. First, why are we bothering with another activation function? It just depends on what our overall goals are. In this case, the rectified linear unit is unbounded, not normalized with other units, and exclusive. "Not normalized" implies the values can be anything, an output of *[12, 99, 318]* is without context, and "exclusive" means each output is independent of the others. To address this lack of context, the softmax activation on the output data can take in non-normalized, or uncalibrated, inputs and produce a normalized distribution of probabilities for our classes. In the case of classification, what we want to see is a prediction of which class the network "thinks" the input represents. This distribution returned by the softmax activation function represents **confidence scores** for each class and will add up to 1. The predicted class is associated with the output neuron that returned the largest confidence score. Still, we can also note the other confidence scores in our overarching algorithm/program that uses this network. For example, if our network has a confidence distribution for two classes: *[0.45, 0.55]*, the prediction is the 2nd class, but the confidence in this prediction isn't very high. Maybe our program would not act in this case since it's not very confident.

Here's the function for the **Softmax**:

$$S_{i,j} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}$$

That might look daunting, but we can break it down into simple pieces and express it in Python code, which you may find is more approachable than the formula above. To start, here are example outputs from a neural network layer:

```
ĬĞÑŦĹçIJŁĻĴŁĻĽ ł äǵǵíd ĝǵǧĝd ǧǵíĥā
```

The first step for us is to “exponentiate” the outputs. We do this with Euler’s number, *e,* which is roughly *2.71828182846* and referred to as the “exponential growth” number. Exponentiating is taking this constant to the power of the given parameter:

$$y = e^x$$

Both the numerator and the denominator of the Softmax function contain *e* raised to the power of *z*, where *z*, given indices, means a singular output value — the index *i* means the current sample and the index *j* means the current output in this sample. The numerator exponentiates the current output value and the denominator takes a sum of all of the exponentiated outputs for a given sample. We need then to calculate these exponentiates to continue:

```
fi VĞĬŁĦĽ ÌĹIJİ ĻÎĦ ĴĹĦŁĬIJŁĽ IJŁĻĴŁĻ ŃÎĦĮ ŃĦ ÂĦĽĢĹĬĠĦÂ
fi ŃÎĞĻ Ğ ĮĦŁĹĞĬ ĮĦĻŃIJĹĪ ĬĽ
ĬĞÑĦĹçIJŁĻĴŁĻĽ ł äǵǵíd ǵǵǵd ǵǵǵíĥā

fi Ħ h ĬĞĻÎĦĬĞĻĬĢĞĬ ĢIJĮĹĻĞĮĻd ŃĦ ŁĽĦ E ÎĦĹĦ ĻIJ ĬĞĻĢÎ Ğ ĢIJĬĬIJĮ ĢIJÂĬĮÍ
fi ĽĻÑĬĦ ŃÎĦĹĦ ĢIJĮĹĻĞĮĻĽ ĞĹĦ ŁĴĴĦĹĢĞĽĦÂ
E ł ǵǵìǵíǵíǵíǵíǵh  fi ÑIJŁ ĢĞĮ ĞĬĽIJ ŁĽĦ ĬĞĻÎgĦ

fi FIJĹ ĦĞĢÎ ĻĞĬŁĦ ĬĮ Ğ ĻĦĢĻIJĹd ĢĞĬĢŁĬĞĻĦ ĻÎĦ ĦŃĴIJĮĦĮĻĬĞĬ ĻĞĬŁĦ
ĦŃĴçĻĞĬŁĦĽ ł äā
ÌIJĹ IJŁĻĴŁĻ ĬĮ ĬĞÑĦĹçIJŁĻĴŁĻĽf
    ĦŃĴçĻĞĬŁĦĽgĞĴĴĦĮÂâE ǻǻ IJŁĻĴŁĻã  fi ǻǻ h ĴIJŃĦĹ IJĴĦĹĞĻIJĹ ĬĮ PÑĻÎIJĮ
ĴĹĬĮĻâďĦŃĴIJĮĦĮĻĬĞĻĦÂ ĻĞĬŁĦĽfďã
ĴĹĬĮĻâĦŃĴçĻĞĬŁĦĽã


ńńń
ĦŃĴIJĮĦĮĻĬĞĻĦÂ ĻĞĬŁĦĽf
äǵǵǵĥǵflǵǵìĥǵíǵíĥíd ǵǵǵĥǵǵíǵĥĥǵĥĥflǵǵíìd ǵflǵíĥíflĥǵĥĥǵíǵíĥǵā
```

Exponentiation serves multiple purposes. To calculate the probabilities, we need non-negative values. Imagine the output as äǵǵíd ǵǵǵd hǵǵíĥā — even after normalization, the last value will still be negative since we’ll just divide all of them by their sum. A negative probability (or confidence) does not make much sense. An exponential value of any number is always non-negative — it returns 0 for negative infinity, 1 for the input of 0, and increases for positive values:

Fig 4.38: Graph of an exponential function.

The exponential function is a monotonic function. This means that, with higher input values, outputs are also higher, so we won't change the predicted class after applying it while making sure that we get non-negative values. It also adds stability to the result as the normalized exponentiation is more about the difference between numbers than their magnitudes. Once we've exponentiated, we want to convert these numbers to a probability distribution (converting the values into the vector of confidences, one for each class, which add up to 1 for everything in the vector). What that means is that we're about to perform a normalization where we take a given value and divide it by the sum of all of the values. For our outputs, exponentiated at this stage, that's what the equation of the Softmax function describes next — to take a given exponentiated value and divide it by the sum of all of the exponentiated values. Since each output value normalizes to a fraction of the sum, all of the values are now in the range of 0 to 1 and add up to 1 — they share the probability of 1 between themselves. Let's add the sum and normalization to the code:

```
ń NIJŃ ĮIJĹİĞĬĨŅĦ ĽĞĬłĦĽ
ĮIJĹİçĠĞĽĦ ł ĽłİâĦŇĴçĽĞĬłĦĽã  ń WĦ Ľłİ ĞĬĬ ĽĞĬłĦĽ
ĮIJĹİçĽĞĬłĦĽ ł äā
ÌIJĹ ĽĞĬłĦ ĨĮ ĦŇĴçĽĞĬłĦĽf
    ĮIJĹİçĽĞĬłĦĽgĞĴĴĦĮĤâĽĞĬłĦ w ĮIJĹİçĠĞĽĦã
ĴĹĨĮĻâďNIJĹİĞĬĨŅĦĤ ĦŇĴIJĮĦĮĻĨĞĻĦĤ ĽĞĬłĦĽfďã
ĴĹĨĮĻâĮIJĹİçĽĞĬłĦĽã

ĴĹĨĮĻâďSłİ IJÌ ĮIJĹİĞĬĨŅĦĤ ĽĞĬłĦĽfďd ĽłİâĮIJĹİçĽĞĬłĦĽãã
```

```
ńńń
NIJĹİĞĬĨŅĦĤ ĦŇĴIJĮĦIĮĨĞĻĦĤ ĽĞĬŁĦĽf
äflgíîĥğíğħħġîĥìġĥflħd flgflğġìflíġflħìíğflìflħħíd flgflíflflflîflğîğħflĥìííìħā
SŁİ IJÌ ĮIJĹİĞĬĨŅĦĤ ĽĞĬŁĦĽf ĝgfl
```

We can perform the same set of operations with the use of NumPy in the following way:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ń VĞĬŁĦĽ ÌĹIJİ ĻÎĦ ĦĞĹĬĨĦĹ ĴĹĦĽĨIJŁĽ ŃÎĦĮ ŃĦ ĤĦĽĢĹĨĠĦĤ
ń ŃÎĞĻ Ğ ĮĦŁĹĞĬ ĮĦĻŃIJĹĪ ĨĽ

ĬĞÑĦĹçIJŁĻĴŁĻĽ ł äġgíd ĝgğĝd ğgġíĥā

ń FIJĹ ĦĞĢÎ ĽĞĬŁĦ ĨĮ Ğ ĽĦĢĻIJĹd ĢĞĬĢŁĬĞĻĦ ĻÎĦ ĦŇĴIJĮĦIĮĨĞĬ ĽĞĬŁĦ
ĦŇĴçĽĞĬŁĦĽ ł ĮĴgĦŇĴâĬĞÑĦĹçIJŁĻĴŁĻĽã
ĴĹĨĮĻâďĦŇĴIJĮĦIĮĨĞĻĦĤ ĽĞĬŁĦĽfďã
ĴĹĨĮĻâĦŇĴçĽĞĬŁĦĽã

ń NIJŃ ĮIJĹİĞĬĨŅĦ ĽĞĬŁĦĽ
ĮIJĹİçĽĞĬŁĦĽ ł ĦŇĴçĽĞĬŁĦĽ w ĮĴgĽŁİâĦŇĴçĽĞĬŁĦĽã
ĴĹĨĮĻâďĮIJĹİĞĬĨŅĦĤ ĦŇĴIJĮĦIĮĨĞĻĦĤ ĽĞĬŁĦĽfďã
ĴĹĨĮĻâĮIJĹİçĽĞĬŁĦĽã
ĴĹĨĮĻâďĽŁİ IJÌ ĮIJĹİĞĬĨŅĦĤ ĽĞĬŁĦĽfďd ĮĴgĽŁİâĮIJĹİçĽĞĬŁĦĽãã


ńńń
ĦŇĴIJĮĦIĮĨĞĻĦĤ ĽĞĬŁĦĽf
äĝğĝgĥĝflġĝìĥğ   ġgġĥġġíġħĥ  ĝflgíĥîflħğħħā
ĮIJĹİĞĬĨŅĦĤ ĦŇĴIJĮĦIĮĨĞĻĦĤ ĽĞĬŁĦĽf
äflgíîĥğíğħħ flgflğġìflíġĝ flgflíflflflîflġā
ĽŁİ IJÌ ĮIJĹİĞĬĨŅĦĤ ĽĞĬŁĦĽf flgîîîîîîîîîîîîîîîî
```

Notice the results are similar, but faster to calculate and the code is easier to read with NumPy. We can exponentiate all of the values with a single call of the `ĮĴgĦŇĴâã`, then immediately normalize them with the sum. To train in batches, we need to convert this functionality to accept layer outputs in batches. Doing this is as easy as:

```
ń GĦĻ ŁĮĮIJĹİĞĬĨŅĦĤ ĴĹIJĠĞĠĨĬĬĮĨĦĽ
ĦŇĴçĽĞĬŁĦĽ ł ĮĴgĦŇĴâĨĮĴŁĻĽã

ń NIJĹİĞĬĨŅĦ ĻÎĦİ ÌIJĹ ĦĞĢÎ ĽĞİĴĬĦ
ĴĹIJĠĞĠĨĬĬĮĨĦĽ ł ĦŇĴçĽĞĬŁĦĽ w ĮĴgĽŁİâĦŇĴçĽĞĬŁĦĽd ĞŇĨĽłĝd ĪĦĦĴĤĨİĽłTĹŁĦã
```

We have some new functions. Specifically, ĮĴgĦŇĴââ does the Eááıjłļĵłļ part. We should also address what *axis* and *keepdims* mean in the above. Let's first discuss the *axis*. Axis is easier to show than tell, but, in a 2D array/matrix, axis 0 refers to the rows, and axis 1 refers to the columns. Let's see some examples of how *axis* affects the sum using NumPy. First, we will just show the default, which is *None*

```
ĨİĴıjĹĻ ĮłİĴÑ ĞĽ ĮĴ


ĬĞÑĦĹçıjłĻĴłĻĽ ł ĮĴgĞĹĹĞÑâäägíd ĝggĝd ggíĥād
                          äígîd hĝgíĝd flgğād
                          äĝgĝĝd ĝgflĥĝd flgflğħāāā

ĴĹĨĮĻâďSłİ ŃĨĻÎıjłĻ ĞŇĨĽďã
ĴĹĨĮĻâĮĴgĽłİâĬĞÑĦĹçıjłĻĴłĻĽãã

ĴĹĨĮĻâďTÎĨĽ ŃĨĬĬ ĠĦ ĨÎĦĮĻĨĢĞĬ ĻıJ ĻÎĦ ĞĠıjLĦ ĽĨĮĢĦ ÎĦÌĞłĬĻ ĨĽ NıJĮĦfďã
ĴĹĨĮĻâĮĴgĽłİâĬĞÑĦĹçıjłĻĴłĻĽd ĞŇĨĽłNıJĮĦãã


ńńń
Słİ ŃĨĻÎıjłĻ ĞŇĨĽ
ĝígĝìğ
TÎĨĽ ŃĨĬĬ ĠĦ ĨÎĦĮĻĨĢĞĬ ĻıJ ĻÎĦ ĞĠıjLĦ ĽĨĮĢĦ ÎĦÌĞłĬĻ ĨĽ NıJĮĦf
ĝígĝìğ
```

With no axis specified, we are just summing all of the values, even if they're in varying dimensions. Next, ĞŇĨĽłfl. This means to sum row-wise, along axis 0. In other words, the output has the same size as this axis, as at each of the positions of this output, the values from all the other dimensions at this position are summed to form it. In the case of our 2D array, where we have only a single other dimension, the columns, the output vector will sum these columns. This means we'll perform *4.8+8.9+1.41* and so on.

```
ĴĹĨĮĻâďAĮıjĻÎĦĹ ŃĞÑ ĻıJ ĻÎĨĮĪ ıjÌ ĨĻ Ńw Ğ İĞĻĹĨŇ łł ĞŇĨĽ flf ĢıjĬłİĮĽfďã
ĴĹĨĮĻâĮĴgĽłİâĬĞÑĦĹçıjłĻĴłĻĽd ĞŇĨĽłflãã


ńńń
AĮıjĻÎĦĹ ŃĞÑ ĻıJ ĻÎĨĮĪ ıjÌ ĨĻ Ńw Ğ İĞĻĹĨŇ łł ĞŇĨĽ flf ĢıjĬłİĮĽf
äĝĥggĝ   flgĝĥĝ  ğgħĝĝā
```

This isn't what we want, though. We want sums of the rows. You can probably guess how to do this with NumPy, but we'll still show the "from scratch" version:

```
ĴĹĨĮĻâďBŁĻ ŃĦ ŃĞĮĻ ĻIJ ĽŁİ ĻĨĦ ĹIJŃĽ ĨĮĽĻĦĞĤd ĬĨĪĦ ĻĨĨĽ Ńw ĹĞŃ ĴÑfďã

ÌIJĹ Ĩ ĨĮ ĬĞÑĦĹçIJŁĻĴŁĻĽf
    ĴĹĨĮĻâĽŁİâĨãã


ńńń
BŁĻ ŃĦ ŃĞĮĻ ĻIJ ĽŁİ ĻĨĦ ĹIJŃĽ ĨĮĽĻĦĞĤd ĬĨĪĦ ĻĨĨĽ Ńw ĹĞŃ ĴÑf
íġġîĥ
ìgğî
ğgǵíħîîîîîîîîîîîî
```

With the above, we could append these to some list in any way we want. That said, we're going to use NumPy. As you probably guessed, we're going to sum along axis 1:

```
ĴĹĨĮĻâďSIJ ŃĦ ĢĞĮ ĽŁİ ĞŃĨĽ ĝd ĠŁĻ ĮIJĻĦ ĻĨĦ ĢŁĹĹĦĮĻ ĽĨĞĴĦfďã
ĴĹĨĮĻâĮĴgĽŁİâĬĞÑĦĹçIJŁĻĴŁĻĽd ĞŃĨĽłĝãã


ńńń
SIJ ŃĦ ĢĞĮ ĽŁİ ĞŃĨĽ ĝd ĠŁĻ ĮIJĻĦ ĻĨĦ ĢŁĹĹĦĮĻ ĽĨĞĴĦf
äíġġîĥ ìgğî ğgǵíìā
```

As pointed out by "ĮIJĻĦ ĻĨĦ ĢŁĹĹĦĮĻ ĽĨĞĴĦ," we did get the sums that we expected, but actually, we want to simplify the outputs to a single value per sample. We're trying to sum all the outputs from a layer for each sample in a batch; converting the layer's output array with row length equal to the number of neurons in the layer, to just one value. We need a column vector with these values since it will let us normalize the whole batch of samples, sample-wise, with a single calculation.

```
ĴĹĨĮĻâďSŁİ ĞŃĨĽ ĝd ĠŁĻ ĪĦĦĴ ĻĨĦ ĽĞİĦ ĤĨİĦĮĽĨIJĮĽ ĞĽ ĨĮĴŁĻfďã
ĴĹĨĮĻâĮĴgĽŁİâĬĞÑĦĹçIJŁĻĴŁĻĽd ĞŃĨĽłĝd ĪĦĦĴĤĨİĽłTĹŁĦãã


ńńń
SŁİ ĞŃĨĽ ĝd ĠŁĻ ĪĦĦĴ ĻĨĦ ĽĞİĦ ĤĨİĦĮĽĨIJĮĽ ĞĽ ĨĮĴŁĻf
ääíġġîĥā
 äìgğî ā
 äğgǵíìāā
```

With this, we keep the same dimensions as the input. Now, if we divide the array containing a batch of the outputs with this array, NumPy will perform this sample-wise. That means that it'll divide all of the values from each output row by the corresponding row from the sum array. Since this sum in each row is a single value, it'll be used for the division with every value from the corresponding output row). We can combine all of this into a softmax class, like:

```
class SoftmaxActivation:
    def forward(self, inputs):
        # Get unnormalized probabilities
        exp_values = np.exp(inputs - np.max(inputs, axis=1,
                                            keepdims=True))
        # Normalize them for each sample
        probabilities = exp_values / np.sum(exp_values, axis=1,
                                            keepdims=True)

        self.output = probabilities
```

Finally, we also included a subtraction of the largest of the inputs before we did the exponentiation. There are two main pervasive challenges with neural networks: "dead neurons" and very large numbers (referred to as "exploding" values). "Dead" neurons and enormous numbers can wreak havoc down the line and render a network useless over time. The exponential function used in softmax activation is one of the sources of exploding values. Let's see some examples of how and why this can easily happen:

```
import numpy as np

inputs = np.array([[1, 2, 3],
                   [-4, -5, -6],
                   [0.1, 0.2, 0.3]])

exp_values = np.exp(inputs)
print(exp_values)
print(np.sum(exp_values, axis=1, keepdims=True))

norm_values = exp_values / np.sum(exp_values, axis=1,
                                  keepdims=True)
print(norm_values)
print(np.sum(norm_values, axis=1, keepdims=True))
```

```
ĴĹĨĮĻâĮĴgĦŇĴâĝflflãã


ńńń
ğgħííĝĝìĝǵĝíĝħĝġĥħĦĸǵġ


ĴĹĨĮĻâĮĴgĦŇĴâĝflflflãã


ńńń
ççİĞĨĮççfĝf RŁĮĻĨİĦWĞĹĮĨĮĺf IJŁĦĹÌĬIJŃ ĦĮĢIJŁĮĻĦĹĦĤ ĨĮ ĦŇĴ
ĨĮÌ
```

It doesn't take a very large number, in this case, a mere *1,000*, to cause an overflow error. We know the exponential function tends toward 0 as its input value approaches negative infinity, and the output is 1 when the input is 0 (as shown in the chart earlier):

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ĴĹĨĮĻâĮĴgĦŇĴâhĮĴgĨĮÌãd ĮĴgĦŇĴâflãã


ńńń
flgfl ĝgfl
```

We can use this property to prevent the exponential function from overflowing. Suppose we subtract the maximum value from a list of input values. We would then change the output values to always be in a range from some negative value up to 0, as the largest number subtracted by itself returns 0, and any smaller number subtracted by it will result in a negative number — exactly the range discussed above. With Softmax, thanks to the normalization, we can subtract any value from all of the inputs, and it will not change the output:

```
ĹIJÌĻİĞŇ Ł AĢĻĨĽĞĻĨIJĮçSIJÌĻİĞŇâã

ĹIJÌĻİĞŇgÌIJĹŃĞĹĤâäägd ğd ġāãã
ĴĹĨĮĻâĹIJÌĻİĞŇgIJŁĻĴŁĻã


ńńń
ääflgflîflflġflĥì flgğǵǵìğíǵì flgħĥĥĝǵflîħãã
```

```
ĽIJÌĻİĞŇgÌIJĹŃĞĹÎâäähğd hĝd flāāã  fi ĽŁĠĻĹĞĢĻĦĤ ġ h İĞŇ ÌĹIJİ ĻÎĦ ĬĨĽĻ
ĴĹĨĮĻâĽIJÌĻİĞŇgIJŁĻĴŁĻã


ńńń
äāflgflîflflġflĥì flgğǵgìğíǵgì flgħĥĥğǵflîħāā
```

This is another useful property of the exponentiated and normalized function. There's one more thing to mention in addition to these calculations. What happens if we divide the layer's output data, äĝd ğd ġā, for example, by 2?

```
ĽIJÌĻİĞŇgÌIJĹŃĞĹÎâäāflgĥd ĝd ĝgĥāāã
ĴĹĨĮĻâĽIJÌĻİĞŇgIJŁĻĴŁĻã


ńńń
āāflgĝíħġğǵìğ flgġflìĝîĥíî flgĥflħǵíflġîāā
```

The output confidences have changed due to the nonlinearity nature of the exponentiation. This is one example of why we need to scale all of the input data to a neural network in the same way, which we'll explain in further detail in chapter 22.

Now, we can add another dense layer as the output layer, setting it to contain as many inputs as the previous layer has outputs and as many outputs as our data includes classes. Then we can apply the softmax activation to the output of this new layer:

```
fi CĹĦĞĻĦ ÎĞĻĞĽĦĻ
Xd Ñ ł ĽĴĬĹĞĬçÎĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

fi CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮÎ ġ IJŁĻĴŁĻ L·ĞĬŁĦĽ
ÎĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ġã

fi CĹĦĞĻĦ RĦLU ĞĢĻĨL·ĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦÎ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨL·ĞĻĨIJĮĝ ł AĢĻĨL·ĞĻĨIJĮçRĦLUâã

fi CĹĦĞĻĦ ĽĦĢIJĮÎ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ġ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĬĦ IJŁĻĴŁĻ
fi IJÌ ĴĹĦL·ĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮÎ ġ IJŁĻĴŁĻ L·ĞĬŁĦĽ
ÎĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâġd ġã

fi CĹĦĞĻĦ SIJÌĻİĞŇ ĞĢĻĨL·ĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦÎ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨL·ĞĻĨIJĮğ ł AĢĻĨL·ĞĻĨIJĮçSIJÌĻİĞŇâã

fi MĞĬĦ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ÎĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑĦĹ
ÎĦĮĽĦĝgÌIJĹŃĞĹÎâXã
```

```
ƒ MĞĪĦ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
ƒ ĨĻ ĻĞĪĦĹ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ÂĦĮĹĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨĿĞĻĨIJĮĝgÌIJĹŃĞĹÂâÂĦĮĹĦĝgIJŁĻĴŁĻã

ƒ MĞĪĦ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĹĦĢIJĮÂ DĦĮĹĦ ĬĞÑĦĹ
ƒ ĨĻ ĻĞĪĦĹ IJŁĻĴŁĻĹ IJÌ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĹ ĨĮĴŁĻĹ
ÂĦĮĹĦğgÌIJĹŃĞĹÂâĞĢĻĨĿĞĻĨIJĮĝgIJŁĻĴŁĻã

ƒ MĞĪĦ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
ƒ ĨĻ ĻĞĪĦĹ ĻÎĦ IJŁĻĴŁĻ IJÌ ĹĦĢIJĮÂ ÂĦĮĹĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨĿĞĻĨIJĮğgÌIJĹŃĞĹÂâÂĦĮĹĦğgIJŁĻĴŁĻã

ƒ LĦĻďĹ ĹĦĦ IJŁĻĴŁĻ IJÌ ĻÎĦ ÌĨĹĽĻ ÌĦŃ ĹĞÌĴĬĦĹf
ĴĹĨĮĻâĞĢĻĨĿĞĻĨIJĮğgIJŁĻĴŁĻäfĥãã


ńńń
ääflgġġġġġġġ flgġġġġġġġ flgġġġġġġġġã
 äflgġġġġġġĝħ flgġġġġġġğ  flgġġġġġġħġã
 äflgġġġġġğíì flgġġġġġğî  flgġġġġġġĝíã
 äflgġġġġġğħ  flgġġġġġğħġ flgġġġġġġìã
 äflgġġġġġğġġ flgġġġġġğġ  flgġġġġġĥğíãã
```

As you can see, the distribution of predictions is almost equal, as each of the samples has ~33% (0.33) predictions for each class. This results from the random initialization of weights (a draw from the normal distribution, as not every random initialization will result in this) and zeroed biases. These outputs are also our "confidence scores." To determine which classification the model has chosen to be the prediction, we perform an *argmax* on these outputs, which checks which of the classes in the output distribution has the highest confidence and returns its index - the predicted class index. That said, the confidence score can be as important as the class prediction itself. For example, the argmax of `äflgğğd flgħd flgĝíã` is the same as the argmax for `äflgġğd flgġħd flgġğã`. In both of these, the argmax function would return an index value of 1 (the 2nd element in Python's zero-indexed paradigm), but obviously, a 60% confidence is much better than a 36% confidence.

Full code up to this point:

```
ĨİĴIJĹĻ ĮłİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĨİĴIJĹĻ ĽĴĨĹĞĬçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


ſi DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ſi LĞÑĦĹ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽãf
        ſi IĮĨĻĨĞĬĨŅĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍÎĻĽ ł flgflĝ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã

    ſi FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ſi CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ŁĞĬŁĦĽ ÌĹIJİ ĨĮĴŁĻĽd ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĽĦĬÌgŃĦĨÍÎĻĽã κ ĽĦĬÌgĠĨĞĽĦĽ


ſi RĦLU ĞĢĻĨŁĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨŁĞĻĨIJĮçRĦLUf

    ſi FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ſi CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ŁĞĬŁĦĽ ÌĹIJİ ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĨİŁİâfld ĨĮĴŁĻĽã
```

```
# SIJÌĻİĞŇ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇf

    # FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ĨĮĴŁĻĽãf

        # GĦĻ ŁĮĮIJĹİĞĬĨŅĦÂ ĴĹIJĠĞĠĨĬĨĻĨĦĽ
        ĦŇĴçĿĞĬŁĦĽ ł ĮĴgĦŇĴâĨĮĴŁĻĽ h ĮĴgİĞŇâĨĮĴŁĻĽd ĞŇĨĽłĝd
                                                ĪĦĦĴÂĨİĽłTĹŁĦãã
        # NIJĹİĞĬĨŅĦ ĻÎĦİ ÌIJĹ ĦĞĢÎ ĽĞİĴĬĦ
        ĴĹIJĠĞĠĨĬĨĻĨĦĽ ł ĦŇĴçĿĞĬŁĦĽ w ĮĴgĽŁİâĦŇĴçĿĞĬŁĦĽd ĞŇĨĽłĝd
                                                ĪĦĦĴÂĨİĽłTĹŁĦã

        ĽĦĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĨĬĨĻĨĦĽ


# CĹĦĞĻĦ ÂĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçÂĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

# CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮÂ ġ IJŁĻĴŁĻ ĿĞĬŁĦĽ
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ġã

# CĹĦĞĻĦ RĦLU ĞĢĻĨĿĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦÂ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĿĞĻĨIJĮĝ ł AĢĻĨĿĞĻĨIJĮçRĦLUâã

# CĹĦĞĻĦ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ġ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĬĦ IJŁĻĴŁĻ
# IJÌ ĴĹĦĿĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮÂ ġ IJŁĻĴŁĻ ĿĞĬŁĦĽ âIJŁĻĴŁĻ ĿĞĬŁĦĽã
ÂĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâġd ġã

# CĹĦĞĻĦ SIJÌĻİĞŇ ĞĢĻĨĿĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦÂ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĿĞĻĨIJĮğ ł AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇâã

# MĞĨĦ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ÂĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑĦĹ
ÂĦĮĽĦĝgÌIJĹŃĞĹÂâXã

# MĞĨĦ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
# ĨĻ ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨĿĞĻĨIJĮĝgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJŁĻĴŁĻã

# MĞĨĦ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ
# ĨĻ ĻĞĨĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
ÂĦĮĽĦğgÌIJĹŃĞĹÂâĞĢĻĨĿĞĻĨIJĮĝgIJŁĻĴŁĻã

# MĞĨĦ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
# ĨĻ ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨĿĞĻĨIJĮğgÌIJĹŃĞĹÂâÂĦĮĽĦğgIJŁĻĴŁĻã
```

```
ńńń
ääflgġġġġġġġġ flgġġġġġġġġ flgġġġġġġġġā
 äflgġġġġġġġĝħ flgġġġġġġġ  flgġġġġġġħġā
 äflgġġġġġġġĭìì flgġġġġġġĭî  flgġġġġġġĝíā
 äflgġġġġġġġħ  flgġġġġġġħġ flgġġġġġġìiā
 äflgġġġġġġĭġġ flgġġġġġġĭġ  flgġġġġġĥĭíāā
```

We've completed what we need for forward-passing data through our model. We used the **Rectified Linear (ReLU**) activation function on the hidden layer, which works on a per-neuron basis. We additionally used the **Softmax** activation function for the output layer since it accepts non-normalized values as input and outputs a probability distribution, which we're using as confidence scores for each class. Recall that, although neurons are interconnected, they each have their respective weights and biases and are not "normalized" with each other.

As you can see, our example model is currently random. To remedy this, we need a way to calculate how wrong the neural network is at current predictions and begin adjusting weights and biases to decrease error over time. Thus, our next step is to quantify how wrong the model is through what's defined as a **loss function**.

Supplementary Material: https://nnfs.io/ch4

Chapter code, further resources, and errata for this chapter.

Chapter 5

Calculating Network Error with Loss

With a randomly-initialized model, or even a model initialized with more sophisticated approaches, our goal is to train, or teach, a model over time. To train a model, we tweak the weights and biases to improve the model's accuracy and confidence. To do this, we calculate how much error the model has. The **loss function**, also referred to as the **cost function**, is the algorithm that quantifies how wrong a model is. **Loss** is the measure of this metric. Since loss is the model's error, we ideally want it to be 0.

You may wonder why we do not calculate the error of a model based on the argmax accuracy. Recall our earlier example of confidence: `äflgğğd flgħd flgĝíā` vs `äflgġğd flgġħd flgġğā`. If the correct class were indeed the middle one (index 1), the model accuracy would be identical between the two above. But are these two examples *really* as accurate as each other? They are not, because accuracy is simply applying an argmax to the output to find the index of the biggest value. The output of a neural network is actually confidence, and more confidence in

the correct answer is better. Because of this, we strive to increase correct confidence and decrease misplaced confidence.

Categorical Cross-Entropy Loss

If you're familiar with linear regression, then you already know one of the loss functions used with neural networks that do regression: **squared error** (or **mean squared error** with neural networks).

We're not performing regression in this example; we're classifying, so we need a different loss function. The model has a softmax activation function for the output layer, which means it's outputting a probability distribution. **Categorical cross-entropy** is explicitly used to compare a "ground-truth" probability (*y* or "*targets*") and some predicted distribution (*y-hat* or "*predictions*"), so it makes sense to use cross-entropy here. It is also one of the most commonly used loss functions with a softmax activation on the output layer.

The formula for calculating the categorical cross-entropy of *y* (actual/desired distribution) and *y-hat* (predicted distribution) is:

$$L_i = -\sum_j y_{i,j} log(\hat{y}_{i,j})$$

Where L_i denotes sample loss value, *i* is the i-th sample in the set, *j* is the label/output index, *y* denotes the target values, and *y-hat* denotes the predicted values.

Once we start coding the solution, we'll simplify it further to *-log(correct_class_confidence)*, the formula for which is:

$$L_i = -\log(\hat{y}_{i,k}) \quad where\ \mathbf{k}\ is\ an\ index\ of\ "true"\ probability$$

Where L_i denotes sample loss value, *i* is the i-th sample in a set, *k* is the index of the target label (ground-true label), *y* denotes the target values and *y-hat* denotes the predicted values.

You may ask why we call this cross-entropy and not **log loss**, which is also a type of loss. If you do not know what log loss is, you may wonder why there is such a fancy looking formula for what looks to be a fairly basic description.

In general, the log loss error function is what we apply to the output of a binary logistic regression model (which we'll describe in chapter 16) — there are only two classes in the distribution, each of them applying to a single output (neuron) which is targeted as a 0 or 1. In our case, we have a classification model that returns a probability distribution over all of the outputs. Cross-entropy compares two probability distributions. In our case, we have a softmax output, let's say it's:

```
ĹIJÌĻİĞŇçIJŁĻĴŁĻ ł äflgìd flgĝd flgğā
```

Which probability distribution do we intend to compare this to? We have 3 class confidences in the above output, and let's assume that the desired prediction is the first class (index 0, which is currently 0.7). If that's the intended prediction, then the desired probability distribution is äĝd fld flā. Cross-entropy can also work on probability distributions like äflgğd flgĥd flgġā; they wouldn't have to look like the one above. That said, the desired probabilities will consist of a 1 in the desired class, and a 0 in the remaining undesired classes. Arrays or vectors like this are called **one-hot**, meaning one of the values is "hot" (on), with a value of 1, and the rest are "cold" (off), with values of 0. When comparing the model's results to a one-hot vector using cross-entropy, the other parts of the equation zero out, and the target probability's log loss is multiplied by 1, making the cross-entropy calculation relatively simple. This is also a special case of the cross-entropy calculation, called categorical cross-entropy. To exemplify this — if we take a softmax output of äflgìd flgĝd flgğā and targets of äĝd fld flā, we can apply the calculations as follows:

$$L_i = -\sum_j y_{i,j} log(\hat{y}_{i,j}) = -(1 \cdot log(0.7) + 0 \cdot log(0.1) + 0 \cdot log(0.2)) =$$
$$= -(-0.35667494393873245 + 0 + 0) = 0.35667494393873245$$

Let's see the Python code for this:

```
ĨİĴIJĹĻ İĞĻÎ


ńĩ AĮ ĦŇĞİĴĬĦ IJŁĻĴŁĻ ÌĹIJİ ĻÎĦ IJŁĻĴŁĻ ĬĞÑĦĹ IJÌ ĻÎĦ ĮĦŁĹĞĬ ĮĦĻŃIJĹĪ
ĽIJÌĻİĞŇçIJŁĻĴŁĻ ł äflgìd flgĝd flgğā
ńĩ GĹIJŁĮĤ ĻĹŁĻÎ
ĻĞĹÍĦĻçIJŁĻĴŁĻ ł äĝd fld flā

ĬIJĹĽ ł hâİĞĻÎgĬIJÍâĽIJÌĻİĞŇçIJŁĻĴŁĻäflāāǻĻĞĹÍĦĻçIJŁĻĴŁĻäflā ĸ
         İĞĻÎgĬIJÍâĽIJÌĻİĞŇçIJŁĻĴŁĻäĝāāǻĻĞĹÍĦĻçIJŁĻĴŁĻäĝā ĸ
         İĞĻÎgĬIJÍâĽIJÌĻİĞŇçIJŁĻĴŁĻäğāāǻĻĞĹÍĦĻçIJŁĻĴŁĻäğāā

ĴĹĨĮĻâĬIJĽĹã


ńńń
flgġĥħħìǵîǵġîġíìġğǵĥ
```

That's the full categorical cross-entropy calculation, but we can make a few assumptions given one-hot target vectors. First, what are the values for `ĻĞĹÍĦĻçIJŁĻĴŁĻäĝā` and `ĻĞĹÍĦĻçIJŁĻĴŁĻäğā` in this case? They're both 0, and anything multiplied by 0 is 0. Thus, we don't need to calculate these indices. Next, what's the value for `ĻĞĹÍĦĻçIJŁĻĴŁĻäflā` in this case? It's 1. So this can be omitted as any number multiplied by 1 remains the same. The same output then, in this example, can be calculated with:

```
ĬIJĹĽ ł hİĞĻÎgĬIJÍâĽIJÌĻİĞŇçIJŁĻĴŁĻäflāā
```

Which still gives us:

```
ńńń
flgġĥħħìǵîǵġîġíìġğǵĥ
```

As you can see with one-hot vector targets, or scalar values that represent them, we can make some simple assumptions and use a more basic calculation — what was once an involved formula reduces to the negative log of the target class' confidence score — the second formula presented at the beginning of this chapter.

As we've already discussed, the example confidence level might look like `äflgğǵd flgħd flgĝíā` or `äflgġǵd flgġħd flgġǵā`. In both cases, the *argmax* of these vectors will return the second class as the prediction, but the model's confidence about these predictions is high only for one of them. The **Categorical Cross-Entropy Loss** accounts for that and outputs a larger loss the lower the confidence is:

```
ĨİĴIJĹĻ İĞĻÎ

ĴĹĨĮĻâİĞĻÎgĬIJÍâĝgãã
ĴĹĨĮĻâİĞĻÎgĬIJÍâflgîĥãã
ĴĹĨĮĻâİĞĻÎgĬIJÍâflgîãã
ĴĹĨĮĻâİĞĻÎgĬIJÍâflgíãã
ĴĹĨĮĻâďgggďã
ĴĹĨĮĻâİĞĻÎgĬIJÍâflgğãã
ĴĹĨĮĻâİĞĻÎgĬIJÍâflgĝãã
ĴĹĨĮĻâİĞĻÎgĬIJÍâflgflĥãã
ĴĹĨĮĻâİĞĻÎgĬIJÍâflgflĝãã


ńńń
flgfl
hflgflĥĝğîġğîġġíìĥĥflĥí
hflgĝflĥġħflĥĝĥħĥìíġħğí
hflgğğġĝĝġĥĥĝġĝĝğflîì
ggg
hĝgħflîġġìîĝğġġĝĝflflġ
hğgġflğĥíĥflîğîîġflġĥĥ
hğgîîĥìġğğìġĥĥġîîĝ
hġgħflĥĝìflĝíĥîííflîĝ
```

We've printed different log values for a few example confidences. When the confidence level equals *1*, meaning the model is 100% "sure" about its prediction, the loss value for this sample equals *0*. The loss value raises with the confidence level, approaching 0. You might also wonder why we did not print the result of *log(0)* — we'll explain that shortly.

So far, we've applied log() to the softmax output, but have neither explained what "log" is nor why we use it. We will save the discussion of "why" until the next chapter, which covers derivatives, gradients, and optimizations; suffice it to say that the log function has some desirable properties. **Log** is short for **logarithm** and is defined as the solution for the x-term in an equation of the form $a^x = b$. For example, $10^x = 100$ can be solved with a log: $log_{10}(100)$, which evaluates to 2. This property of the log function is *especially* beneficial when *e* (Euler's number or *~2.71828*) is used in the base (where 10 is in the example). The logarithm with *e* as its base is referred to as the **natural logarithm**, **natural log**, or simply **log** — you may also see this written as **ln**: $ln(x) = log(x) = log_e(x)$ The variety of conventions can make this confusing, so to simplify things, **any mention of log will always be a natural logarithm throughout this book**. The natural log represents the solution for the x-term in the equation $e^x = b$; for example, $e^x = 5.2$ is solved by *log(5.2)*.

In Python code:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

Ġ ł ĥgğ
ĴĹĨĮĻâĮĴgĬIJÍâĠãã


ńńń
ĝgħǵíħĥíħğĥĥíìġíĝħ
```

We can confirm this by exponentiating our result:

```
ĨİĴIJĹĻ İĞĻÎ

ĴĹĨĮĻâİĞĻÎgĦ ǻǻ ĝgħǵíħĥíħğĥĥíìġíĝħã


ńńń
ĥgĝîîîîîîîîîîîîî
```

The small difference is the result of floating-point precision in Python. Getting back to the loss calculation, we need to modify our output in two additional ways. First, we'll update our process to work on batches of softmax output distributions; and second, make the negative log calculation dynamic to the target index (the target index has been hard-coded so far).

Consider a scenario with a neural network that performs classification between three classes, and the neural network classifies in batches of three. After running through the softmax activation function with a batch of 3 samples and 3 classes, the network's output layer yields:

```
ﬁ PĹIJĠĞĠĨĬĨĻĨĦĹ ÌIJĹ ġ ĽĞİĴĬĦĹ
ĽIJÌĻİĞŇçIJŁĻĴŁĻĽ ł ĮĴgĞĹĹĞÑâääﬂgìd ﬂgĝd ﬂgğād
                                äﬂgĝd ﬂgĥd ﬂgǵād
                                äﬂgﬂğd ﬂgîd ﬂgﬂíāāã
```

We need a way to dynamically calculate the categorical cross-entropy, which we now know is a negative log calculation. To determine which value in the softmax output to calculate the negative log from, we simply need to know our target values. In this example, there are 3 classes; let's say we're trying to classify something as a "dog," "cat," or "human." A dog is class 0 (at index 0), a cat class 1 (index 1), and a human class 2 (index 2). Let's assume the batch of three sample inputs to this neural network is being mapped to the target values of a dog, cat, and cat. So the targets (as

a list of target indices) would be *[0, 1, 1]*.

```
ĽIJÌĻİĞŇçIJŁĻĴŁĻĽ ł ääflgìd flgĝd flgğād
                   äflgĝd flgĥd flgǵād
                   äflgflğd flgîd flgflíāā

ĢĬĞĽĽçĻĞĹÍĦĻĽ ł äfld ĝd ĝā  fi ĤIJÍd ĢĞĻd ĢĞĻ
```

The first value, 0, in ĢĬĞĽĽçĻĞĹÍĦĻĽ means the first softmax output distribution's intended prediction was the one at the 0th index of äflgìd flgĝd flgğā; the model has a flgì confidence score that this observation is a dog. This continues throughout the batch, where the intended target of the 2nd softmax distribution, äflgĝd flgĥd flgǵā, was at an index of 1; the model only has a flgĥ confidence score that this is a cat — the model is less certain about this observation. In the last sample, it's also the 2nd index from the softmax distribution, a value of flgî in this case — a pretty high confidence.

With a collection of softmax outputs and their intended targets, we can map these indices to retrieve the values from the softmax distributions:

```
ĽIJÌĻİĞŇçIJŁĻĴŁĻĽ ł ääflgìd flgĝd flgğād
                   äflgĝd flgĥd flgǵād
                   äflgflğd flgîd flgflíāā

ĢĬĞĽĽçĻĞĹÍĦĻĽ ł äfld ĝd ĝā

ÌIJĹ ĻĞĹÍçĨĤŇd ĤĨĽĻĹĨĠŁĻĨIJĮ ĨĮ ŅĨĴâĢĬĞĽĽçĻĞĹÍĦĻĽd ĽIJÌĻİĞŇçIJŁĻĴŁĻĽãf
    ĴĹĨĮĻâĤĨĽĻĹĨĠŁĻĨIJĮäĻĞĹÍçĨĤŇāã


ńńń
flgì
flgĥ
flgî
```

The ŅĨĴâã function, again, lets us iterate over multiple iterables at the same time in Python. This can be further simplified using NumPy (we're creating a NumPy array of the Softmax outputs this time):

```
ĽIJÌĻİĞŇçIJŁĻĴŁĻĽ ł ĮĴgĞĹĹĞÑâääflgìd flgĝd flgğād
                                äflgĝd flgĥd flgǵād
                                äflgflğd flgîd flgflíāāã
ĢĬĞĽĽçĻĞĹÍĦĻĽ ł äfld ĝd ĝā
```

```
ĴĹĨĮĻâĽIJÌĻİĞŇçIJŁĻĴŁĻĽääfld ĝd ğād ĢĬĞĽĽçĻĞĹÍĦĻĽāã


ńńń
äflgì flgĥ flgîā
```

What are the 0, 1, and 2 values? NumPy lets us index an array in multiple ways. One of them is to use a list filled with indices and that's convenient for us — we could use the ĢĬĞĽĽçĻĞĹÍĦĻĽ for this purpose as it already contains the list of indices that we are interested in. The problem is that this has to filter data rows in the array — the second dimension. To perform that, we also need to explicitly filter this array in its first dimension. This dimension contains the predictions and we, of course, want to retain them all. We can achieve that by using a list containing numbers from 0 through all of the indices. We know we're going to have as many indices as distributions in our entire batch, so we can use a ĹĞĮÍĦāã instead of typing each value ourselves:

```
ĴĹĨĮĻâĽIJÌĻİĞŇçIJŁĻĴŁĻĽä
    ĹĞĮÍĦâĬĦĮâĽIJÌĻİĞŇçIJŁĻĴŁĻĽãād ĢĬĞĽĽçĻĞĹÍĦĻĽ
āã


ńńń
äflgì flgĥ flgîā
```

This returns a list of the confidences at the target indices for each of the samples. Now we apply the negative log to this list:

```
ĴĹĨĮĻâhĮĴgĬIJÍâĽIJÌĻİĞŇçIJŁĻĴŁĻĽä
    ĹĞĮÍĦâĬĦĮâĽIJÌĻİĞŇçIJŁĻĴŁĻĽãād ĢĬĞĽĽçĻĞĹÍĦĻĽ
āāā


ńńń
äflgġĥĥìġîġ flgĥîġĝġìĝí flgĝflĥġĥflĥğā
```

Finally, we want an average loss per batch to have an idea about how our model is doing during training. There are many ways to calculate an average in Python; the most basic form of an average is the **arithmetic mean**: *sum(iterable) / len(iterable)*. NumPy has a method that computes this average on arrays, so we will use that instead. We add NumPy's average to the code:

```
ĮĦÍçĬIJÍ ł hĮĴgĬIJÍâĽIJÌĻİĞŇçIJŁĻĴŁĻĽä
              ĹĞĮÍĦâĬĦĮâĽIJÌĻİĞŇçIJŁĻĴŁĻĽããd ĢĬĞĽĽçĻĞĹÍĦĻĽ
          āã
ĞĿĦĹĞÍĦçĬIJĽĽ ł ĮĴgİĦĞĮâĮĦÍçĬIJÍã
ĴĹĨĮĻâĞĿĦĹĞÍĦçĬIJĽĽã


ńńń
flgġíĥflħflííflflĥğĝħíflġ
```

We have already learned that targets can be one-hot encoded, where all values, except for one, are zeros, and the correct label's position is filled with 1. They can also be sparse, which means that the numbers they contain are the correct class numbers — we are generating them this way with the `ĽĴĨĹĞĬçĤĞĻĞâã` function, and we can allow the loss calculation to accept any of these forms. Since we implemented this to work with sparse labels (as in our training data), we have to add a check if they are one-hot encoded and handle it a bit differently in this new case. The check can be performed by counting the dimensions — if targets are single-dimensional (like a list), they are sparse, but if there are 2 dimensions (like a list of lists), then there is a set of one-hot encoded vectors. In this second case, we'll implement a solution using the first equation from this chapter, instead of filtering out the confidences at the target labels. We have to multiply confidences by the targets, zeroing out all values except the ones at correct labels, performing a sum along the row axis (axis *1*). We have to add a test to the code we just wrote for the number of dimensions, move calculations of the log values outside of this new *if* statement, and implement the solution for the one-hot encoded labels following the first equation:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ĽIJÌĻİĞŇçIJŁĻĴŁĻĽ ł ĮĴgĞĹĹĞÑâääflgìd flgĝd flgğād
                              äflgĝd flgĥd flgġād
                              äflgflğd flgîd flgflíāāã
ĢĬĞĽĽçĻĞĹÍĦĻĽ ł ĮĴgĞĹĹĞÑâäägd fld flād
                            äfld ĝd flād
                            äfld ĝd flāāã


fi PĹIJĠĞĠĨĬĬĨĻĨĦĽ ÌIJĹ ĻĞĹÍĦĻ ĿĞĬŁĦĽ h
fi IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĽ
ĨÌ ĬĦĮâĢĬĞĽĽçĻĞĹÍĦĻĽgĽÎĞĴĦã łł ĝf
    ĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĽ ł ĽIJÌĻİĞŇçIJŁĻĴŁĻĽä
        ĹĞĮÍĦâĬĦĮâĽIJÌĻİĞŇçIJŁĻĴŁĻĽããd
        ĢĬĞĽĽçĻĞĹÍĦĻĽ
    ā
```

```
fi MĞĹĪ ĽĞĬŁĦĹ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJÂĦÂ ĬĞĠĦĬĹ
ĦĬĨÌ ĬĦĮâĢĬĞĹĽçĻĞĹÍĦĻĽgĹÎĞĴĦã łł ğf
    ĢIJĹĹĦĢĻçĢIJĮÌĨÎĦĦĮĢĦĹ ł ĮĴgĹŁİâ
        ĹIJÌĻİĞÑçIJŁĻĴŁĻĹ ǻ ĢĬĞĹĽçĻĞĹÍĦĻĹd
        ĞÑĨĹłĝ
    ã

fi LIJĹĹĦĹ
ĮĦÍçĬIJÍ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨÎĦĦĮĢĦĹã


ĞĽĦĹĞÍĦçĬIJĹĹ ł ĮĴgİĦĞĮâĮĦÍçĬIJÍã
ĴĹĨĮĻâĞĽĦĹĞÍĦçĬIJĹĹã
```

Before we move on, there is one additional problem to solve. The softmax output, which is also an input to this loss function, consists of numbers in the range from 0 to 1 - a list of confidences. It is possible that the model will have full confidence for one label making all the remaining confidences zero. Similarly, it is also possible that the model will assign full confidence to a value that wasn't the target. If we then try to calculate the loss of this confidence of 0:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĹ ĮĴ
hĮĴgĬIJÍâflã


ńńń
ççİĞĨĮççfĝf RŁĮĻĨİĦWĞĹĮĨĮÍf ÂĨĽĨÂĦ ĠÑ ŅĦĹIJ ĦĮĢIJŁĮĻĦĹĦÂ ĨĮ ĬIJÍ
ĨĮÌ
```

Before we explain this, we need to talk about *log(0)*. From the mathematical point of view, *log(0)* is undefined. We already know the following dependence: if $y=log(x)$, then $e^y=x$. The question of what the resulting y is in $y=log(0)$ is the same as the question of what's the y in $e^y=0$. In simplified terms, the constant e to any power is always a positive number, and there is no y resulting in $e^y=0$. This means the *log(0)* is undefined. We need to be aware of what the *log(0)* is, and "undefined" does not mean that we don't know anything about it. Since *log(0)* is undefined, what's the result for a value very close to *0*? We can calculate the limit of a function. How to exactly calculate it exceeds this book, but the solution is:

$$\lim_{x \to 0^+} log(x) = -\infty$$

We read it as the limit of a natural logarithm of x, with x approaching *0* from a positive (it is

impossible to calculate the natural logarithm of a negative value) equals negative infinity. What this means is that the limit is negative infinity for an infinitely small *x*, where *x* never reaches *0*.

The situation is a bit different in programming languages. We do not have limits here, just a function which, given a parameter, returns some value. The negative natural logarithm of *0*, in Python with NumPy, equals an infinitely big number, rather than undefined, and prints a warning about a division by *0* (which is a result of how this calculation is done). If `-ĮĴgĬIJÍâfłã` equals `ÕĮÌ`, is it possible to calculate e to the power of negative infinity with Python?

```
ĮĴgĦŐŐâhĮĴgÕĮÌã

ńńń
łgfł
```

In programming, the fewer things that are undefined, the better. Later on, we'll see similar simplifications, for example when calculating a derivative of the absolute value function, which does not exist for an input of *0* and we'll have to make some decisions to work around this.

Back to the result of `ÕĮÌ` for `hĮĴgĬIJÍâfłã` — as much as that makes sense, since the model would be fully wrong, this will be a problem for us to do further calculations with. Later, with optimization, we will also have a problem calculating gradients, starting with a mean value of all sample-wise losses since a single infinite value in a list will cause the average of that list to also be infinite:

```
ÕĬĴIJĹĻ ĮŁĬĴÑ ĞĽ ĮĴ
ĮĴgĬĦĞĮâäĝd ğd ġd hĮĴgĬIJÍâfłããã


ńńń
ççĬĞÕĮççfĝf RŁĮĻÕĬĬĦWĞĹĮÕĮÍf ÂÕĿÕÂĦ ĠÑ ŅĦĹIJ ĦĮĢIJŁĮĻĦĹĦÂ ÕĮ ĬIJÍ
ÕĮÌ
```

We could add a very small value to the confidence to prevent it from being a zero, for example, *1e-7*:

```
hĮĴgĬIJÍâĝĦhìã


ńńń
ĝĥgĝĝíflîĥĥĥflîĥíġğ
```

Adding a very small value, one-tenth of a million, to the confidence at its far edge will insignificantly impact the result, but this method yields an additional 2 issues. First, in the case where the confidence value is *1*:

```
hĮĴgĬIJÍâĝĸĝĦhìã


ńńń
hîgîîîîîîĥflĥíġíìflġĦhflí
```

When the model is fully correct in a prediction and puts all the confidence in the correct label, loss becomes a negative value instead of being 0. The other problem here is shifting confidence towards *1*, even if by a very small value. To prevent both issues, it's better to clip values from both sides by the same number, *1e-7* in our case. That means that the lowest possible value will become *1e-7* (like in the demonstration we just performed) but the highest possible value, instead of being *1+1e-7*, will become *1-1e-7* (so slightly less than *1*):

```
hĮĴgĬIJÍâĝhĝĦhìã

ńńń
ĝgflflflflflflflġîġìġħġìġĦhflì
```

This will prevent loss from being exactly *0*, making it a very small value instead, but won't make it a negative value and won't bias overall loss towards *1*. Within our code and using numpy, we'll accomplish that using `ĮĴgĢĬĨĴâã` method:

```
ÑçĴĹĦĤçĢĬĨĴĴĦĤ ł ĮĴgĢĬĨĴâÑçĴĹĦĤd ĝĦhìd ĝ h ĝĦhìã
```

This method can perform clipping on an array of values, so we can apply it to the predictions directly and save this as a separate array, which we'll use shortly.

The Categorical Cross-Entropy Loss Class

In the later chapters, we'll be adding more loss functions and some of the operations that we'll be performing are common for all of them. One of these operations is how we calculate the overall loss — no matter which loss function we'll use, the overall loss is always a mean value of all sample losses. Let's create the `LIJĽĽ` class containing the `ĢĞĬĢŁĬĞĻĦ` method that will call our loss object's forward method and calculate the mean value of the returned sample losses:

```
class CIJİİIJĮ ĬIJĽĽ ĢĬĞĽĽ
ĢĬĞĽĽ LIJĽĽf

    def CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽĦĽ
    def ÍĨĿĦĮ İIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñãf

        def CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦ ĬIJĽĽĦĽ
        ĽĞİĴĬĦçĬIJĽĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        def CĞĬĢŁĬĞĻĦ İĦĞĮ ĬIJĽĽ
        ÂĞĻĞçĬIJĽĽ ł ĮĴgİĦĞĮâĽĞİĴĬĦçĬIJĽĽĦĽã

        def RĦĻŁĹĮ ĬIJĽĽ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĽĽ
```

In later chapters, we'll add more code to this class, and the reason for it to exist will become more clear. For now, we'll use it for this single purpose.

Let's convert our loss code into a class for convenience down the line:

```
class CategoricalCrossEntropyLoss(Loss):

    def forward(self, y_pred, y_true):

        # Number of samples in a batch
        samples = len(y_pred)

        # Clip data to prevent division by 0
        # Clipped data will affect both sides
        y_pred_clipped = np.clip(y_pred,
                         1e-7, 1 - 1e-7)

        # Probabilities for target values -
        # only if categorical labels
        if isinstance(y_true, np.ndarray):
            correct_confidences = y_pred_clipped[
                range(samples),
                y_true
            ]

        # Mask values - only for one-hot encoded labels
        elif isinstance(y_true, np.ndarray):
            correct_confidences = np.sum(
                y_pred_clipped *
                y_true,
                axis=1
            )

        # Losses
        negative_log_likelihoods = -np.log(correct_confidences)
        return negative_log_likelihoods
    # Calculates the data and regularization losses
    def calculate(self, output, y):
        data_loss = self.forward(output, y)
        return data_loss
```

This class inherits the `Loss` class and performs all the error calculations that we derived throughout this chapter and can be used as an object. For example, using the manually-created output and targets:

```
loss_function = Loss_CategoricalCrossEntropy()
loss = loss_function.calculate(softmax_outputs, class_targets)
print(loss)
```

```
0.3857096
```

Combining everything up to this point:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĨİĴIJĹĻ ĽĴĨĹĞĬçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


ſi DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ſi LĞÑĦĹ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽãf
        ſi IĮĨĻĨĞĬĨŅĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍÎĻĽ ł ſlgſlĝ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã

    ſi FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ſi CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ŁĞĬŁĦĽ ÌĹIJİ ĨĮĴŁĻĽd ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĽĦĬÌgŃĦĨÍÎĻĽã ĸ ĽĦĬÌgĠĨĞĽĦĽ


ſi RĦLU ĞĢĻĨŁĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨŁĞĻĨIJĮçRĦLUf

    ſi FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ſi CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ŁĞĬŁĦĽ ÌĹIJİ ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĬİŁİâſld ĨĮĴŁĻĽã
```

```
# SIJÌĻİĞŇ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇf

    # FIJĹŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ÌIJĹŃĞĹÎâĽĦĬÌd ĨĮĴŁĻĽãf

        # GĦĻ ŁĮĮIJĹİĞĬĨŢĦÎ ĴĹIJĠĞĠĨĬĨĻĨĦĽ
        ĦŇĴçĿĞĬŁĦĽ ł ĮĴgĦŇĴâĨĮĴŁĻĽ h ĮĴgİĞŇâĨĮĴŁĻĽd ĞŇĨĽłĝd
                                                ĪĦĦĴÎĨİĽłTĹŁĦãã
        # NIJĹİĞĬĨŢĦ ĻÎĦİ ÌIJĹ ĦĞĢÎ ĽĞİĴĬĦ
        ĴĹIJĠĞĠĨĬĨĻĨĦĽ ł ĦŇĴçĿĞĬŁĦĽ w ĮĴgĽŁİâĦŇĴçĿĞĬŁĦĽd ĞŇĨĽłĝd
                                                ĪĦĦĴÎĨİĽłTĹŁĦã

        ĽĦĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĨĬĨĻĨĦĽ


# CIJİİIJĮ ĬIJĽĽ ĢĬĞĽĽ
ĢĬĞĽĽ LIJĽĽf

    # CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÎĞĻĞ ĞĮÎ ĹĦÍŁĬĞĹĨŢĞĻĨIJĮ ĬIJĽĽĦĽ
    # ÍĨĿĦĮ İIJÎĦĬ IJŁĻĴŁĻ ĞĮÎ ÍĹIJŁĮÎ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ÎĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñãf

        # CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦ ĬIJĽĽĦĽ
        ĽĞİĴĬĦçĬIJĽĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÎâIJŁĻĴŁĻd Ñã

        # CĞĬĢŁĬĞĻĦ İĦĞĮ ĬIJĽĽ
        ÎĞĻĞçĬIJĽĽ ł ĮĴgİĦĞĮâĽĞİĴĬĦçĬIJĽĽĦĽã

        # RĦĻŁĹĮ ĬIJĽĽ
        ĹĦĻŁĹĮ ÎĞĻĞçĬIJĽĽ


# CĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ
ĢĬĞĽĽ LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâLIJĽĽãf

    # FIJĹŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ÌIJĹŃĞĹÎâĽĦĬÌd ÑçĴĹĦÎd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ ĨĮ Ğ ĠĞĻĢÎ
        ĽĞİĴĬĦĽ ł ĬĦĮâÑçĴĹĦÎã

        # CĬĨĴ ÎĞĻĞ ĻIJ ĴĹĦĿĦĮĻ ÎĨĿĨĽĨIJĮ ĠÑ #
        # CĬĨĴ ĠIJĻÎ ĽĨÎĦĽ ĻIJ ĮIJĻ ÎĹĞÍ İĦĞĮ ĻIJŃĞĹÎĽ ĞĮÑ ĿĞĬŁĦ
        ÑçĴĹĦÎçĢĬĨĴĴĦÎ ł ĮĴgĢĬĨĴâÑçĴĹĦÎd ĝĦhìd ĝ h ĝĦhìã
```

```
        fi PĹIJĠĞĠĞĨĬĨĨĻĨĦĽ ÌIJĹ ĻĞĹÍĦĻ ŁĠĬŁĦĽ h
        fi IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÎĦĮĢĦĽ ł ÑçĴĹĦÂçĢĬĨĴĴĦÂä
                ĹĞĮÍĦâĽĞİĴĬĦĽãd
                ÑçĻĹŁĦ
            ã

        fi MĞĽĨ ŁĠĬŁĦĽ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJÂĦÂ ĬĞĠĦĬĽ
        ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÎĦĮĢĦĽ ł ĮĴgĽŁİâ
                ÑçĴĹĦÂçĢĬĨĴĴĦÂ ǻ ÑçĻĹŁĦd
                ĞÑĨĽłĝ
            ã

        fi LIJĽĽĦĽ
        ĮĦÍĞĻĨŁĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĽ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĽã
        ĹĦĻŁĹĮ ĮĦÍĞĻĨŁĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĽ
```

```
fi CĹĦĞĻĦ ÂĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçÂĞĻĞâĽĞİĴĬĦĽłĝflld ĢĬĞĽĽĦĽłġã

fi CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ÑĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮÂ ġ IJŁĻĴŁĻ ŁĠĬŁĦĽ
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ġã

fi CĹĦĞĻĦ RĦLU ĞĢĻĨŁĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦÂ ÑĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨŁĞĻĨIJĮĝ ł AĢĻĨŁĞĻĨIJĮçRĦLUâã

fi CĹĦĞĻĦ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ ÑĨĻÎ ġ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ÑĦ ĻĞĨĦ IJŁĻĴŁĻ
fi IJÌ ĴĹĦŁĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮÂ ġ IJŁĻĴŁĻ ŁĠĬŁĦĽ
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâġd ġã

fi CĹĦĞĻĦ SIJÌĻİĞÑ ĞĢĻĨŁĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦÂ ÑĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨŁĞĻĨIJĮğ ł AĢĻĨŁĞĻĨIJĮçSIJÌĻİĞÑâã

fi CĹĦĞĻĦ ĬIJĽĽ ÌŁĮĢĻĨIJĮ
ĬIJĽĽçÌŁĮĢĻĨIJĮ ł LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

fi PĦĹÌIJĹİ Ğ ÌIJĹÑĞĹÂ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ÂĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑĦĹ
ÂĦĮĽĦĝgÌIJĹÑĞĹÂâXã

fi PĦĹÌIJĹİ Ğ ÌIJĹÑĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨŁĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
fi ĨĻ ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨŁĞĻĨIJĮĝgÌIJĹÑĞĹÂâÂĦĮĽĦĝgIJŁĻĴŁĻã
```

```
fi PHĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽHĢIJĮÂ DHĮĽH ĬĞÑHĹ
fi ĨĻ ĻĞĪHĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑHĹ ĞĽ ĨĮĴŁĻĽ
ÂHĮĽHğgÌIJĹŃĞĹÂâĞĢĻĨĿĞĻĨIJĮĝgIJŁĻĴŁĻã

fi PHĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
fi ĨĻ ĻĞĪHĽ ĻÎH IJŁĻĴŁĻ IJÌ ĽHĢIJĮÂ ÂHĮĽH ĬĞÑHĹ ÎHĹH
ĞĢĻĨĿĞĻĨIJĮğgÌIJĹŃĞĹÂâÂHĮĽHğgIJŁĻĴŁĻã

fi LHĻďĽ ĽHH IJŁĻĴŁĻ IJÌ ĻÎH ÌĨĹĽĻ ÌHŃ ĽĞİĴĬHĽf
ĴĹĨĮĻâĞĢĻĨĿĞĻĨIJĮğgIJŁĻĴŁĻäfĥāã

fi PHĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĬIJĽĽ ÌŁĮĢĻĨIJĮ
fi ĨĻ ĻĞĪHĽ ĻÎH IJŁĻĴŁĻ IJÌ ĽHĢIJĮÂ ÂHĮĽH ĬĞÑHĹ ÎHĹH ĞĮÂ ĹHĻŁĹĮĽ ĬIJĽĽ
ĬIJĽĽ ł ĬIJĽĽçÌŁĮĢĻĨIJĮgĢĞĬĢŁĬĞĻHâĞĢĻĨĿĞĻĨIJĮğgIJŁĻĴŁĻd Ñã

fi PĹĨĮĻ ĬIJĽĽ ĿĞĬŁH
ĴĹĨĮĻâďĬIJĽĽfďd ĬIJĽĽã


ńńń
ääflgġġġġġġġ flgġġġġġġġ flgġġġġġġġgā
 äflgġġġġġġĝħ flgġġġġġġğ  flgġġġġġġħgā
 äflgġġġġġğíì flgġġġġġğî  flgġġġġġġĝíā
 äflgġġġġġğħ  flgġġġġġğħġ flgġġġġġìì ā
 äflgġġġġğğġ flgġġġġğğ  flgġġġġġĥğíāā
ĬIJĽĽf ĝgflîíħĝflġ
```

Again, we get *~0.33* values since the model is random, and its average loss is also not great for these data, as we've not yet trained our model on how to correct its errors.

Accuracy Calculation

While loss is a useful metric for optimizing a model, the metric commonly used in practice along with loss is the **accuracy**, which describes how often the largest confidence is the correct class in terms of a fraction. Conveniently, we can reuse existing variable definitions to calculate the accuracy metric. We will use the *argmax* values from the *softmax outputs* and then compare these to the targets. This is as simple as doing (note that we slightly modified the `ĽIJÌĻİĞŇçIJŁĻĴŁĻĽ` for the purpose of this example):

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

fi PĹIJĠĞĠĨĬĨĻĨĦĽ IJÌ ġ ĽĞİĴĬĦĽ
ĽIJÌĻİĞŇçIJŁĻĴŁĻĽ ł ĮĴgĞĹĹĞÑâääflgìd flgğd flgĝād
                            äflgĥd flgĝd flgġād
                            äflgflğd flgîd flgflíāāã
fi TĞĹÍĦĻ âÍĹIJŁĮĤhĻĹŁĻÎã ĬĞĠĦĬĽ ÌIJĹ ġ ĽĞİĴĬĦĽ
ĢĬĞĽĽçĻĞĹÍĦĻĽ ł ĮĴgĞĹĹĞÑâäfld ĝd ĝāã

fi CĞĬĢŁĬĞĻĦ ŁĠĬŁĦĽ ĞĬIJĮÍ ĽĦĢIJĮĤ ĞŇĨĽ âĞŇĨĽ IJÌ ĨĮĤĦŇ ĝã
ĴĹĦĤĨĢĻĨIJĮĽ ł ĮĴgĞĹÍİĞŇâĽIJÌĻİĞŇçIJŁĻĴŁĻĽd ĞŇĨĽłĝã
fi IÌ ĻĞĹÍĦĻĽ ĞĹĦ IJĮĦhÎIJĻ ĦĮĢIJĤĦĤ h ĢIJĮŁĦĹĻ ĻÎĦİ
ĨÌ ĬĦĮâĢĬĞĽĽçĻĞĹÍĦĻĽgĽÎĞĴĦã łł ğf
    ĢĬĞĽĽçĻĞĹÍĦĻĽ ł ĮĴgĞĹÍİĞŇâĢĬĞĽĽçĻĞĹÍĦĻĽd ĞŇĨĽłĝã
fi TĹŁĦ ĦŁĞĬŁĞĻĦĽ ĻIJ ĝe FĞĬĽĦ ĻIJ fl
ĞĢĢŁĹĞĢŇ ł ĮĴgİĦĞĮâĴĹĦĤĨĢĻĨIJĮĽłłĢĬĞĽĽçĻĞĹÍĦĻĽã


ĴĹĨĮĻâďĞĢĢfďd ĞĢĢŁĹĞĢŇã


ńńń
ĞĢĢf flghhhhhhhhhhhhhhhh
```

We are also handling one-hot encoded targets by converting them to sparse values using `ĮĴgĞĹÍİĞŇâã`.

We can add the following to the end of our full script above to calculate its accuracy:

```
# CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĿĞĻĨIJĮğ ĞĮĤ ĻĞĹÍĦĻĹ
# ĢĞĬĢŁĬĞĻĦ ĿĞĬŁĦĹ ĞĬIJĮÍ ÌĨĹĹĻ ĞÑĨĹ
ĴĹĦĤĨĢĻĨIJĮĹ ł ĮĴgĞĹÍİĞÑâĞĢĻĨĿĞĻĨIJĮğglJŁĻĴŁĻd ĞÑĨĹłĝã
ĨÌ ĬĦĮâÑgĹÎĞĴĦã łł ğf
    Ñ ł ĮĴgĞĹÍİĞÑâÑd ĞÑĨĹłĝã
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦĤĨĢĻĨIJĮĹłłÑã

# PĹĨĮĻ ĞĢĢŁĹĞĢÑ
ĴĹĨĮĻâďĞĢĢfďd ĞĢĢŁĹĞĢÑã


ńńń
ĞĢĢf flgġġ
```

Now that you’ve learned how to perform a forward pass through our network and calculate the metrics to signal if the model is performing poorly, we will embark on optimization in the next chapter!

Supplementary Material: https://nnfs.io/ch5
Chapter code, further resources, and errata for this chapter.

Chapter 6

Introducing Optimization

Now that the neural network is built, able to have data passed through it, and capable of calculating loss, the next step is to determine how to adjust the weights and biases to decrease the loss. Finding an intelligent way to adjust the neurons' input's weights and biases to minimize loss is the main difficulty of neural networks.

The first option one might think of is randomly changing the weights, checking the loss, and repeating this until happy with the lowest loss found. To see this in action, we'll use a simpler dataset than we've been working with so far:

```
ÕİĴIJĹĻ İĞĻĴĬIJĻĬÕġgĴÑĴĬIJĻ ĞĽ ĴĬĻ

ÕİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ÕİĴIJĹĻ ĽĦĹĻĬĢĞĬçĤĞĻĞ

ĮĮÌĽgÕĬĮÕĻâã
```

```
Xd Ñ ł LĦĹĻĨĢĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

ĴĬĻgĽĢĞĻĻĦĹâXäfd flād Xäfd ĝād Ģł Ñd ĽłġFld ĢİĞĴłďĠĹÍďã
ĴĬĻgĽÎIJŃâã
```

Which looks like:

Fig 6.01: "Vertical data" graphed.

Using the previously created code up to this point, we can use this new dataset with a simple neural network:

```
fi CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł LĦĹĻĨĢĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

fi CĹĦĞĻĦ İIJĤĦĬ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ġã  fi ÌĨĹĽĻ ĤĦĮĽĦ ĬĞÑĦĹd ğ ĨĮĴłĻĽ
ĞĢĻĨĽĞĻĨIJĮĝ ł AĢĻĨĽĞĻĨIJĮçRĦLUâã
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâġd ġã  fi ĽĦĢIJĮĤ ĤĦĮĽĦ ĬĞÑĦĹd ġ ĨĮĴłĻĽd ġ IJłĻĴłĻĽ
ĞĢĻĨĽĞĻĨIJĮğ ł AĢĻĨĽĞĻĨIJĮçSIJÌĻİĞŃâã

fi CĹĦĞĻĦ ĬIJĽĽ ÌłĮĢĻĨIJĮ
ĬIJĽĽçÌłĮĢĻĨIJĮ ł LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã
```

Then create some variables to track the best loss and the associated weights and biases:

```
# HHĬĴHĹ ĽĞĹĨĞĠĬHĽ
ĬIJŃHĽĻçĬIJĹĽ ł îîîîîîî  # ĽIJİH ĨĮĨĻĨĞĬ ĽĞĬłH
ĠHĽĻçĤHĮĹHĝçŃHĨÍÎĻĽ ł ĤHĮĹHĝgŃHĨÍÎĻĽgĢIJĴÑâã
ĠHĽĻçĤHĮĹHĝçĠĨĞĹHĽ ł ĤHĮĹHĝgĠĨĞĹHĽgĢIJĴÑâã
ĠHĽĻçĤHĮĹHğçŃHĨÍÎĻĽ ł ĤHĮĹHğgŃHĨÍÎĻĽgĢIJĴÑâã
ĠHĽĻçĤHĮĹHğçĠĨĞĹHĽ ł ĤHĮĹHğgĠĨĞĹHĽgĢIJĴÑâã
```

We initialized the loss to a large value and will decrease it when a new, lower, loss is found. We are also copying weights and biases (ĢIJĴÑâã ensures a full copy instead of a reference to the object). Now we iterate as many times as desired, pick random values for weights and biases, and save the weights and biases if they generate the lowest-seen loss:

```
ÌIJĹ ĨĻHĹĞĻĨIJĮ ĨĮ ĹĞĮÍHâĝflflflflãf

    # GHĮHĹĞĻH Ğ ĮHŃ ĽHĻ IJÌ ŃHĨÍÎĻĽ ÌIJĹ ĨĻHĹĞĻĨIJĮ
    ĤHĮĹHĝgŃHĨÍÎĻĽ ł flgflĥ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâğd ġã
    ĤHĮĹHĝgĠĨĞĹHĽ ł flgflĥ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĝd ġã
    ĤHĮĹHğgŃHĨÍÎĻĽ ł flgflĥ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâġd ġã
    ĤHĮĹHğgĠĨĞĹHĽ ł flgflĥ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĝd ġã

    # PHĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĽ IJÌ ĻĹĤ ĻĹĞĨĮĨĮÍ ĤĞĻĞ ĻĨĹIJłÍÎ ĻĨĨĽ ĬĞÑHĹ
    ĤHĮĹHĝgÌIJĹŃĞĹĤâXã
    ĞĢĻĨĽĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤHĮĹHĝgIJłĻĴłĻã
    ĤHĮĹHğgÌIJĹŃĞĹĤâĞĢĻĨĽĞĻĨIJĮĝgIJłĻĴłĻã
    ĞĢĻĨĽĞĻĨIJĮğgÌIJĹŃĞĹĤâĤHĮĹHğgIJłĻĴłĻã

    # PHĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĽ ĻĨĹIJłÍÎ ĞĢĻĨĽĞĻĨIJĮ ÌłĮĢĻĨIJĮ
    # ĨĻ ĻĞĨHĹ ĻĨH IJłĻĴłĻ IJÌ ĽHĢIJĮĤ ĤHĮĹH ĬĞÑHĹ ĨHĹH ĞĮĤ ĹHĻłĹĮĽ ĬIJĹĽ
    ĬIJĹĽ ł ĬIJĹĽçÌłĮĢĻĨIJĮgĢĞĬĢłĬĞĻHâĞĢĻĨĽĞĻĨIJĮğgIJłĻĴłĻd Ñã

    # CĞĬĢłĬĞĻH ĞĢĢłĹĞĢÑ ÌĹIJİ IJłĻĴłĻ IJÌ ĞĢĻĨĽĞĻĨIJĮğ ĞĮĤ ĻĞĹĺHĻĽ
    # ĢĞĬĢłĬĞĻH ĽĞĬłHĽ ĞĬIJĮÍ ÌĨĹĹĻ ĞŃĨĽ
    ĴĹHĤĨĢĻĨIJĮĽ ł ĮĴgĞĹÍİĞÑâĞĢĻĨĽĞĻĨIJĮğgIJłĻĴłĻd ĞŃĨĽłĝã
    ĞĢĢłĹĞĢÑ ł ĮĴgİHĞĮâĴĹHĤĨĢĻĨIJĮĽłłÑã

    # IÌ ĬIJĹĽ ĨĽ ĽİĞĬĬHĹ h ĴĹĨĮĻ ĞĮĤ ĽĞĽH ŃHĨÍÎĻĽ ĞĮĤ ĠĨĞĽHĽ ĞĽĨĤH
    ĨÌ ĬIJĹĽ l· ĬIJŃHĽĻçĬIJĹĽf
        ĴĹĨĮĻâďNHŃ ĽHĻ IJÌ ŃHĨÍÎĻĽ ÌIJłĮĤd ĨĻHĹĞĻĨIJĮfďd ĨĻHĹĞĻĨIJĮd
              ďĬIJĹĽfďd ĬIJĹĽd ďĞĢĢfďd ĞĢĢłĹĞĢÑã
        ĠHĽĻçĤHĮĹHĝçŃHĨÍÎĻĽ ł ĤHĮĹHĝgŃHĨÍÎĻĽgĢIJĴÑâã
        ĠHĽĻçĤHĮĹHĝçĠĨĞĹHĽ ł ĤHĮĹHĝgĠĨĞĹHĽgĢIJĴÑâã
        ĠHĽĻçĤHĮĹHğçŃHĨÍÎĻĽ ł ĤHĮĹHğgŃHĨÍÎĻĽgĢIJĴÑâã
        ĠHĽĻçĤHĮĹHğçĠĨĞĹHĽ ł ĤHĮĹHğgĠĨĞĹHĽgĢIJĴÑâã
        ĬIJŃHĽĻçĬIJĹĽ ł ĬIJĹĽ
```

```
ńńń
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf fl ĬIJĽĽf ĝgflîíħĥħġ ĞĢĢf
flgġġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ġ ĬIJĽĽf ĝgflîíĝġí ĞĢĢf
flgġġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ĝĝì ĬIJĽĽf ĝgflîíflĝĝĥ ĞĢĢf
flgġġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ĝğġ ĬIJĽĽf ĝgflîììĥĝħ ĞĢĢf flgħ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ĝħĥ ĬIJĽĽf ĝgflîìĥìĝ ĞĢĢf
flgġġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ĥĥğ ĬIJĽĽf ĝgflîìġħîġ ĞĢĢf flgġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ììí ĬIJĽĽf ĝgflîħíğĥì ĞĢĢf
flgġġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ġġflì ĬIJĽĽf ĝgflîħĥĥġġ ĞĢĢf
flgġġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ġħĝĥ ĬIJĽĽf ĝgflîħġġîî ĞĢĢf
flgġġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf îġĥfl ĬIJĽĽf ĝgflîħğîĥ ĞĢĢf
flgġġġġġġġġġġġġġġġġ
```

Loss certainly falls, though not by much. Accuracy did not improve, except for a singular situation where the model randomly found a set of weights yielding better accuracy. Still, with a fairly large loss, this state is not stable. Running an additional 90,000 iterations for 100,000 in total:

```
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ĝġġħĝ ĬIJĽĽf ĝgflîħġflĝġ ĞĢĢf
flgġġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ĝġflflĝ ĬIJĽĽf ĝgflîĥîíĥí ĞĢĢf
flgġġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ğġĥîí ĬIJĽĽf ĝgflîġìġġġ ĞĢĢf
flgġġġġġġġġġġġġġġġġ
```

Loss continued to drop, but accuracy did not change. This doesn't appear to be a reliable method for minimizing loss. After running for 1 billion iterations, the following was the best (lowest loss) result:

```
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ğğîíħĥflflfl ĬIJĽĽf ĝgflîĝĝġflĥ ĞĢĢf
flgġġġġġġġġġġġġġġġġ
```

Even with this basic dataset, we see that randomly searching for weight and bias combinations will take far too long to be an acceptable method. Another idea might be, instead of setting parameters with randomly-chosen values each iteration, apply a fraction of these values to parameters. With this, weights will be updated from what currently yields us the lowest loss instead of aimlessly randomly. If the adjustment decreases loss, we will make it the new point to adjust from. If loss instead increases due to the adjustment, then we will revert to the previous point. Using similar code from earlier, we will first change from randomly selecting weights and biases to randomly *adjusting* them:

```
ƒ UĴĤĞĻĦ ŃĦĨÍÎĻĽ ŃĨĻÎ ĹIJİĦ ĽİĞĬĬ ĹĞĮĤIJİ ĽĞĬŁĦĽ
ĤĦĮĽĦĝgŃĦĨÍÎĻĽ κł flgflĥ ấ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâğd ġã
ĤĦĮĽĦĝgĠĨĞĽĦĽ κł flgflĥ ấ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĝd ġã
ĤĦĮĽĦğgŃĦĨÍÎĻĽ κł flgflĥ ấ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâġd ġã
ĤĦĮĽĦğgĠĨĞĽĦĽ κł flgflĥ ấ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĝd ġã
```

Then we will change our ending ĨÌ statement to be:

```
ƒ IÌ ĬIJĹĽ ĨĽ ĽİĞĬĬĦĹ h ĴĹĨĮĻ ĞĮĤ ĽĞŁĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ ĞĽĨĤĦ
ĨÌ ĬIJĹĽ ł· ĬIJŃĦĽĻçĬIJĹĽf
    ĴĹĨĮĻâďNĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮfďd ĨĻĦĹĞĻĨIJĮd
          ďĬIJĹĽfďd ĬIJĹĽd ďĞĢĢfďd ĞĢĢŁĹĞĢÑã
    ĠĦĽĻçĤĦĮĽĦĝçŃĦĨÍÎĻĽ ł ĤĦĮĽĦĝgŃĦĨÍÎĻĽgĢIJĴÑâã
    ĠĦĽĻçĤĦĮĽĦĝçĠĨĞĽĦĽ ł ĤĦĮĽĦĝgĠĨĞĽĦĽgĢIJĴÑâã
    ĠĦĽĻçĤĦĮĽĦğçŃĦĨÍÎĻĽ ł ĤĦĮĽĦğgŃĦĨÍÎĻĽgĢIJĴÑâã
    ĠĦĽĻçĤĦĮĽĦğçĠĨĞĽĦĽ ł ĤĦĮĽĦğgĠĨĞĽĦĽgĢIJĴÑâã
    ĬIJŃĦĽĻçĬIJĹĽ ł ĬIJĹĽ
ƒ RĦŁĦĹĻ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
ĦĬĽĦf
    ĤĦĮĽĦĝgŃĦĨÍÎĻĽ ł ĠĦĽĻçĤĦĮĽĦĝçŃĦĨÍÎĻĽgĢIJĴÑâã
    ĤĦĮĽĦĝgĠĨĞĽĦĽ ł ĠĦĽĻçĤĦĮĽĦĝçĠĨĞĽĦĽgĢIJĴÑâã
    ĤĦĮĽĦğgŃĦĨÍÎĻĽ ł ĠĦĽĻçĤĦĮĽĦğçŃĦĨÍÎĻĽgĢIJĴÑâã
    ĤĦĮĽĦğgĠĨĞĽĦĽ ł ĠĦĽĻçĤĦĮĽĦğçĠĨĞĽĦĽgĢIJĴÑâã
```

Full code up to this point:

```
fi CĹĦĞĻĦ ÂĞĻĞĽĦĻ
Xd Ñ ł ĿĦĹĻĨĢĞĬçÂĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

fi CĹĦĞĻĦ İIJÂĦĬ
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ġã  fi ÌĨĹĽĻ ÂĦĮĽĦ ĬĞÑĦĹd ğ ĨĮĴłĻĽ
ĞĢĻĨĿĞĻĨIJĮĝ ł AĢĻĨĿĞĻĨIJĮçRĦLUâã
ÂĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâġd ġã  fi ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹd ġ ĨĮĴłĻĽd ġ IJłĻĴłĻĽ
ĞĢĻĨĿĞĻĨIJĮğ ł AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞÑâã

fi CĹĦĞĻĦ ĬIJĽĽ ÌłĮĢĻĨIJĮ
ĬIJĽĽçÌłĮĢĻĨIJĮ ł LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

fi ĦĦĬĴĦĹ ĿĞĹĨĞĠĬĦĽ
ĬIJŃĦĽĻçĬIJĽĽ ł îîîîîîî  fi ĽIJİĦ ĨĮĨĻĨĞĬ ĿĞĬłĦ
ĠĦĽĻçÂĦĮĽĦĝçŃĦĨÍÎĻĽ ł ÂĦĮĽĦĝgŃĦĨÍÎĻĽgĢIJĴÑâã
ĠĦĽĻçÂĦĮĽĦĝçĠĨĞĽĦĽ ł ÂĦĮĽĦĝgĠĨĞĽĦĽgĢIJĴÑâã
ĠĦĽĻçÂĦĮĽĦğçŃĦĨÍÎĻĽ ł ÂĦĮĽĦğgŃĦĨÍÎĻĽgĢIJĴÑâã
ĠĦĽĻçÂĦĮĽĦğçĠĨĞĽĦĽ ł ÂĦĮĽĦğgĠĨĞĽĦĽgĢIJĴÑâã

ÌIJĹ ĨĻĦĹĞĻĨIJĮ ĨĮ ĹĞĮÍĦâĝflflflfãf

    fi UĴÂĞĻĦ ŃĦĨÍÎĻĽ ŃĨĻÎ ĽIJİĦ ĽİĞĬĬ ĹĞĮÂIJİ ĿĞĬłĦĽ
    ÂĦĮĽĦĝgŃĦĨÍÎĻĽ κł flgflĥ ấ ĮĴgĹĞĮÂIJİgĹĞĮÂĮâğd ġã
    ÂĦĮĽĦĝgĠĨĞĽĦĽ κł flgflĥ ấ ĮĴgĹĞĮÂIJİgĹĞĮÂĮâĝd ġã
    ÂĦĮĽĦğgŃĦĨÍÎĻĽ κł flgflĥ ấ ĮĴgĹĞĮÂIJİgĹĞĮÂĮâġd ġã
    ÂĦĮĽĦğgĠĨĞĽĦĽ κł flgflĥ ấ ĮĴgĹĞĮÂIJİgĹĞĮÂĮâĝd ġã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ IJÌ IJłĹ ĻĹĞĨĮĨĮÍ ÂĞĻĞ ĻÎĹIJłÍÎ ĻÎĨĽ ĬĞÑĦĹ
    ÂĦĮĽĦĝgÌIJĹŃĞĹÂâXã
    ĞĢĻĨĿĞĻĨIJĮĝgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJłĻĴłĻã
    ÂĦĮĽĦğgÌIJĹŃĞĹÂâĞĢĻĨĿĞĻĨIJĮĝgIJłĻĴłĻã
    ĞĢĻĨĿĞĻĨIJĮğgÌIJĹŃĞĹÂâÂĦĮĽĦğgIJłĻĴłĻã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĞĢĻĨĿĞĻĨIJĮ ÌłĮĢĻĨIJĮ
    fi ĨĻ ĻĞĪĦĽ ĻÎĦ IJłĻĴłĻ IJÌ ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮÂ ĹĦĻłĹĮĽ ĬIJĽĽ
    ĬIJĽĽ ł ĬIJĽĽçÌłĮĢĻĨIJĮgĢĞĬĢłĬĞĻĦâĞĢĻĨĿĞĻĨIJĮğgIJłĻĴłĻd Ñã
```

```
ƒ CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨŁĞĻĨIJĮğ ĞĮĤ ĻĞĹÍĦĻĹ
ƒ ĢĞĬĢŁĬĞĻĦ ŁĞĬŁĦĹ ĞĬIJĮÍ ÌĬĹĹĻ ĞŃĬĹ
ĴĹĦĤĬĢĻĨIJĮĹ ł ĮĴgĞĹÍİĞŃâĞĢĻĨŁĞĻĨIJĮğgIJŁĻĴŁĻd ĞŃĬĹłĝã
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦĤĬĢĻĨIJĮĹłłÑã

ƒ IÌ ĬIJĹĹ ĨĹ ĹİĞĬĬĦĹ h ĴĹĨĮĻ ĞĮĤ ĹĞŁĦ ŃĦĬÍÎĻĹ ĞĮĤ ĠĬĞĹĦĹ ĞĹĬĤĦ
ĬÌ ĬIJĹĹ l· ĬIJŃĦĹĻçĬIJĹĹf
    ĴĹĨĮĻâďNĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮfďd ĨĻĦĹĞĻĨIJĮd
          ďĬIJĹĹfďd ĬIJĹĹd ďĞĢĢfďd ĞĢĢŁĹĞĢÑã
    ĠĦĹĻçĤĦĮĹĦĝçŃĦĬÍÎĻĹ ł ĤĦĮĹĦĝgŃĦĬÍÎĻĹgĢIJĴÑâã
    ĠĦĹĻçĤĦĮĹĦĝçĠĬĞĹĦĹ ł ĤĦĮĹĦĝgĠĬĞĹĦĹgĢIJĴÑâã
    ĠĦĹĻçĤĦĮĹĦğçŃĦĬÍÎĻĹ ł ĤĦĮĹĦğgŃĦĬÍÎĻĹgĢIJĴÑâã
    ĠĦĹĻçĤĦĮĹĦğçĠĬĞĹĦĹ ł ĤĦĮĹĦğgĠĬĞĹĦĹgĢIJĴÑâã
    ĬIJŃĦĹĻçĬIJĹĹ ł ĬIJĹĹ
ƒ RĦŁĦĹĻ ŃĦĬÍÎĻĹ ĞĮĤ ĠĬĞĹĦĹ
ĦĬĹĦf
    ĤĦĮĹĦĝgŃĦĬÍÎĻĹ ł ĠĦĹĻçĤĦĮĹĦĝçŃĦĬÍÎĻĹgĢIJĴÑâã
    ĤĦĮĹĦĝgĠĬĞĹĦĹ ł ĠĦĹĻçĤĦĮĹĦĝçĠĬĞĹĦĹgĢIJĴÑâã
    ĤĦĮĹĦğgŃĦĬÍÎĻĹ ł ĠĦĹĻçĤĦĮĹĦğçŃĦĬÍÎĻĹgĢIJĴÑâã
    ĤĦĮĹĦğgĠĬĞĹĦĹ ł ĠĦĹĻçĤĦĮĹĦğçĠĬĞĹĦĹgĢIJĴÑâã

ńńń
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf fl ĬIJĹĹf ĝgflîíìĥíġ ĞĢĢf
flgġġġġġġġġġġġġġġġġġ ggg
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ğî ĬIJĹĹf ĝgflìğĥğġġ ĞĢĢf
flgĥğĥĥĥĥĥĥĥĥĥĥĥĥĥ
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ġfl ĬIJĹĹf ĝgflìğġġġğ ĞĢĢf
flgġġĥĥĥĥĥĥĥĥĥĥĥĥĥì ggg
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ġí ĬIJĹĹf ĝgflġflġĥğğ ĞĢĢf
flgĥĥĥĥĥĥĥĥĥĥĥĥĥĥĥĥ
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ġî ĬIJĹĹf ĝgflğîğĥíĥ ĞĢĢf
flgĥĥĥĥĥĥĥĥĥĥĥĥĥĥĥĥ ggg
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf îì ĬIJĹĹf flgîğììġġĥ ĞĢĢf
flgìġġġġġġġġġġġġġġġġ ggg
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ĝĥğ ĬIJĹĹf flgìġġîflġíġ ĞĢĢf
flgíġġġġġġġġġġġġġġġġ
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ĝĥĥ ĬIJĹĹf flgìğġĥĥĝĥ ĞĢĢf flgíì
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ĝĥfl ĬIJĹĹf flgìflġîflìĥ ĞĢĢf
flgîflĥĥĥĥĥĥĥĥĥĥĥĥĥĥ ggg
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf ìġġĥ ĬIJĹĹf flgĝìğíflĝflğ ĞĢĢf
flgîġġġġġġġġġġġġġġġġ
NĦŃ ĹĦĻ IJÌ ŃĦĬÍÎĻĹ ÌIJŁĮĤd ĨĻĦĹĞĻĨIJĮf îġîì ĬIJĹĹf flgĝìğìîìĝĝ ĞĢĢf flgîġ
```

Loss descended by a decent amount this time, and accuracy raised significantly. Applying a fraction of random values actually lead to a result that we could almost call a solution. If you try 100,000 iterations, you will not progress much further:

```
ńńń
ggg
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮÂd ĨĻĦĹĞĻĨIJĮf ĝǵğfłħ ĬIJĽĽf fłgĝìğìîġğ ĞĢĢf
fłgîġġġġġġġġġġġġġġġ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮÂd ĨĻĦĹĞĻĨIJĮf ħġìfłǵ ĬIJĽĽf fłgĝìğìíğġğ ĞĢĢf
fłgîġġġġġġġġġġġġġġġ
```

Let’s try this with the previously-seen spiral dataset instead:

```
fi CĹĦĞĻĦ ÂĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçÂĞĻĞâĽĞİĴĬĦĽłĝfłfłd ĢĬĞĽĽĦĽłġã

ńńń
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮÂd ĨĻĦĹĞĻĨIJĮf fł ĬIJĽĽf ĝĝĝfłfłíħìì ĞĢĢf
fłgġġġġġġġġġġġġġġġ ggg
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮÂd ĨĻĦĹĞĻĨIJĮf ġĝ ĬIJĽĽf ĝgfłîíğĝħǵ ĞĢĢf
fłgġìġġġġġġġġġġġġġĥ ggg
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮÂd ĨĻĦĹĞĻĨIJĮf ħĥ ĬIJĽĽf ĝgfłîĥġĝħğ ĞĢĢf
fłgġíġġġġġġġġġġġġħ
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮÂd ĨĻĦĹĞĻĨIJĮf ħì ĬIJĽĽf ĝgfłîġîíî ĞĢĢf
fłgǵĝħħħħħħħħħħħħì ggg
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮÂd ĨĻĦĹĞĻĨIJĮf ĝğî ĬIJĽĽf ĝgfłíìǵĝğğ ĞĢĢf
fłgǵğġġġġġġġġġġġġġǵ ggg
NĦŃ ĽĦĻ IJÌ ŃĦĨÍÎĻĽ ÌIJŁĮÂd ĨĻĦĹĞĻĨIJĮf ĥǵĝĥ ĬIJĽĽf ĝgfłìîfłĥìĥ ĞĢĢf fłgġî
```

This training session ended with almost no progress. Loss decreased slightly and accuracy is barely above the initial value. Later, we’ll learn that the most probable reason for this is called a local minimum of loss. The data complexity is also not irrelevant here. It turns out hard problems are hard for a reason, and we need to approach this problem more intelligently.

Supplementary Material: https://nnfs.io/ch6

Chapter code, further resources, and errata for this chapter.

Chapter 7

Derivatives

Randomly changing and searching for optimal weights and biases did not prove fruitful for one main reason: the number of possible combinations of weights and biases is infinite, and we need something smarter than pure luck to achieve any success. Each weight and bias may also have different degrees of influence on the loss — this influence depends on the parameters themselves as well as on the current sample, which is an input to the first layer. These input values are then multiplied by the weights, so the input data affects the neuron's output and affects the impact that the weights make on the loss. The same principle applies to the biases and parameters in the next layers, taking the previous layer's outputs as inputs. This means that the impact on the output values depends on the parameters as well as the samples — which is why we are calculating the loss value per each sample separately. Finally, the function of *how* a weight or bias impacts the overall loss is not necessarily linear. In order to know *how* to adjust weights and biases, we first need to understand their impact on the loss.

One concept to note is that we refer to weights and biases and their impact on the loss function. The loss function doesn't contain weights or biases, though. The input to this function is the output of the model, and the weights and biases of the neurons influence this output. Thus, even though we calculate loss from the model's output, not weights/biases, these weights and biases

directly impact the loss.

In the coming chapters, we will describe exactly how this happens by explaining partial derivatives, gradients, gradient descent, and backpropagation. Basically, we'll calculate how much each singular weight and bias changes the loss value (how much of an impact it has on it) given a sample (as each sample produces a separate output, thus also a separate loss value), and how to change this weight or bias for the loss value to decrease. Remember — our goal here is to decrease loss, and we'll do this by using gradient descent. Gradient, on the other hand, is a result of the calculation of the partial derivatives, and we'll backpropagate it using the chain rule to update all of the weights and biases. Don't worry if that doesn't make much sense yet; we'll explain all of these terms and how to perform these actions in this and the coming chapters.

To understand partial derivatives, we need to start with derivatives, which are a special case of partial derivatives — they are calculated from functions taking single parameters.

The Impact of a Parameter on the Output

Let's start with a simple function and discover what is meant by "impact."
A very simple function $y=2x$, which takes x as an input:

```
ÂHÌ ÌâŇãf
    ĹHĻŁĹĮ ğǻŇ
```

Now let's create some code around this to visualize the data — we'll import NumPy and Matplotlib, create an array of 5 input values from 0 to 4, calculate the function output for each of these input values, and plot the result as lines between consecutive points. These points' coordinates are inputs as x and function outputs as y:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ÂHÌ ÌâŇãf
    ĹHĻŁĹĮ ğǻŇ
```

```
Ň ł ĮĴgĞĹĹĞÑâĹĞĮÍĦâĥãã
Ñ ł ÌâŇã


ĴĹĨĮĻâŇã
ĴĹĨĮĻâŇã


ńńń
äfl ĝ ğ ġ ǵā
äfl ğ ǵ ħ íā


ĴĬĻgĴĬIJĻâŇd Ñã
ĴĬĻgĹÎIJŃâã
```

Fig 7.01: Linear function y=2x graphed

The Slope

This looks like an output of the *f(x) = 2x* function, which is a line. How might you define the *impact* that *x* will have on *y*? Some will say, *"y* is double *x*" Another way to describe the *impact* of a linear function such as this comes from algebra: the **slope**. "Rise over run" might be a phrase you recall from school. The slope of a line is:

$$\frac{Change\ in\ y}{Change\ in\ x} = \frac{\Delta y}{\Delta x}$$

It is change in *y* divided by change in *x,* or, in math — *delta y* divided by *delta x*. What's the slope of *f(x) = 2x* then?
To calculate the slope, first we have to take any two points lying on the function's graph and subtract them to calculate the change. Subtracting the points means to subtract their x and y dimensions respectively. Division of the change in y by the change in x returns the slope:

$$\begin{array}{c} x\ \ y \\ \downarrow\downarrow \end{array}$$
$$\begin{cases} p_1 = [0, 0] \\ p_2 = [1, 2] \end{cases}$$

$$\Delta x = p_{2x} - p_{1x} = 1 - 0 = 1$$
$$\Delta y = p_{2y} - p_{1y} = 2 - 0 = 2$$

$$slope = \frac{\Delta y}{\Delta x} = \frac{2}{1} = 2$$

Continuing the code, we keep all values of *x* in a single-dimensional NumPy array, Ňäflā and all results in a single-dimensional array, Ñäflā for the first point, then Ňäĝā and Ñäĝā for the second one. Now we can calculate the slope between them:

```
ĴĹĨĮĻââÑäĝāhÑäflāã w âŇäĝāhŇäflāãã
```

```
ńńń
ĝgfl
```

It is not surprising that the slope of this line is 2. We could say the measure of the impact that *x* has on *y* is 2. We can calculate the slope in the same way for any linear function, including linear functions that aren't as obvious.
What about a nonlinear function like $f(x)=2x^2$?

```
ÎĦÌ ÌâŇãf
    ĹĦĻŁĹĮ ĝǻŇǻǻĝ
```

This function creates a graph that does not form a straight line:

Fig 7.02: Approximation of the parabolic function $y=2x^2$ graphed

Can we measure the slope of this curve? Depending on which 2 points we choose to use, we will measure varying slopes:

```
Ñ ł ÌâÑã  ﬁ CĞĬĢŁĬĞĻĦ ÌŁĮĢĻĨIJĮ IJŁĻĴŁĻĹ ÌIJĹ ĮĦŃ ÌŁĮĢĻĨIJĮ

ĴĹĨĮĻâÑã
ĴĹĨĮĻâÑã


ńńń
äﬂ ĝ ğ ġ ǵā
ä ﬂ  ğ  í ĝí ġğā
```

Now for the first pair of points:

```
ĴĹĨĮĻââÑäĝāhÑäﬂāã w âÑäĝāhÑäﬂāãã


ńńń
ğ
```

And for another one:

```
ĴĹĨĮĻââÑäġāhÑäğāã w âÑäġāhÑäğāãã

ńńń
ĝﬂ
```

Fig 7.03: Approximation of the parabolic function's example tangents

Anim 7.03: https://nnfs.io/bro

How might we measure the impact that x has on y in this nonlinear function? Calculus proposes that we measure the slope of the **tangent line** at *x* (for a specific input value to the function), giving us the **instantaneous slope** (slope at this point)**,** which is the **derivative**. The **tangent line** is created by drawing a line between two points that are "infinitely close" on a curve, but this curve has to be differentiable at the derivation point. This means that it has to be continuous and smooth (we cannot calculate the slope at something that we could describe as a "sharp corner," since it contains an infinite number of slopes). Then, because this is a curve, there is no single slope. Slope depends on where we measure it. To give an immediate example, we can approximate a derivative of the function at *x* by using this point and another one also taken at x, but with a very small delta added to it, such as *0.0001*. This number is a common choice as it does not introduce too large an error (when estimating the derivative) or cause the whole expression to be numerically unstable (Δx might round to 0 due to floating-point number resolution). This lets us perform the same calculation for the slope as before, but on two points that are very close to each other, resulting in a good approximation of a slope at x:

```
ĴğçĥĥĭĻĞ ł flgflflflĝ

Ňĝ ł ĝ
Ňğ ł Ňĝ к ĴğçĥĥĭĻĞ  fi ĞĥĤ ĤĥĭĻĞ

Ňĝ ł ÌâŇĝã  fi ĹĥĽłĭĻ ĞĻ ĻÎĥ ĤĥĹĨĿĞĻĨIJĮ ĴIJĨĮĻ
Ňğ ł ÌâŇğã  fi ĹĥĽłĭĻ ĞĻ ĻÎĥ IJĻÎĥĹd ĢĭIJĽĥ ĴIJĨĮĻ

ĞĴĴĹIJŇĨİĞĻĥçĤĥĹĨĿĞĻĨĿĥ ł âŇğhŇĝãwâŇğhŇĝã
ĴĹĨĮĻâĞĴĴĹIJŇĨİĞĻĥçĤĥĹĨĿĞĻĨĿĥã


ńńń
ĝgflflflĝîîîîîîîîìíĝĥ
```

As we will soon learn, the derivative of $2x^2$ at $x=1$ should be exactly *4*. The difference we see (~4.0002) comes from the method used to compute the tangent. We chose a delta small enough to

approximate the derivative as accurately as possible but large enough to prevent a rounding error. To elaborate, an infinitely small delta value will approximate an accurate derivative; however, the delta value needs to be numerically stable, meaning, our delta can not surpass the limitations of Python's floating-point precision (can't be too small as it might be rounded to *0* and, as we know, dividing by *0* is "illegal"). Our solution is, therefore, restricted between estimating the derivative and remaining numerically stable, thus introducing this small but visible error.

The Numerical Derivative

This method of calculating the derivative is called **numerical differentiation** — calculating the slope of the tangent line using two *infinitely* close points, or as with the code solution — calculating the slope of a tangent line made from two points that were "sufficiently close." We can visualize why we perform this on two close points with the following:

Fig 7.04: Why we want to use 2 points that are sufficiently close — large delta inaccuracy.

Fig 7.05: Why we want to use 2 points that are sufficiently close — very small delta accuracy.

Anim 7.04-7.05: https://nnfs.io/cat

We can see that the closer these two points are to each other, the more correct the tangent line appears to be.

Continuing with **numerical differentiation**, let us visualize the tangent lines and how they change depending on where we calculate them. To begin, we'll make the graph of this function more granular using Numpy's `ĞĹĞĮÍĦâã`, allowing us to plot with smaller steps. The `ĮĴgĞĹĞĮÍĦâã` function takes in *start, stop,* and *step* parameters, allowing us to take fractions of a step, such as *0.001* at a time:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ


ĤĦÌ ÌâŇãf
    ĹĦĻŁĹĮ ğǻÑǻǻğ


ﬠ ĮĴgĞĹĞĮÍĦâĽĻĞĹĻd ĽĻIJĴd ĽĻĦĴã ĻIJ ÍĨĿĦ ŁĽ ĽİIJIJĻÎĦĹ ĬĨĮĦ
Ñ ł ĮĴgĞĹĞĮÍĦâﬠd ĥd ﬠgﬠﬠĝã
Ñ ł ÌâÑã
```

```
ĴĬĻgĴĬIJĻâŇd Ñã

ĴĬĻgĽÎIJŃâã
```

Fig 7.06: Matplotlib output that you should see from graphing $y=2x^2$.

To draw these tangent lines, we will derive the function for the tangent line at a point and plot the tangent on the graph at this point. The function for a straight line is $y = mx+b$. Where m is the slope or the *approximate_derivative* that we've already calculated. And x is the input which leaves b, or the y-intercept, for us to calculate. The slope remains unchanged, but currently, you can "move" the line up or down using the y-intercept. We already know x and m, but b is still unknown. Let's assume $m=1$ for the purpose of the figure and see what exactly it means:

Fig 7.07: Various biases graphed where slope = 1.

Anim 7.07: https://nnfs.io/but

To calculate *b,* the formula is $b = y - mx$:

$$y = mx + b$$

$$y - mx = b$$

$$b = y - mx$$

So far we've used two points — the point that we want to calculate the derivative at and the "close enough" to it point to calculate the approximation of the derivative. Now, given the above equation for *b*, the approximation of the derivative and the same "close enough" point (its *x* and *y* coordinates to be specific), we can substitute them in the equation and get the y-intercept for the tangent line at the derivation point. Using code:

```
Ġ ł Ñğ h ĞĴĴĹIJŇĨİĞĻĦçĤĦĹĨĿĞĻĨĿĦáŇğ
```

Putting everything together:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ


ĤĦÌ ÌâŇãf
    ĹĦĻŁĹĮ ğáÑááğ


fi ĮĴgĞĹĞĮÍĦâĽĻĞĹĻd ĽĻIJĴd ĽĻĦĴã ĻIJ ÍĨĿĦ ŁĽ ĽİIJIJĻÎĦĹ ĬĨĮĦ
Ň ł ĮĴgĞĹĞĮÍĦâfld ĥd flgflflĝã
Ñ ł ÌâÑã

ĴĬĻgĴĬIJĻâÑd Ñã
```

```
fi TÎH ĴIJÎĮĻ ĞĮĤ ĻÎH ðĢĬIJĽH HĮIJŁÍÎð ĴIJÎĮĻ
Ĵğç̂ĤĬĻĞ ł flgflflfl̂g
Ňĝ ł ğ
Ňğ ł Ňĝκ̂Ĵğç̂ĤĬĻĞ

Ňĝ ł ÌâŇĝã
Ňğ ł ÌâŇğã

ĴĹÎĮĻââŇĝd Ňĝãd âŇğd Ňğãã

fi DHĹÎĽĞĻÎĽH ĞĴĴĹIJŇÎİĞĻÎIJĮ ĞĮĤ ŇhÎĮĻHĹĢHĴĻ ÌIJĹ ĻÎH ĻĞĮÍHĮĻ ĬÎĮH
ĞĴĴĹIJŇÎİĞĻHçĤHĹÎĽĞĻÎĽH ł âŇğhŇĝãwâŇğhŇĝã
ġ ł Ňğ h ĞĴĴĹIJŇÎİĞĻHçĤHĹÎĽĞĻÎĽHåŇğ


fi WH ĴŁĻ ĻÎH ĻĞĮÍHĮĻ ĬÎĮH ĢĞĬĢŁĬĞĻÎIJĮ ÎĮĻIJ Ğ ÌŁĮĢĻÎIJĮ ĹIJ ŃH ĢĞĮ ĢĞĬĬ
fi ÎĻ İŁĬĻÎĴĬH ĻÎİHĹ ÌIJĹ ĤÎÌÌHĹHĮĻ ĽĞĬŁHĹ IJÌ Ň
fi ĞĴĴĹIJŇÎİĞĻHçĤHĹÎĽĞĻÎĽH ĞĮĤ ġ ĞĹH ĢIJĮĹĻĞĮĻ ÌIJĹ ÍÎĽHĮ ÌŁĮĢĻÎIJĮ
fi ĻÎŁĹ ĢĞĬĢŁĬĞĻHĤ IJĮĢH ĞġIJĽH ĻÎÎĹ ÌŁĮĢĻÎIJĮ
ĤHÌ ĻĞĮÍHĮĻçĬÎĮHâŇãf
    ĹHĻŁĹĮ ĞĴĴĹIJŇÎİĞĻHçĤHĹÎĽĞĻÎĽHåŇ κ ġ


fi ĴĬIJĻĻÎĮÍ ĻÎH ĻĞĮÍHĮĻ ĬÎĮH
fi κwh flgî ĻIJ ĤĹĞŃ ĻÎH ĻĞĮÍHĮĻ ĬÎĮH IJĮ IJŁĹ ÍĹĞĴÎ
fi ĻÎHĮ ŃH ĢĞĬĢŁĬĞĻH ĻÎH Ñ ÌIJĹ ÍÎĽHĮ Ň ŁĹÎĮÍ ĻÎH ĻĞĮÍHĮĻ ĬÎĮH ÌŁĮĢĻÎIJĮ
fi MĞĻĴĬIJĻĬÎġ ŃÎĬĬ ĤĹĞŃ Ğ ĬÎĮH ÌIJĹ ŁĹ ĻÎĹIJŁÍÎ ĻÎHĹH ĴIJÎĮĻĹ
ĻIJçĴĬIJĻ ł äŇĝhflgîd Ňĝd Ňĝκflgîã
ĴĬĻgĴĬIJĻâĻIJçĴĬIJĻd äĻĞĮÍHĮĻçĬÎĮHâÎã ÌIJĹ Î ÎĮ ĻIJçĴĬIJĻãã

ĴĹÎĮĻâďAĴĴĹIJŇÎİĞĻH ĤHĹÎĽĞĻÎĽH ÌIJĹ ÌâŇãďd
      ÌďŃÎHĹH Ň ł ăŇĝå ÎĽ ăĞĴĴĹIJŇÎİĞĻHçĤHĹÎĽĞĻÎĽHåďã

ĴĬĻgĽÎIJŃâã


ńńń
âğd íã âğgflflflĝd ígflflflíflflflğflflflflflğã
AĴĴĹIJŇÎİĞĻH ĤHĹÎĽĞĻÎĽH ÌIJĹ ÌâŇã ŃÎHĹH Ň ł ğ ÎĽ ígflflflĝîîîîîîíìíĥ
```

Fig 7.08: Graphed approximate derivative for f(x) where x=2

The orange line is the approximate tangent line at *x=2* for the function $f(x) = 2x^2$. Why do we care about this? You will soon find that we care only about the *slope* of this tangent line but both visualizing and understanding the **tangent line** are very important. We care about the slope of the tangent line because it informs us about the *impact* that *x* has on this function at a particular point, referred to as the **instantaneous rate of change**. We will use this concept to determine the effect of a specific weight or bias on the overall loss function given a sample. For now, with different values for *x*, we can observe resulting impacts on the function. We can continue the previous code to see the tangent line for various inputs (*x)* - we put a part of the code in a loop over example *x* values and plot multiple tangent lines:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ


ÂĦÌ ÌâŇãf
    ĹĦĻŁĹĮ ğǻÑǻǻğ


ﬁ ĮĴgĞĹĞĮÍĦâĽĻĞĹĻd ĽĻIJĴd ĽĻĦĴã ĻIJ ÍĨĿĦ ŁĽ Ğ ĽİIJIJĻÎĦĹ ĢŁĹĿĦ
Ñ ł ĮĴgĞĹĹĞÑâĮĴgĞĹĞĮÍĦâﬂdĥdﬂgﬂﬂĝãã
Ñ ł ÌâÑã

ĴĬĻgĴĬIJĻâÑd Ñã

ĢIJĬIJĹĽ ł äďĪďddďÍďddďĹďddďĠďddďĢďã


ÂĦÌ ĞĴĴĹIJÑĨİĞĻĦçĻĞĮÍĦĮĻçĬĨĮĦâŇd ĞĴĴĹIJÑĨİĞĻĦçÂĦĹĨĿĞĻĨĿĦãf
    ĹĦĻŁĹĮ âĞĴĴĹIJÑĨİĞĻĦçÂĦĹĨĿĞĻĨĿĦǻÑã κ Ġ
```

```
ÌIJĹ Ĩ ĨĮ ĹĞĮÍĦâĥãf
    Ĵğç̂ĦĬĻĞ ł flgflflflĝ
    Ñĝ ł Ĩ
    Ñğ ł Ñĝκĵğç̂ĦĬĻĞ

    Ñĝ ł ÌâÑĝã
    Ñğ ł ÌâÑğã

    ĴĹĨĮĻââÑĝd Ñĝãd âÑğd Ñğãã
    ĞĴĴĹIJÑĨİĞĻĦçĤĦĹĨĿĞĻĨĿĦ ł âÑğhÑĝãwâÑğhÑĝã
    Ġ ł ÑğhâĞĴĴĹ̇IJÑĨİĞĻĦçĤĦĹĨĿĞĻĨĿĦáÑğã

    ĻIJçĴĬIJĻ ł äÑĝhflgîd Ñĝd Ñĝκflgîã

    ĴĬĻgĽĢĞĻĻĦĹâÑĝd Ñĝd Ģ̣łĢIJĬIJĹĽäĨãã
    ĴĬĻgĴĬIJĻâäĴIJĨĮĻ ÌIJĹ ĴIJĨĮĻ ĨĮ ĻIJçĴĬIJĻãd
             äĞĴĴĹIJÑĨİĞĻĦçĻĞĮÍĦĮĻçĬĨĮĦâĴIJĨĮĻd ĞĴĴĹIJÑĨİĞĻĦçĤĦĹĨĿĞĻĨĿĦã
                 ÌIJĹ ĴIJĨĮĻ ĨĮ ĻIJçĴĬIJĻãd
             Ģ̣łĢIJĬIJĹĽäĨãã

    ĴĹĨĮĻâďAĴĴĹIJÑĨİĞĻĦ ĤĦĹĨĿĞĻĨĿĦ ÌIJĹ ÌâÑãďd
          ÌďÑÎĦĹĦ Ñ ł äÑĝå ĨĽ äĞĴĴĹIJÑĨİĞĻĦçĤĦĹĨĿĞĻĨĿĦåďã

ĴĬĻgĽÎIJÑâã


ńńń
âfld flã âflgflflflĝd ğĦhflíã
AĴĴĹIJÑĨİĞĻĦ ĤĦĹĨĿĞĻĨĿĦ ÌIJĹ ÌâÑã ÑÎĦĹĦ Ñ ł fl ĨĽ flgflflflĝîîîîîîîîîîîîîîî
âĝd ğã âĝgflflflĝd ğgflflflgflflflğã
AĴĴĹIJÑĨİĞĻĦ ĤĦĹĨĿĞĻĨĿĦ ÌIJĹ ÌâÑã ÑÎĦĹĦ Ñ ł ĝ ĨĽ ĝgflflflĝîîîîîîîîìígĥ
âğd íã âğgflflflĝd ígflflflíflflflğflflflflflğã
AĴĴĹIJÑĨİĞĻĦ ĤĦĹĨĿĞĻĨĿĦ ÌIJĹ ÌâÑã ÑÎĦĹĦ Ñ ł ğ ĨĽ ígflflflĝîîîîîîîìíĥ
âġd ĝíã âġgflflflĝd ĝígflflĝğflflflğflflflflflğã
AĴĴĹIJÑĨİĞĻĦ ĤĦĹĨĿĞĻĨĿĦ ÌIJĹ ÌâÑã ÑÎĦĹĦ Ñ ł ġ ĨĽ ĝğgflflflĝîîîîîîîìíĥ
âǵd ġğã âǵgflflflĝd ġğgflflĝħflflflğã
AĴĴĹIJÑĨİĞĻĦ ĤĦĹĨĿĞĻĨĿĦ ÌIJĹ ÌâÑã ÑÎĦĹĦ Ñ ł ǵ ĨĽ ĝħgflflflğflflflflflflĝħĥǵí
```

Fig 7.09: Derivative calculated at various points.

For this simple function, $f(x) = 2x^2$, we didn't pay a high penalty by approximating the derivative (i.e., the slope of the tangent line) like this, and received a value that was close enough for our needs.

The problem is that the *actual* function employed in our neural network is not so simple. The loss function contains all of the layers, weights, and biases — it's an absolutely massive function operating in multiple dimensions! Calculating derivatives using **numerical differentiation** requires multiple forward passes for a single parameter update (we'll talk about parameter updates in chapter 10). We need to perform the forward pass as a reference, then update a single parameter by the delta value and perform the forward pass through our model again to see the change of the loss value. Next, we need to calculate the **derivative** and revert the parameter change that we made for this calculation. We have to repeat this for every weight and bias and for every sample, which will be very time-consuming. We can also think of this method as brute-forcing the derivative calculations. To reiterate, as we quickly covered many terms, the **derivative** is the **slope** of the **tangent line** for a function that takes a single parameter as an input. We'll use this ability to calculate the slopes of the loss function at each of the weight and bias points — this brings us to the multivariate function, which is a function that takes multiple parameters and is a topic for the next chapter — the partial derivative.

The Analytical Derivative

Now that we have a better idea of what a derivative *is*, how to calculate the numerical (also called universal) derivative, and why it's not a good approach for us, we can move on to the **Analytical Derivative**, the actual solution to the derivative that we'll implement in our code.

In mathematics, there are two general ways to solve problems: **numerical** and **analytical** methods. Numerical solution methods involve coming up with a number to find a solution, like the above approach with ĞĴĴĹIJŇĨİĞĻĦçĤĦĹĨĿĞĻĨĿĦ. The numerical solution is also an approximation. On the other hand, the analytical method offers the exact and much quicker, in terms of calculation, solution. However, identifying the analytical solution for the derivative of a given function, as we'll quickly learn, will vary in complexity, whereas the numerical approach never gets more complicated — it's always calling the method twice with two inputs to calculate the approximate derivative at a point. Some analytical solutions are quite obvious, some can be calculated with simple rules, and some complex functions can be broken down into simpler parts and calculated using the so-called **chain rule**. We can leverage already-proven derivative solutions for certain functions, and others — like our loss function — can be solved with combinations of the above.

To compute the derivative of functions using the analytical method, we can split them into simple, elemental functions, finding the derivatives of those and then applying the **chain rule**, which we will explain soon, to get the full derivative. To start building an intuition, let's start with simple functions and their respective derivatives.

The derivative of a simple constant function:

$$f(x) = 1 \quad \rightarrow \quad \frac{d}{dx} f(x) = \frac{d}{dx} 1 = 0$$

$$f(x) = 1$$

$$f'(x) = \frac{d}{dx} 1$$

$$f'(x) = 0$$

https://nnfs.io

Fig 7.10: Derivative of a constant function — calculation steps.

Anim 7.10: https://nnfs.io/cow

When calculating the derivative of a function, recall that the derivative can be interpreted as a slope. In this example, the result of this function is a horizontal line as the output value for any x is 1:

By looking at it, it becomes evident that the derivative equals 0 since there's no change from one value of x to any other value of x (i.e., there's no slope).

So far, we are calculating derivatives of the functions by taking a single parameter, *x* in our case, in each example. This changes with partial derivatives since they take functions with multiple parameters, and we'll be calculating the derivative with respect to only one of them at a time. For now, with derivatives, it's always with respect to a single parameter. To denote the derivative, we can use prime notation, where, for the function $f(x)$, we add a prime (') like $f'(x)$. For our example, $f(x) = 1$, the derivative $f'(x) = 0$. Another notation we can use is called the Leibniz's notation — the dependence on the prime notation and multiple ways of writing the derivative with the Leibniz's notation is as follows:

$$f'(x) = \frac{d}{dx}f(x) = \frac{df}{dx}(x) = \frac{df(x)}{dx}$$

Each of these notations has the same meaning — the derivative of a function (with respect to *x*).

In the following examples, we use both notations, since sometimes it's convenient to use one notation or another. We can also use both of them in a single equation.

In summary: the derivative of a constant function equals *0*:

$$f(x) = 1 \quad \rightarrow \quad f'(x) = 0$$

The derivative of a linear function:

$$f(x) = x \quad \rightarrow \quad \frac{d}{dx}f(x) = \frac{d}{dx}x = \frac{d}{dx}x^1 = 1 \cdot x^{1-1} = 1 \cdot x^0 = 1 \cdot 1 = 1$$

Fig 7.11: Derivative of a linear function — calculation steps.

Anim 7.11: https://nnfs.io/tob

In this case, the derivative is 1, and the intuition behind this is that for every change of x, y changes by the same amount, so y changes one times the x.

The derivative of the linear function equals *1* (but not in every case, which we'll explain next):

$$f(x) = x \quad \rightarrow \quad f'(x) = 1$$

What if we try 2x, which is also a linear function?

$$f(x) = 2x \quad \rightarrow \quad \frac{d}{dx}f(x) = \frac{d}{dx}2x = 2 \cdot \frac{d}{dx}x = 2 \cdot 1x^{1-1} = 2 \cdot 1x^0 = 2 \cdot 1 = 2$$

$$f(x) = 2x$$
$$f'(x) = \frac{d}{dx}2x$$
$$f'(x) = 2 \cdot \frac{d}{dx}x^1$$
$$f'(x) = 2 \cdot x^{1-1}$$
$$f'(x) = 2 \cdot 1x^0$$
$$f'(x) = 2 \cdot 1 \cdot 1$$
$$f'(x) = 2$$

https://nnfs.io

Fig 7.12: Derivative of another linear function — calculation steps.

Anim 7.12: https://nnfs.io/pop

When calculating the derivative, we can take any constant that function is multiplied by and move it outside of the derivative — in this case it's 2 multiplied by the derivative of x. Since we already determined that the derivative of *f(x) = x* was *1*, we now multiply it by *2* to give us the result.

The derivative of a linear function equals the slope, *m* In this case *m = 2*:

$$f(x) = 2x \quad \rightarrow \quad f'(x) = 2$$

If you associate this with numerical differentiation, you're absolutely right — we already concluded that the derivative of a linear function equals its slope:

$$f(x) = mx \quad \rightarrow \quad f'(x) = m$$

m, in this case, is a constant, no different than the value *2,* as it's not a parameter — every non-parameter to the function can't change its value; thus, we consider it to be a constant. We have just found a simpler way to calculate the derivative of a linear function and also generalized it for the equations of different slopes, *m*. It's also an exact derivative, not an approximation, as with the numerical differentiation.

What happens when we introduce exponents to the function?

$$f(x) = 3x^2 \quad \rightarrow \quad \frac{d}{dx}f(x) = \frac{d}{dx}3x^2 = 3 \cdot \frac{d}{dx}x^2 = 3 \cdot 2x^{2-1} = 3 \cdot 2x^1 = 6x$$

$$f(x) = 3x^2$$
$$f'(x) = \frac{d}{dx}3x^2$$
$$f'(x) = 3 \cdot x^{2-1}$$
$$f'(x) = 3 \cdot 2x^1$$
$$f'(x) = 6x$$

https://nnfs.io

Fig 7.13: Derivative of quadratic function — calculation steps.

Anim 7.13: https://nnfs.io/rok

First, we are applying the rule of a constant — we can move the coefficient (the value that multiplies the other value) outside of the derivative. The rule for handling exponents is as follows: take the exponent, in this case a *2*, and use it as a coefficient for the derived value, then, subtract 1 from the exponent, as seen here: *2 - 1 = 1*.

If $f(x) = 3x^2$ then $f'(x) = 3 \cdot 2x^1$ or simply $6x$. This means the slope of the tangent line, at any point, x, for this quadratic function, will be $6x$. As discussed with the numerical solution of the quadratic function differentiation, the derivative of a quadratic function depends on the x and in this case it equals $6x$:

$$f(x) = 3x^2 \quad \rightarrow \quad f'(x) = 6x$$

A commonly used operator in functions is addition, how do we calculate the derivative in this case?

$$\begin{aligned} f(x) = 3x^2 + 5x \quad \rightarrow \quad \frac{d}{dx}f(x) &= \frac{d}{dx}[3x^2 + 5x] = \\ &= \frac{d}{dx}3x^2 + \frac{d}{dx}5x^1 = \\ &= 3 \cdot \frac{d}{dx}x^2 + 5 \cdot \frac{d}{dx}x^1 = \\ &= 3 \cdot 2x^{2-1} + 5 \cdot 1x^{1-1} = \\ &= 3 \cdot 2x^1 + 5 \cdot x^0 = \\ &= 6x + 5 \end{aligned}$$

$$f(x) = 3x^2 + 5x$$
$$f'(x) = \frac{d}{dx}[3x^2 + 5x]$$
$$f'(x) = \frac{d}{dx}3x^2 + \frac{d}{dx}5x$$
$$f'(x) = 3 \cdot x^{2-1} + 5 \cdot x^{1-1}$$
$$f'(x) = 3 \cdot 2x^1 + 5 \cdot 1x^0$$
$$f'(x) = 6x + 5$$

https://nnfs.io

Fig 7.14: Derivative of quadratic function with addition — calculation steps.

Anim 7.14: https://nnfs.io/mob

The derivative of a sum operation is the sum of derivatives, so we can split the derivative of a more complex sum operation into a sum of the derivatives of each term of the equation and solve the rest of the derivative using methods we already know.

The derivative of a sum of functions equals their derivatives:

$$\frac{d}{dx}[f(x) + g(x)] = \frac{d}{dx}f(x) + \frac{d}{dx}g(x) = f'(x) + g'(x)$$

In this case, we've shown the rule using both notations.

Let's try a couple more examples:

$$f(x) = 5x^5 + 4x^3 - 5 \quad \rightarrow \quad \frac{d}{dx}f(x) = \frac{d}{dx}[5x^5 + 4x^3 - 5] =$$

$$= \frac{d}{dx}5x^5 + \frac{d}{dx}4x^3 - \frac{d}{dx}5 =$$

$$= 5 \cdot \frac{d}{dx}x^5 + 4 \cdot \frac{d}{dx}x^3 - \frac{d}{dx}5 =$$

$$= 5 \cdot 5x^{5-1} + 4 \cdot 3x^{3-1} - 0 =$$

$$= 5 \cdot 5x^4 + 4 \cdot 3x^2 =$$

$$= 25x^4 + 12x^2$$

$$f(x) = 5x^5 + 4x^3 - 5$$

$$f'(x) = \frac{d}{dx}[5x^5 + 4x^3 - 5]$$

$$f'(x) = \frac{d}{dx}5x^5 + \frac{d}{dx}4x^3 - \frac{d}{dx}5$$

$$f'(x) = 5 \cdot x^{5-1} + 4 \cdot x^{3-1} - 0$$

$$f'(x) = 5 \cdot 5x^4 + 4 \cdot 3x^2$$

$$f'(x) = 25x^4 + 12x^2$$

https://nnfs.io

Fig 7.15: Analytical derivative of multi-dimensional function example — calculation steps.

Anim 7.15: https://nnfs.io/tom

The derivative of a constant 5 equals 0, as we already discussed at the beginning of this chapter. We also have to apply the other rules that we've learned so far to perform this calculation.

$$f(x) = x^3 + 2x^2 - 5x + 7 \quad \rightarrow \quad \frac{d}{dx}f(x) = \frac{d}{dx}[x^3 + 2x^2 - 5x + 7] =$$

$$= \frac{d}{dx}x^3 + \frac{d}{dx}2x^2 - \frac{d}{dx}5x + \frac{d}{dx}7 =$$

$$= \frac{d}{dx}x^3 + 2 \cdot \frac{d}{dx}x^2 - 5 \cdot \frac{d}{dx}x + \frac{d}{dx}7 =$$

$$= 3x^{3-1} + 2 \cdot 2x^{2-1} - 5 \cdot 1x^{1-1} + 0 =$$

$$= 3x^2 + 2 \cdot 2x^1 - 5 \cdot 1 + 0 =$$

$$= 3x^2 + 4x - 5$$

$$f(x) = x^3 + 2x^2 - 5x + 7$$

$$f'(x) = \frac{d}{dx}[x^3 + 2x^2 - 5x + 7]$$

$$f'(x) = \frac{d}{dx}x^3 + \frac{d}{dx}2x^2 - \frac{d}{dx}5x + \frac{d}{dx}7$$

$$f'(x) = x^{3-1} + 2 \cdot x^{2-1} - 5 \cdot x^{1-1} + 0$$

$$f'(x) = 3x^2 + 2 \cdot 2x^1 - 5 \cdot 1x^0$$

$$f'(x) = 3x^2 + 4x - 5$$

https://nnfs.io

Fig 7.16: Analytical derivative of another multi-dimensional function example — calculation steps.

Anim 7.16: https://nnfs.io/sun

This looks relatively straight-forward so far, but, with neural networks, we'll work with functions that take multiple parameters as inputs, so we're going to calculate the partial derivatives as well.

Summary

Let's summarize some of the solutions and rules that we have learned in this chapter.

Solutions:

The derivative of a constant equals 0 (m is a constant in this case, as it's not a parameter that we are deriving with respect to, which is *x* in this example):

$$\frac{d}{dx}1 = 0$$

$$\frac{d}{dx}m = 0$$

The derivative of *x* equals 1:

$$\frac{d}{dx}x = 1$$

The derivative of a linear function equals its slope:

$$\frac{d}{dx}mx + b = m$$

Rules:

The derivative of a constant multiple of the function equals the constant multiple of the function's

derivative:

$$\frac{d}{dx}[k \cdot f(x)] = k \cdot \frac{d}{dx}f(x)$$

The derivative of a sum of functions equals the sum of their derivatives:

$$\frac{d}{dx}[f(x) + g(x)] = \frac{d}{dx}f(x) + \frac{d}{dx}g(x) = f'(x) + g'(x)$$

The same concept applies to subtraction:

$$\frac{d}{dx}[f(x) - g(x)] = \frac{d}{dx}f(x) - \frac{d}{dx}g(x) = f'(x) - g'(x)$$

The derivative of an exponentiation:

$$\frac{d}{dx}x^n = n \cdot x^{n-1}$$

We used the value x instead of the whole function f(x) here since the derivative of an entire function is calculated a bit differently. We'll explain this concept along with the chain rule in the next chapter.

Since we've already learned what derivatives are and how to calculate them analytically, which we'll later implement in code, we can go a step further and cover partial derivatives in the next chapter.

Supplementary Material: https://nnfs.io/ch7
Chapter code, further resources, and errata for this chapter.

Chapter 8

Gradients, Partial Derivatives, and the Chain Rule

Two of the last pieces of the puzzle, before we continue coding our neural network, are the related concepts of **gradients** and **partial derivatives**. The derivatives that we've solved so far have been cases where there is only one independent variable in the function — that is, the result depended solely on, in our case, x. However, our neural network consists, for example, of neurons, which have multiple inputs. Each input gets multiplied by the corresponding weight (a function of 2 parameters), and they get summed with the bias (a function of as many parameters as there are inputs, plus one for a bias). As we'll explain soon in detail, to learn the impact of all of the inputs, weights, and biases to the neuron output and at the end of the loss function, we need to calculate the derivative of each operation performed during the forward pass in the neuron and the whole model. To do that and get answers, we'll need to use the **chain rule**, which we'll explain soon in this chapter.

The Partial Derivative

The **partial derivative** measures how much impact a single input has on a function's output. The method for calculating a partial derivative is the same as for derivatives explained in the previous chapter; we simply have to repeat this process for each of the independent inputs.

Each of the function's inputs has some impact on this function's output, even if the impact is 0. We need to know these impacts; this means that we have to calculate the derivative with respect to each input separately to learn about each of them. That's why we call these partial derivatives with respect to given input — we are calculating a partial of the derivative, related to a singular input. The partial derivative is a single equation, and the full multivariate function's derivative consists of a set of equations called the **gradient**. In other words, the **gradient** is a vector of the size of inputs containing partial derivative solutions with respect to each of the inputs. We'll get back to gradients shortly.

To denote the partial derivative, we'll be using Euler's notation. It's very similar to Leibniz's notation, as we only need to replace the differential operator *d* with ∂. While the *d* operator might be used to denote the differentiation of a multivariate function, its meaning is a bit different — it can mean the rate of the function's change in relation to the given input, but when other inputs might change as well, and it is used mostly in physics. We are interested in the partial derivatives, a situation where we try to find the impact of the given input to the output while treating all of the other inputs as constants. We are interested in the impact of singular inputs since our goal, in the model, is to update parameters. The ∂ operator means explicitly that — the partial derivative:

$$f(x, y, z) \quad \rightarrow \quad \frac{\partial}{\partial x} f(x, y, z), \frac{\partial}{\partial y} f(x, y, z), \frac{\partial}{\partial z} f(x, y, z)$$

The Partial Derivative of a Sum

Calculating the partial derivative with respect to a given input means to calculate it like the regular derivative of one input, just while treating other inputs as constants. For example:

$$f(x,y) = x + y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = \frac{\partial}{\partial x}[x+y] = \frac{\partial}{\partial x}x + \frac{\partial}{\partial x}y = 1 + 0 = 1$$
$$\frac{\partial}{\partial y} f(x,y) = \frac{\partial}{\partial y}[x+y] = \frac{\partial}{\partial y}x + \frac{\partial}{\partial y}y = 0 + 1 = 1$$

First, we applied the sum rule — the derivative of a sum is the sum of derivatives. Then, we already know that the derivative of *x* with respect to *x* equals *1*. The new thing is the derivative of *y* with respect to *x*. As we mentioned, *y* is treated as a constant, as it does not change when we are deriving with respect to *x*, and the derivative of a constant equals *0*. In the second case, we derived with respect to *y*, thus treating *x* as constant. Put another way, regardless of the value of y in this example, the slope of *x* does not depend on *y*. This will not always be the case, though, as we will soon see.

Let's try another example:

$$f(x,y) = 2x + 3y^2 \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = \frac{\partial}{\partial x}[2x + 3y^2] = \frac{\partial}{\partial x}2x + \frac{\partial}{\partial x}3y^2 =$$
$$= 2 \cdot \frac{\partial}{\partial x}x + 3 \cdot \frac{\partial}{\partial x}y^2 = 2 \cdot 1 + 3 \cdot 0 = 2$$
$$\frac{\partial}{\partial y} f(x,y) = \frac{\partial}{\partial y}[2x + 3y^2] = \frac{\partial}{\partial y}2x + \frac{\partial}{\partial y}3y^2 =$$
$$= 2 \cdot \frac{\partial}{\partial y}x + 3 \cdot \frac{\partial}{\partial y}y^2 = 2 \cdot 0 + 3 \cdot 2y^1 = 6y$$

In this example, we also applied the sum rule first, then moved constants to the outside of the derivatives and calculated what remained with respect to *x* and *y* individually. The only difference to the non-multivariate derivatives from the previous chapter is the "partial" part, which means

we are deriving with respect to each of the variables separately. Other than that, there is nothing new here.

Let's try something seemingly more complicated:

$$f(x,y) = 3x^3 - y^2 + 5x + 2 \quad \rightarrow$$

$$\frac{\partial}{\partial x} f(x,y) = \frac{\partial}{\partial x}[3x^3 - y^2 + 5x + 2] = \frac{\partial}{\partial x}3x^3 - \frac{\partial}{\partial x}y^2 + \frac{\partial}{\partial x}5x + \frac{\partial}{\partial x}2 =$$

$$= 3 \cdot \frac{\partial}{\partial x}x^3 - \frac{\partial}{\partial x}y^2 + 5 \cdot \frac{\partial}{\partial x}x + \frac{\partial}{\partial x}2 = 3 \cdot 3x^2 - 0 + 5 \cdot 1 + 0 = 9x^2 + 5$$

$$\frac{\partial}{\partial y} f(x,y) = \frac{\partial}{\partial y}[3x^3 - y^2 + 5x + 2] = \frac{\partial}{\partial y}3x^3 - \frac{\partial}{\partial y}y^2 + \frac{\partial}{\partial y}5x + \frac{\partial}{\partial y}2 =$$

$$= 3 \cdot \frac{\partial}{\partial y}x^3 - \frac{\partial}{\partial y}y^2 + 5 \cdot \frac{\partial}{\partial y}x + \frac{\partial}{\partial y}2 = 3 \cdot 0 - 2y^1 + 5 \cdot 0 + 0 = -2y$$

Pretty straight-forward — we're constantly applying the same rules over and over again, and we did not add any new calculation or rules in this example.

The Partial Derivative of Multiplication

Before we move on, let's introduce the partial derivative of multiplication operation:

$$f(x,y) = x \cdot y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = \frac{\partial}{\partial x}[x \cdot y] = y\frac{\partial}{\partial x}x = y \cdot 1 = y$$
$$\frac{\partial}{\partial y} f(x,y) = \frac{\partial}{\partial y}[x \cdot y] = x\frac{\partial}{\partial y}y = x \cdot 1 = x$$

We have already mentioned that we need to treat the other independent variables as constants, and we also have learned that we can move constants to the outside of the derivative. That's exactly how we solve the calculation of the partial derivative of multiplication — we treat other variables as constants, like numbers, and move them outside of the derivative. It turns out that when we derive with respect to *x*, *y* is treated as a constant, and the result equals *y* multiplied by the derivative of *x* with respect to *x,* which is *1*. The whole derivative then results with *y*. The intuition behind this example is when calculating the partial derivative with respect to *x*, every change of *x* by *1* changes the function's output by *y*. For example, if *y=3* and *x=1*, the result is *1·3=3*. When we change *x* by *1* so *y=3* and *x=2*, the result is *2·3=6*. We changed *x* by *1* and the result changed by *3*, by the *y*. That's what the partial derivative of this function with respect to *x* tells us.

Let's introduce a third input variable and add multiplication of variables for another example:

$$f(x,y,z) = 3x^3z - y^2 + 5z + 2yz \quad \rightarrow$$
$$\frac{\partial}{\partial x} f(x,y,z) = \frac{\partial}{\partial x}[3x^3z - y^2 + 5z + 2yz] =$$
$$= \frac{\partial}{\partial x}3x^3z - \frac{\partial}{\partial x}y^2 + \frac{\partial}{\partial x}5z + \frac{\partial}{\partial x}2yz =$$
$$= 3z \cdot \frac{\partial}{\partial x}x^3 - \frac{\partial}{\partial x}y^2 + 5 \cdot \frac{\partial}{\partial x}z + 2 \cdot \frac{\partial}{\partial x}yz =$$
$$= 3z \cdot 3x^2 - 0 + 5 \cdot 0 + 2 \cdot 0 = 9x^2z$$

$$f(x,y,z) = 3x^3z - y^2 + 5z + 2yz \quad \rightarrow$$

$$\begin{aligned}
\frac{\partial}{\partial y} f(x,y,z) &= \frac{\partial}{\partial y}[3x^3z - y^2 + 5z + 2yz] = \\
&= \frac{\partial}{\partial y}3x^3z - \frac{\partial}{\partial y}y^2 + \frac{\partial}{\partial y}5z + \frac{\partial}{\partial y}2yz = \\
&= 3 \cdot \frac{\partial}{\partial y}x^3z - \frac{\partial}{\partial y}y^2 + 5 \cdot \frac{\partial}{\partial y}z + 2z \cdot \frac{\partial}{\partial y}y = \\
&= 3 \cdot 0 - 2y + 5 \cdot 0 + 2z \cdot 1 = -2y + 2z
\end{aligned}$$

$$f(x,y,z) = 3x^3z - y^2 + 5z + 2yz \quad \rightarrow$$

$$\begin{aligned}
\frac{\partial}{\partial z} f(x,y,z) &= \frac{\partial}{\partial z}[3x^3z - y^2 + 5z + 2yz] = \\
&= \frac{\partial}{\partial z}3x^3z - \frac{\partial}{\partial z}y^2 + \frac{\partial}{\partial z}5z + \frac{\partial}{\partial z}2yz = \\
&= 3x^3 \cdot \frac{\partial}{\partial z}z - \frac{\partial}{\partial z}y^2 + 5 \cdot \frac{\partial}{\partial z}z + 2y \cdot \frac{\partial}{\partial z}z = \\
&= 3x^3 \cdot 1 - 0 + 5 \cdot 1 + 2y \cdot 1 = 3x^3 + 5 + 2y
\end{aligned}$$

The only new operation here is, as mentioned, moving variables other than the one that we derive with respect to, outside of the derivative. The results in this example appear more complicated, but only because of the existence of other variables in them — variables that are treated as constants during derivation. Equations of the derivatives are longer, but not necessarily more complicated.

The reason to learn about partial derivatives is we'll be calculating the partial derivatives of multivariate functions soon, an example of which is the neuron. From the code perspective and the *Dense* layer class, more specifically, the *forward* method of this class, we're passing in a single variable — the input array, containing either a batch of samples or outputs from the previous layer. From the math perspective, each value of this single variable (an array) is a separate input — it contains as many inputs as we have data in the input array. For example, if we pass a vector of *4* values to the neuron, it's a singular variable in the code, but *4* separate inputs in the equation. This forms a function that takes multiple inputs. To learn about the impact that each input makes to the function's output, we'll need to calculate the partial derivative of this function with respect to each of its inputs, which we'll explain in detail in the next chapter.

The Partial Derivative of *Max*

Derivatives and partial derivatives are not limited to addition and multiplication operations, or constants. We need to derive them for the other functions that we used in the forward pass, one of which is the derivative of the *max()* function:

$$f(x, y) = max(x, y) \quad \rightarrow \quad \frac{\partial}{\partial x} f(x, y) = \frac{\partial}{\partial x} max(x, y) = 1(x > y)$$

The max function returns the greatest input. We know that the derivative of *x* with respect to x equals *1,* so the derivative of this function with respect to *x* equals 1 if *x* is greater than *y*, since the function will return *x*. In the other case, where *y* is greater than *x* and will get returned instead, the derivative of *max()* with respect to *x* equals 0 — we treat *y* as a constant, and the derivative of *y* with respect to *x* equals 0. We can denote that as *1(x > y)*, which means *1* if the condition is met, and *0* otherwise. We could also calculate the partial derivative of *max()* with respect to *y*, but we won't need it anywhere in this book.

One special case for the derivative of the *max()* function is when we have only one variable parameter, and the other parameter is always constant at *0*. This means that we want whichever is bigger in return — *0* or the input value, effectively clipping the input value at *0* from the positive side. Handling this is going to be useful when we calculate the derivative of the **ReLU** activation function since that activation function is defined as *max(x, 0)*:

$$f(x) = max(x, 0) \quad \rightarrow \quad \frac{d}{dx} f(x) = \frac{d}{dx} max(x, 0) = 1(x > 0)$$

Notice that since this function takes a single parameter, we used the *d* operator instead of the ∂ to calculate the non-partial derivative. In this case, the derivative is *1* when *x* is greater than *0,* otherwise, it's *0*.

The Gradient

As we mentioned at the beginning of this chapter, the gradient is a **vector** composed of all of the partial derivatives of a function, calculated with respect to each input variable.

Let's return to one of the partial derivatives of the sum operation that we calculated earlier:

$$f(x, y, z) = 3x^3z - y^2 + 5z + 2yz \quad \rightarrow$$

$$\frac{\partial}{\partial x} f(x, y, z) = 9x^2z$$

$$\frac{\partial}{\partial y} f(x, y, z) = -2y + 2z$$

$$\frac{\partial}{\partial z} f(x, y, z) = 3x^3 + 5 + 2y$$

If we calculate all of the partial derivatives, we can form a gradient of the function. Using different notations, it looks as follows:

$$\nabla f(x, y, z) = \begin{bmatrix} \frac{\partial}{\partial x} f(x, y, z) \\ \frac{\partial}{\partial y} f(x, y, z) \\ \frac{\partial}{\partial z} f(x, y, z) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} f(x, y, z) = \begin{bmatrix} 9x^2z \\ -2y + 2z \\ 3x^3 + 5 + 2y \end{bmatrix}$$

That's all we have to know about the **gradient** - it's a vector of all of the possible partial derivatives of the function, and we denote it using the ∇ — nabla symbol that looks like an inverted delta symbol.

We'll be using **derivatives** of single-parameter functions and **gradients** of multivariate functions to perform **gradient descent** using the **chain rule,** or, in other words, to perform the **backward pass**, which is a part of the model training. How exactly we'll do that is the subject of the next chapter.

The Chain Rule

During the forward pass, we're passing the data through the neurons, then through the activation function, then through the neurons in the next layer, then through another activation function, and so on. We're calling a function with an input parameter, taking an output, and using that output as an input to another function. For this simple example, let's take 2 functions: *f* and *g*:

$$z = f(x)$$
$$y = g(z)$$

x is the input data, z is an output of the function f, but also an input for the function g, and y is an output of the function g. We could write the same calculation as:

$$y = g(f(x))$$

In this form, we do not use the intermediate z variable, showing that function g takes the output of function f directly as an input. This does not differ much from the above 2 equations but shows an important property of functions chained this way — since x is an input to the function f and then the output of the function f is an input to the function g, the output of the function g is influenced by x in some way, so there must exist a derivative which can inform us of this influence.

The forward pass through our model is a chain of functions similar to these examples. We are passing in samples, the data flows through all of the layers, and activation functions to form an output. Let's bring the equation and the code of the example model from chapter 1:

$$L = -\sum_{l=1}^{N} y_l log(\forall_{j=1}^{n_3} \frac{e^{\sum_{i=1}^{n_2}(\forall_{j=1}^{n_2} max(0,\sum_{i=1}^{n_1}(\forall_{j=1}^{n_1} max(0,\sum_{i=1}^{n_0} X_i w_{1,i,j}+b_{1,j}))_i w_{2,i,j}+b_{2,j}))_i w_{3,i,j}+b_{3,j}}{\sum_{k=1}^{n_3} e^{\sum_{i=1}^{n_2}(\forall_{j=1}^{n_2} max(0,\sum_{i=1}^{n_1}(\forall_{j=1}^{n_1} max(0,\sum_{i=1}^{n_0} X_i w_{1,i,j}+b_{1,k}))_i w_{2,i,j}+b_{2,k}))_i w_{3,i,k}+b_{3,k}}})$$

https://nnfs.io

```
loss = -np.log(
    np.sum(
        y * np.exp(
            np.dot(
                np.maximum(
                    0,
                    np.dot(
                        np.maximum(
                            0,
                            np.dot(
                                X,
                                w1.T
                            ) + b1
                        ),
                        w2.T
                    ) + b2
                ),
                w3.T
            ) + b3
        ) /
        np.sum(
            np.exp(
                np.dot(
                    np.maximum(
                        0,
                        np.dot(
                            np.maximum(
                                0,
                                np.dot(
                                    X,
                                    w1.T
                                ) + b1
                            ),
                            w2.T
                        ) + b2
                    ),
                    w3.T
                ) + b3
            ),
            axis=1,
            keepdims=True
        )
    )
)
```

https://nnfs.io

Fig 8.01: Code for a forward pass of an example neural network model.

If you look closely, you'll see that we are presenting the loss as a big function, or a chain of functions, of multiple inputs — input data, weights, and biases. We are passing input data to the first layer where we also have that layer's weights and biases, then the outputs flow through the ReLU activation function, and another layer, which brings more weights and biases, and another ReLU activation, up to the end — the output layer and softmax activation. The model output, along with the targets, is passed to the loss function, which returns the model's error. We can look at the loss function not only as a function that takes the model's output and targets as parameters to produce the error, but also as a function that takes targets, samples, and all of the weights and biases as inputs if we chain all of the functions performed during the forward pass as we've just shown in the images. To improve loss, we need to learn how each weight and bias impacts it. How to do that for a chain of functions? By using the chain rule. This rule says that the derivative of a function chain is a product of derivatives of all of the functions in this chain, for example:

$$\frac{d}{dx}f(g(x)) = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx} = f'(g(x)) \cdot g'(x)$$

First, we wrote the derivative of the outer function, *f(g(x))*, with respect to the inner function, *g(x)*, as this inner function is its parameter. Next, we multiplied it by the derivative of the inner function, *g(x)*, with respect to its parameters, *x*. We also denoted this derivative using 2 different notations. With 3 functions and multiple inputs, the partial derivative of this function with respect to *x* is as follows (we can't use the prime notation in this case since we have to mention which variable we are deriving with respect to):

$$\frac{\partial}{\partial x}f(g(y, h(x, z))) = \frac{\partial f(g(y, h(x, z)))}{\partial g(y, h(x, z))} \cdot \frac{\partial g(y, h(x, z))}{\partial h(x, z)} \cdot \frac{\partial h(x, z)}{\partial x}$$

To calculate the partial derivative of a chain of functions with respect to some parameter, we take the partial derivative of the outer function with respect to the inner function in a chain to the parameter. Then multiply this partial derivative by the partial derivative of the inner function with respect to the more inner function in a chain to the parameter, then multiply this by the partial derivative of the more inner function with respect to the other function in the chain. We repeat this all the way down to the parameter in question. Notice, for example, how the middle derivative is with respect to *h(x, z)* and not *y* as *h(x, z)* is in the chain to the parameter *x*. The **chain rule** turns out to be the most important rule in finding the impact of singular input to the output of a chain of functions, which is the calculation of loss in our case. We'll use it again in the next chapter when we discuss and code backpropagation. For now, let's cover an example of the chain rule.

Let's solve the derivative of $h(x) = 3(2x^2)^5$. The first thing that we can notice here is that we have a complex function that can be split into two simpler functions. First is an equation part contained inside the parentheses, which we can write as $g(x) = 2x^2$. That's the inside function that we exponentiate and multiply with the rest of the equation. The remaining part of the equation can then be written as $f(y) = 3(y)^5$. *y* in this case is what we denoted as $g(x)=2x^2$ and when we combine it back, we get $h(x) = f(g(x)) = 3(2x^2)^5$. To calculate a derivative of this function, we start by taking that outside exponent, the ^{5}, and place it in front of the component that we are exponentiating to multiply it later by the leading 3, giving us 15. We then subtract 1 from the ^{5} exponent, leaving us with a ^{4}.

$$h(x) = f(g(x)) = 3(2x^2)^5 \quad \rightarrow \quad f'(g(x)) = 3 \cdot 5(2x^2)^{5-1} = 15(2x^2)^4$$

Then the chain rule informs us to multiply the above derivative of the outer function, with the derivative of the interior function, giving us:

$$\rightarrow \quad h'(x) = f'(g(x)) \cdot g'(x) = 15(2x^2)^4 \cdot \frac{d}{dx}2x^2 = 15(2x^2)^4 \cdot 2 \cdot \frac{d}{dx}x^2 =$$
$$= 15(2x^2)^4 \cdot 2 \cdot 2x^1 = 15(2x^2)^4 \cdot 4x$$

Recall that $4x$ was the derivative of $2x^2$, which is the inner function, $g(x)$. This highlights the **chain rule** concept in an example, allowing us to calculate the derivatives of more complex functions by chaining together the derivatives. Note that we multiplied by the derivative of that interior function, but left the interior function *unchanged* within the derivative of the outer function.

In theory, we could just stop here with a perfectly-usable derivative of the function. We can enter some input into $15(2x^2)^4 \cdot 4x$ and get the answer. That said, we can also go ahead and simplify this function for more practice. Coming back to the original problem, so far we've found:

$$f(x) = 3(2x^2)^5 \quad \rightarrow \quad f'(x) = 15(2x^2)^4 \cdot 4x$$

To simplify this derivative function, we first take $(2x^2)^4$ and distribute the ^{4} exponent:

$$f'(x) = 15(2x^2)^4 \cdot 4x = 15 \cdot (2^4 \cdot x^{2 \cdot 4}) \cdot 4x = 15 \cdot 2^4 \cdot x^8 \cdot 4x$$

Combine the *x's*:

$$f'(x) = 15 \cdot 2^4 \cdot x^8 \cdot 4x = 15 \cdot 2^4 \cdot 4 \cdot x^{8+1} = 15 \cdot 2^4 \cdot 4 \cdot x^9$$

And the constants:

$$f'(x) = 15 \cdot 2^4 \cdot 4 \cdot x^9 = 15 \cdot 16 \cdot 4 \cdot x^9 = 960x^9$$

We'll simplify derivatives later as well for faster computation — there's no reason to repeat the same operations when we can solve them in advance.

Hopefully, now you understand what derivatives and partial derivatives are, what the gradient is, what the derivative of the loss function with respect to weights and biases means, and how to use the chain rule. For now, these terms might sound disconnected, but we're going to use them all to perform gradient descent in the backpropagation step, which is the subject of the next chapters.

Summary

Let's summarize the rules that we have learned in this chapter.

The partial derivative of the sum with respect to any input equals 1:

$$f(x,y) = x + y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = 1$$
$$\frac{\partial}{\partial y} f(x,y) = 1$$

The partial derivative of the multiplication operation with 2 inputs, with respect to any input, equals the other input:

$$f(x,y) = x \cdot y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = y$$
$$\frac{\partial}{\partial y} f(x,y) = x$$

The partial derivative of the max function of 2 variables with respect to any of them is 1 if this variable is the biggest and 0 otherwise. An example of x:

$$f(x,y) = max(x,y) \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = 1(x > y)$$

The derivative of the max function of a single variable and 0 equals 1 if the variable is greater than 0 and 0 otherwise:

$$f(x) = max(x,0) \quad \rightarrow \quad \frac{d}{dx} f(x) = 1(x > 0)$$

The derivative of chained functions equals the product of the partial derivatives of the subsequent functions:

$$\frac{d}{dx}f(g(x)) = \frac{d}{dg(x)}f(g(x)) \cdot \frac{d}{dx}g(x) = f'(g(x)) \cdot g'(x)$$

The same applies to the partial derivatives. For example:

$$\frac{\partial}{\partial x}f(g(y, h(x, z))) = f'(g(y, h(x, z))) \cdot g'(y, h(x, z)) \cdot h'(x, z)$$

The gradient is a vector of all possible partial derivatives. An example of a triple-input function:

$$\nabla f(x, y, z) = \begin{bmatrix} \frac{\partial}{\partial x}f(x, y, z) \\ \frac{\partial}{\partial y}f(x, y, z) \\ \frac{\partial}{\partial z}f(x, y, z) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} f(x, y, z)$$

Supplementary Material: https://nnfs.io/ch8
Chapter code, further resources, and errata for this chapter.

Chapter 9

Backpropagation

Now that we have an idea of how to measure the impact of variables on a function's output, we can begin to write the code to calculate these partial derivatives to see their role in minimizing the model's loss. Before applying this to a complete neural network, let's start with a simplified forward pass with just one neuron. Rather than backpropagating from the loss function for a full neural network, let's backpropagate the ReLU function for a single neuron and act as if we intend to minimize the output for this single neuron. We're first doing this only as a demonstration to simplify the explanation, since minimizing the output from a ReLU activated neuron doesn't serve any purpose other than as an exercise. Minimizing the loss value is our end goal, but in this case, we'll start by showing how we can leverage the chain rule with derivatives and partial derivatives to calculate the impact of each variable on the ReLU activated output. We'll also start by minimizing this more basic output before jumping to the full network and overall loss.

Let's quickly recall the forward pass and atomic operations that we need to perform for this single neuron and ReLU activation. We'll use an example neuron with 3 inputs, which means that it also has 3 weights and a bias:

```
Ň ł äĝgfld hğgfld ġgflā  fi ĨĮĴłĻ ŁĞĬłĦĽ
Ń ł ähġgfld hĝgfld ğgflā  fi ŃĦĨÍÎĻĽ
Ġ ł ĝgfl  fi ĠĨĞĽ
```

We then start with the first input, Ňäflā, and the related weight, Ńäflā:

Fig 9.01: Beginning a forward pass with the first input and weight.

We have to multiply the input by the weight:

```
Ň ł äĝgfld hğgfld ġgflā  fi ĨĮĴłĻ ŁĞĬłĦĽ
Ń ł ähġgfld hĝgfld ğgflā  fi ŃĦĨÍÎĻĽ
Ġ ł ĝgfl  fi ĠĨĞĽ


ŇŃfl ł Ňäflā ś Ńäflā
ĴĹĨĮĻâŇŃflã


ńńń
hġgfl
```

Visually:

Fig 9.02: The first input and weight multiplication.

We repeat this operation for Ňĝ, Ńĝ and Ňğ, Ńğ pairs:

```
ŇŃĝ ł Ňäĝā ő Ńäĝā
ŇŃğ ł Ňäğā ő Ńäğā
ĴĹĨĮĻâŇŃĝd ŇŃğã


ńńń
ğgfl ħgfl
```

Visually:

Fig 9.03: Input and weight multiplication of all of the inputs.

Code all together:

```
ſ FIJĹŃĞĹĤ ĴĞĽĹ
Ň ł äĝgfld hğgfld ġgflā  ſ ĨĮĴłĻ ŁĞĬłĦĹ
Ń ł ähġgfld hĝgfld ğgflā  ſ ŃĦĨĺĴĻĹ
Ġ ł ĝgfl  ſ ĠĨĞĹ

ſ MłĬĻĨĴĬÑĨĮĺ ĨĮĴłĻĹ ĠÑ ŃĦĨĺĴĻĹ
ŇŃfl ł Ňäflā ő Ńäflā
ŇŃĝ ł Ňäĝā ő Ńäĝā
ŇŃğ ł Ňäğā ő Ńäğā
ĴĹĨĮĻâŇŃfld ŇŃĝd ŇŃğã


ńńń
hġgfl ğgfl ħgfl
```

The next operation to perform is a sum of all weighted inputs with a bias:

```
ƒ FIJĹŃĞĹÎ ĴĞĽĽ
Ň ł äĝgfld hğgfld ġgflā  ƒ ĨĮĴŁĻ ĿĞĬŁĦĽ
Ń ł ähġgfld hĝgfld ğgflā  ƒ ŃĦĨÍÎĻĽ
Ġ ł ĝgfl  ƒ ĠĨĞĽ

ƒ MŁĬĻĨĴĬÑĨĮÍ ĨĮĴŁĻĽ ĠÑ ŃĦĨÍÎĻĽ
ŇŃfl ł Ňäflā ǻ Ńäflā
ŇŃĝ ł Ňäĝā ǻ Ńäĝā
ŇŃğ ł Ňäğā ǻ Ńäğā
ĴĹĨĮĻâŇŃfld ŇŃĝd ŇŃğd Ġã

ƒ AĤĤĨĮÍ ŃĦĨÍÎĻĦĤ ĨĮĴŁĻĽ ĞĮĤ Ğ ĠĨĞĽ
Ņ ł ŇŃfl ĸ ŇŃĝ ĸ ŇŃğ ĸ Ġ
ĴĹĨĮĻâŅã


ńńń
hġgfl ğgfl ħgfl ĝgfl
ħgfl
```

Fig 9.04: Weighted inputs and bias addition.

This forms the neuron's output. The last step is to apply the ReLU activation function on this output:

```
ń FIJĹŃĞĹĤ ĴĞĽĽ
Ň ł äĝgfld hğgfld ġgflā  ń ĨĮĴŁĻ ĽĞĬŁĦĽ
Ń ł ähġgfld hĝgfld ğgflā  ń ŃĦĨÍÎĻĽ
Ġ ł ĝgfl  ń ĠĨĞĽ

ń MŁĬĻĨĴĬÑĨĮÍ ĨĮĴŁĻĽ ĠÑ ŃĦĨÍÎĻĽ
ŇŃfl ł Ňäflā ń́ Ńäflā
ŇŃĝ ł Ňäĝā ń́ Ńäĝā
ŇŃğ ł Ňäğā ń́ Ńäğā
ĴĹĨĮĻâŇŃfld ŇŃĝd ŇŃğd Ġã

ń AĤĤĨĮÍ ŃĦĨÍÎĻĦĤ ĨĮĴŁĻĽ ĞĮĤ Ğ ĠĨĞĽ
Ņ ł ŇŃfl к ŇŃĝ к ŇŃğ к Ġ
ĴĹĨĮĻâŅã

ń RĦLU ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
Ñ ł ĬĞŇâŅd flã
ĴĹĨĮĻâÑã


ńńń
hġgfl ğgfl ħgfl ĝgfl
ħgfl
ħgfl
```

Fig 9.05: ReLU activation applied to the neuron output.

This is the full forward pass through a single neuron and a ReLU activation function. Let's treat all of these chained functions as one big function which takes input values (x), weights (w), and bias (b), as inputs, and outputs y. This big function consists of multiple simpler functions — there is a multiplication of input values and weights, sum of these values and bias, as well as a *max* function as the ReLU activation — 3 chained functions in total:

The first step is to backpropagate our gradients by calculating derivatives and partial derivatives with respect to each of our parameters and inputs. To do this, we're going to use the **chain rule**. Recall that the chain rule for a function stipulates that the derivative for nested functions like *f(g(x))* solves to:

$$\frac{d}{dx}f(g(x)) = \frac{d}{dg(x)}f(g(x)) \cdot \frac{d}{dx}g(x) = f'(g(x)) \cdot g'(x)$$

This big function that we just mentioned can be, in the context of our neural network, loosely interpreted as:

$$ReLU(\sum[inputs \cdot weights] + bias)$$

Or in the form that matches code more precisely as:

$$ReLU(x_0w_0 + x_1w_1 + x_2w_2 + b)$$

Our current task is to calculate how much each of the inputs, weights, and a bias impacts the output. We'll start by considering what we need to calculate for the partial derivative of w_0, for example. But first, let's rewrite our equation to the form that will allow us to determine how to calculate the derivatives more easily:

$y = ReLU(sum(mul(x_0, w_0), mul(x_1, w_1), mul(x_2, w_2), b))$

The above equation contains 3 nested functions: *ReLU*, a sum of weighted inputs and a bias, and multiplications of the inputs and weights. To calculate the impact of the example weight, w_0, on the output, the chain rule tells us to calculate the derivative of *ReLU* with respect to its parameter, which is the sum, then multiply it with the partial derivative of the sum operation with respect to its $mul(x_0, w_0)$ input, as this input contains the parameter in question. Then, multiply this with the partial derivative of the multiplication operation with respect to the x_0 input. Let's see this in a simplified equation:

$$\frac{\partial}{\partial x_0}[ReLU(sum(mul(x_0, w_0), mul(x_1, w_1), mul(x_2, w_2), b))] =$$

$$\frac{dReLU()}{dsum()} \cdot \frac{\partial sum()}{\partial mul(x_0, w_0)} \cdot \frac{\partial mul(x_0, w_0)}{\partial x_0}$$

For legibility, we did not denote the *ReLU*() parameter, which is the full sum, and the sum parameters, which are all of the multiplications of inputs and weights. We excluded this because the equation would be longer and harder to read. This equation shows that we have to calculate the derivatives and partial derivatives of all of the atomic operations and multiply them to acquire the impact that x_0 makes on the output. We can then repeat this to calculate all of the other remaining impacts. The derivatives with respect to the weights and a bias will inform us about their impact and will be used to update these weights and bias. The derivatives with respect to inputs are used to chain more layers by passing them to the previous function in the chain.

We'll have multiple chained layers of neurons in the neural network model, followed by the loss function. We want to know the impact of a given weight or bias on the loss. That means that we will have to calculate the derivative of the loss function (which we'll do later in this chapter) and apply the chain rule with the derivatives of all activation functions and neurons in all of the consecutive layers. The derivative with respect to the layer's inputs, as opposed to the derivative with respect to the weights and biases, is not used to update any parameters. Instead, it is used to chain to another layer (which is why we backpropagate to the previous layer in a chain).

During the backward pass, we'll calculate the derivative of the loss function, and use it to multiply with the derivative of the activation function of the output layer, then use this result to multiply by the derivative of the output layer, and so on, through all of the hidden layers and activation functions. Inside these layers, the derivative with respect to the weights and biases will form the gradients that we'll use to update the weights and biases. The derivatives with respect to inputs will form the gradient to chain with the previous layer. This layer can calculate the impact of its weights and biases on the loss and backpropagate gradients on inputs further.

For this example, let's assume that our neuron receives a gradient of *1* from the next layer. We're making up this value for demonstration purposes, and a value of *1* won't change the values, which means that we can more easily show all of the processes. We are going to use the color of red for derivatives:

Fig 9.06: Initial gradient (received during backpropagation).

Recall that the derivative of *ReLU()* with respect to its input is *1,* if the input is greater than *0,* and *0* otherwise:

$$f(x) = max(x, 0) \quad \rightarrow \quad \frac{d}{dx} f(x) = 1(x > 0)$$

We can write that in Python as:

```
ĹĦĬŁçĤŅ ł âĝg ĨÌ Ņ ń fl ĦĬĹĦ flgã
```

Where the ĤĹĦĬŁçĤŅ means the derivative of the *ReLU* function with respect to *z* — we used *z* instead of *x* from the equation since the equation denotes the *max* function in general, and we are applying it to the neuron's output, which is *z*.

The input value to the *ReLU* function is *6*, so the derivative equals *1*. We have to use the chain rule and multiply this derivative with the derivative received from the next layer, which is *1* for the purpose of this example:

```
ń FIJĹŃĞĹĤ ĴĞĹĹ
Ň ł äĝgfld hĝgfld ġgflā  ń ĨĮĴŁĻ ĽĞĬŁĦĹ
Ń ł ähġgfld hĝgfld ĝgflā  ń ŃĦĨÍÎĻĹ
Ġ ł ĝgfl  ń ĠĨĞĹ
```

```
ﬁ MŁĬĻĨĴĬÑĨĮÍ ĨĮĴŁĻĽ ĠÑ ŃĦĨÍÎĻĽ
ŇŃﬂ ł Ňäﬂā ő Ńäﬂā
ŇŃĝ ł Ňäĝā ő Ńäĝā
ŇŃğ ł Ňäğā ő Ńäğā

ﬁ AĤĤĨĮÍ ŃĦĨÍÎĻĦĤ ĨĮĴŁĻĽ ĞĮĤ Ğ ĠĨĞĽ
Ņ ł ŇŃﬂ κ ŇŃĝ κ ŇŃğ κ Ġ

ﬁ RĦLU ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
Ñ ł ĬĞŇâŅd ﬂã

ﬁ BĞĢĪŃĞĹĤ ĴĞĽĽ

ﬁ TÎĦ ÂĦĹĨĿĞĻĨĿĦ ÌĹIJİ ĻÎĦ ĮĦŇĻ ĬĞÑĦĹ
ĤĿĞĬŁĦ ł ĝgﬂ

ﬁ DĦĹĨĿĞĻĨĿĦ IJÌ RĦLU ĞĮĤ ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ
ĤĹĦĬŁçĤŅ ł ĤĿĞĬŁĦ ő âĝg ĨÌ Ņ ń ﬂ ĦĬĽĦ ﬂgã
ĴĹĨĮĻâĤĹĦĬŁçĤŅã


ńńń
ĝgﬂ
```

Fig 9.07: Derivative of the ReLU function and chain rule.

This results with the derivative of *1*:

Fig 9.08: ReLU and chain rule gradient.

Moving backward through our neural network, what is the function that comes immediately before we perform the activation function?

It's a sum of the weighted inputs and bias. This means that we want to calculate the partial derivative of the sum function, and then, using the chain rule, multiply this by the partial derivative of the subsequent, outer, function, which is *ReLU*. We'll call these results the:

- ĤĹĦĬŁçĤŇŃfl — the partial **d**erivative of the **ReLU** w.r.t. the first weighed input, $\boldsymbol{w_0x_0}$,
- ĤĹĦĬŁçĤŇŃĝ — the partial **d**erivative of the **ReLU** w.r.t. the second weighed input, $\boldsymbol{w_1x_1}$,
- ĤĹĦĬŁçĤŇŃğ — the partial **d**erivative of the **ReLU** w.r.t. the third weighed input, $\boldsymbol{w_2x_2}$d
- ĤĹĦĬŁçĤĠ — the partial **d**erivative of the **ReLU** with respect to the bias, $\boldsymbol{b}$.

The partial derivative of the sum operation is always *1*, no matter the inputs:

$$f(x,y) = x + y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = 1$$

$$\frac{\partial}{\partial y} f(x,y) = 1$$

The weighted inputs and bias are summed at this stage. So we will calculate the partial derivatives of the sum operation with respect to each of these, multiplied by the partial derivative for the subsequent function (using the chain rule), which is the *ReLU* function, denoted by ĤĹĦĬŁçĤŅ

For the first partial derivative:

```
ĤĽŁİçĤŇŃfl ł ĝ
ĤĹĦĬŁçĤŇŃfl ł ĤĹĦĬŁçĤŅ ấ ĤĽŁİçĤŇŃfl
```

To be clear, the ĤĽŁİçĤŇŃfl above means the partial **d**erivative of the **sum** with respect to the **x** (input), **w**eighted, for the **0**th pair of inputs and weights. *1* is the value of this partial derivative, which we multiply, using the chain rule, with the derivative of the subsequent function, which is the *ReLU* function.

Again, we have to apply the chain rule and multiply the derivative of the ReLU function with the partial derivative of the sum, with respect to the first weighted input:

```
ń FIJĹŃĞĹĤ ĴĞĽĽ
Ň ł äĝgfld hğgfld ġgflā  ń ĨĮĴŁĻ ĽĞĬŁĦĽ
Ń ł ähġgfld hĝgfld ğgflā  ń ŃĦĨÍÎĻĽ
Ġ ł ĝgfl  ń ĠĨĞĽ

ń MŁĬĻĨĴĬÑĨĮÍ ĨĮĴŁĻĽ ĠÑ ŃĦĨÍÎĻĽ
ŇŃfl ł Ňäflā ấ Ńäflā
ŇŃĝ ł Ňäĝā ấ Ńäĝā
ŇŃğ ł Ňäğā ấ Ńäğā

ń AĤĤĨĮÍ ŃĦĨÍÎĻĦĤ ĨĮĴŁĻĽ ĞĮĤ Ğ ĠĨĞĽ
Ņ ł ŇŃfl ĸ ŇŃĝ ĸ ŇŃğ ĸ Ġ

ń RĦLU ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
Ñ ł ĬĞŇâŅd flã

ń BĞĢĪŃĞĹĤ ĴĞĽĽ

ń TĨĦ ĤĦĹĨĽĞĻĨĽĦ ÌĹIJĬ ĻĨĦ ĮĦŇĻ ĬĞÑĦĹ
ĤĽĞĬŁĦ ł ĝgfl

ń DĦĹĨĽĞĻĨĽĦ IJÌ RĦLU ĞĮĤ ĻĨĦ ĢĨĞĨĮ ĹŁĬĦ
ĤĹĦĬŁçĤŅ ł ĤĽĞĬŁĦ ấ âĝg ĨÌ Ņ ń fl ĦĬĽĦ flgã
ĴĹĨĮĻâĤĹĦĬŁçĤŅã

ń PĞĹĻĨĞĬ ĤĦĹĨĽĞĻĨĽĦĽ IJÌ ĻĨĦ ĬŁĬĻĨĴĬĨĢĞĻĨIJĮd ĻĨĦ ĢĨĞĨĮ ĹŁĬĦ
ĤĽŁİçĤŇŃfl ł ĝ
ĤĹĦĬŁçĤŇŃfl ł ĤĹĦĬŁçĤŅ ấ ĤĽŁİçĤŇŃfl
ĴĹĨĮĻâĤĹĦĬŁçĤŇŃflã


ńńń
ĝgfl
ĝgfl
```

Fig 9.09: Partial derivative of the sum function w.r.t. the first weighted input; the chain rule.

This results with a partial derivative of 1 again:

Fig 9.10: The sum and chain rule gradient (for the first weighted input).

We can then perform the same operation with the next weighed input:

```
ĤĹŁİçĤŇŃĝ ł ĝ
ĤĹĦĬŁçĤŇŃĝ ł ĤĹĦĬŁçĤŅ ǻ ĤĹŁİçĤŇŃĝ
```

Fig 9.11: Partial derivative of the sum function w.r.t. the second weighted input; the chain rule.

Which results with the next calculated partial derivative:

Fig 9.12: The sum and chain rule gradient (for the second weighted input).

And the last weighted input:

```
ĤĽŁİçĤŇŃğ ł ĝ
ĤĹĦĬŁçĤŇŃğ ł ĤĹĦĬŁçĤŅ ǻ ĤĽŁİçĤŇŃğ
```

Fig 9.13: Partial derivative of the sum function w.r.t. the third weighted input; the chain rule.

Fig 9.14: The sum and chain rule gradient (for the third weighted input).

Then the bias:

```
ÂĹŁİçÂĠ ł ĝ
ÂĹĦĬŁçÂĠ ł ÂĹĦĬŁçÂŅ ǻ ÂĹŁİçÂĠ
```

Fig 9.15: Partial derivative of the sum function w.r.t. the bias; the chain rule.

Fig 9.16: The sum and chain rule gradient (for the bias).

Let’s add these partial derivatives, with the applied chain rule, to our code:

```
# FIJĹŃĞĹÂ ĴĞĽĽ
Ň ł äĝgfld hğgfld ġgflā  # ĨĮĴŁĻ ĿĞĬŁĦĽ
Ń ł ähġgfld hĝgfld ğgflā  # ŃĦĨÍÎĻĽ
Ġ ł ĝgfl  # ĠĨĞĽ

# MŁĬĻĨĴĬÑĨĮÍ ĨĮĴŁĻĽ ĠÑ ŃĦĨÍÎĻĽ
ŇŃfl ł Ňäflā ő Ńäflā
ŇŃĝ ł Ňäĝā ő Ńäĝā
ŇŃğ ł Ňäğā ő Ńäğā

# AĤĤĨĮÍ ŃĦĨÍÎĻĦĤ ĨĮĴŁĻĽ ĞĮĤ Ğ ĠĨĞĽ
Ņ ł ŇŃfl к ŇŃĝ к ŇŃğ к Ġ

# RĦLU ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
Ñ ł ĬĞŇâŅd flã

# BĞĢĪŃĞĹÂ ĴĞĽĽ

# TÎĦ ÂĦĹĨĿĞĻĨĿĦ ÌĹIJĬ ĻÎĦ ĮĦŇĻ ĬĞÑĦĹ
ÂĿĞĬŁĦ ł ĝgfl

# DĦĹĨĿĞĻĨĿĦ IJÌ RĦLU ĞĮĤ ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ
ÂĹĦĬŁçÂŅ ł ÂĿĞĬŁĦ ő âĝg ĨÌ Ņ ń fl ĦĬĽĦ flgã
ĴĹĨĮĻâÂĹĦĬŁçÂŅã

# PĞĹĻĨĞĬ ÂĦĹĨĿĞĻĨĿĦĽ IJÌ ĻÎĦ ĬŁĬĻĨĴĬĨĢĞĻĨIJĮd ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ
ÂĽŁĬçÂŇŃfl ł ĝ
ÂĽŁĬçÂŇŃĝ ł ĝ
ÂĽŁĬçÂŇŃğ ł ĝ
ÂĽŁĬçÂĠ ł ĝ
ÂĹĦĬŁçÂŇŃfl ł ÂĹĦĬŁçÂŅ ő ÂĽŁĬçÂŇŃfl
ÂĹĦĬŁçÂŇŃĝ ł ÂĹĦĬŁçÂŅ ő ÂĽŁĬçÂŇŃĝ
ÂĹĦĬŁçÂŇŃğ ł ÂĹĦĬŁçÂŅ ő ÂĽŁĬçÂŇŃğ
ÂĹĦĬŁçÂĠ ł ÂĹĦĬŁçÂŅ ő ÂĽŁĬçÂĠ
ĴĹĨĮĻâÂĹĦĬŁçÂŇŃfld ÂĹĦĬŁçÂŇŃĝd ÂĹĦĬŁçÂŇŃğd ÂĹĦĬŁçÂĠã


ńńń
ĝgfl
ĝgfl ĝgfl ĝgfl ĝgfl
```

Continuing backward, the function that comes before the sum is the multiplication of weights and inputs. The derivative for a product is whatever the input is being multiplied by. Recall:

$$f(x, y) = x \cdot y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x, y) = y$$

$$\frac{\partial}{\partial y} f(x, y) = x$$

The partial derivative of *f* with respect to *x* equals *y*. The partial derivative of *f* with respect to *y* equals *x*. Following this rule, the partial derivative of the first *weighted input* with respect to the *input* equals the *weight* (the other input of this function). Then, we have to apply the chain rule and multiply this partial derivative with the partial derivative of the subsequent function, which is the sum (we just calculated its partial derivative earlier in this chapter):

```
ĤİłĬçĤŇfl ł Ńäflā
ĤĹĦĬłçĤŇfl ł ĤĹĦĬłçĤŇŃfl ő ĤİłĬçĤŇfl
```

This means that we are calculating the partial derivative with respect to the x_0 input, the value of which is w_0, and we are applying the chain rule with the derivative of the subsequent function, which is `ĤĹĦĬłçĤŇŃfl`.

This is a good time to point out that, as we apply the chain rule in this way — working backward by taking the *ReLU()* derivative, taking the summing operation's derivative, multiplying both, and so on, this is a process called **backpropagation** using the **chain rule**. As the name implies, the resulting output function's gradients are passed back through the neural network, using multiplication of the gradient of subsequent functions from later layers with the current one. Let's add this partial derivative to the code and show it on the chart:

```
fi FIJĹŃĞĹĤ ĴĞĹĽ
Ň ł äĝgfld hğgfld ġgflā  fi ĨĮĴłĻ ĽĞĬłĦĽ
Ń ł ähġgfld hĝgfld ğgflā  fi ŃĦĨĺĤĮĽ
Ġ ł ĝgfl  fi ĠĨĞĽ

fi MłĬĻĨĴĬÑĨĮĺ ĨĮĴłĻĽ ĠÑ ŃĦĨĺĤĮĽ
ŇŃfl ł Ňäflā ő Ńäflā
ŇŃĝ ł Ňäĝā ő Ńäĝā
ŇŃğ ł Ňäğā ő Ńäğā

fi AĤĤĨĮĺ ŃĦĨĺĤĮĦĤ ĨĮĴłĻĽ ĞĮĤ Ğ ĠĨĞĽ
Ņ ł ŇŃfl ĸ ŇŃĝ ĸ ŇŃğ ĸ Ġ

fi RĦLU ĞĢĻĨĽĞĻĨIJĮ ÌłĮĢĻĨIJĮ
Ñ ł ĬĞŇâŅd flā

fi BĞĢĪŃĞĹĤ ĴĞĹĽ
```

```
ﬁ TÎĦ ÂĦĹĨĿĞĻĨĿĦ ÌĹIJİ ĻÎĦ ĮĦŇĻ ĬĞÑĦĹ
ÂĿĞĬŁĦ ł ĝgﬂ

ﬁ DĦĹĨĿĞĻĨĿĦ IJÌ RĦLU ĞĮÂ ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ
ÂĹĦĬŁçÂŅ ł ÂĿĞĬŁĦ ǻ âĝg ĨÌ Ņ ń ﬂ ĦĬĹĦ ﬂgã
ĴĹĨĮĻâÂĹĦĬŁçÂŅã

ﬁ PĞĹĻĨĞĬ ÂĦĹĨĿĞĻĨĿĦĹ IJÌ ĻÎĦ ĮŁĬĻĨĴĬĨĢĞĻĨIJĮd ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ
ÂĹŁĮçÂŇŃﬂ ł ĝ
ÂĹŁĮçÂŇŃĝ ł ĝ
ÂĹŁĮçÂŇŃğ ł ĝ
ÂĹŁĮçÂĠ ł ĝ
ÂĹĦĬŁçÂŇŃﬂ ł ÂĹĦĬŁçÂŅ ǻ ÂĹŁĮçÂŇŃﬂ
ÂĹĦĬŁçÂŇŃĝ ł ÂĹĦĬŁçÂŅ ǻ ÂĹŁĮçÂŇŃĝ
ÂĹĦĬŁçÂŇŃğ ł ÂĹĦĬŁçÂŅ ǻ ÂĹŁĮçÂŇŃğ
ÂĹĦĬŁçÂĠ ł ÂĹĦĬŁçÂŅ ǻ ÂĹŁĮçÂĠ
ĴĹĨĮĻâÂĹĦĬŁçÂŇŃﬂd ÂĹĦĬŁçÂŇŃĝd ÂĹĦĬŁçÂŇŃğd ÂĹĦĬŁçÂĠã

ﬁ PĞĹĻĨĞĬ ÂĦĹĨĿĞĻĨĿĦĹ IJÌ ĻÎĦ ĮŁĬĻĨĴĬĨĢĞĻĨIJĮd ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ
ÂĮŁĬçÂŇﬂ ł Ńäﬂã
ÂĹĦĬŁçÂŇﬂ ł ÂĹĦĬŁçÂŇŃﬂ ǻ ÂĮŁĬçÂŇﬂ
ĴĹĨĮĻâÂĹĦĬŁçÂŇﬂã


ńńń
ĝgﬂ
ĝgﬂ ĝgﬂ ĝgﬂ ĝgﬂ
hġgﬂ
```

Fig 9.17: Partial derivative of the multiplication function w.r.t. the first input; the chain rule.

Fig 9.18: The multiplication and chain rule gradient (for the first input).

We perform the same operation for other inputs and weights:

```
ſ FIJĹŃĞĹĤ ĴĞĽĽ
Ň ł äĝgfld hğgfld ġgflā  ſ ĨĮĴŁĻ ĿĞĬŁĦĽ
Ń ł ähġgfld hĝgfld ğgflā  ſ ŃĦĨÍÎĻĽ
Ġ ł ĝgfl  ſ ĠĨĞĽ

ſ MŁĬĻĨĴĬÑĨĮÍ ĨĮĴŁĻĽ ĠÑ ŃĦĨÍÎĻĽ
ŇŃfl ł Ňäflā ǻ Ńäflā
ŇŃĝ ł Ňäĝā ǻ Ńäĝā
ŇŃğ ł Ňäğā ǻ Ńäğā

ſ AĤĤĨĮÍ ŃĦĨÍÎĻĦĤ ĨĮĴŁĻĽ ĞĮĤ Ğ ĠĨĞĽ
Ņ ł ŇŃfl ĸ ŇŃĝ ĸ ŇŃğ ĸ Ġ

ſ RĦLU ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
Ñ ł ĬĞŇâŅd flā

ſ BĞĢĨŃĞĹĤ ĴĞĽĽ

ſ TÎĦ ÂĦĹĨĿĞĻĨĿĦ ÌĹIJİ ĻÎĦ ĮĦŇĻ ĬĞÑĦĹ
ÂĿĞĬŁĦ ł ĝgfl
```

```
fi DHĹĨĿĞĻĨĿH IJÌ RHLU ĞĮĤ ĻÎH ĢÎĞĨĮ ĹŁĬH
ÎĹHĬŁçÎĤŅ ł ÎĿĞĬŁH ő âĝg ĨÌ Ņ ń fl HĬĹH flgã
ĴĹĨĮĻâÎĹHĬŁçÎĤŅã

fi PĞĹĻĨĞĬ ÎHĹĨĿĞĻĨĿHĹ IJÌ ĻÎH ĬŁĬĻĨĴĬĨĢĞĻĨIJĮd ĻÎH ĢÎĞĨĮ ĹŁĬH
ÎĹŁİçÎĤŃfl ł ĝ
ÎĹŁİçÎĤŃĝ ł ĝ
ÎĹŁİçÎĤŃğ ł ĝ
ÎĹŁİçÎĤĠ ł ĝ
ÎĹHĬŁçÎĤŃfl ł ÎĹHĬŁçÎĤŅ ő ÎĹŁİçÎĤŃfl
ÎĹHĬŁçÎĤŃĝ ł ÎĹHĬŁçÎĤŅ ő ÎĹŁİçÎĤŃĝ
ÎĹHĬŁçÎĤŃğ ł ÎĹHĬŁçÎĤŅ ő ÎĹŁİçÎĤŃğ
ÎĹHĬŁçÎĤĠ ł ÎĹHĬŁçÎĤŅ ő ÎĹŁİçÎĤĠ
ĴĹĨĮĻâÎĹHĬŁçÎĤŃfld ÎĹHĬŁçÎĤŃĝd ÎĹHĬŁçÎĤŃğd ÎĹHĬŁçÎĤĠã

fi PĞĹĻĨĞĬ ÎHĹĨĿĞĻĨĿHĹ IJÌ ĻÎH ĬŁĬĻĨĴĬĨĢĞĻĨIJĮd ĻÎH ĢÎĞĨĮ ĹŁĬH
ÎİŁĬçÎĤfl ł Ńäflā
ÎİŁĬçÎĤĝ ł Ńäĝā
ÎİŁĬçÎĤğ ł Ńäğā
ÎİŁĬçÎŃfl ł Ñäflā
ÎİŁĬçÎŃĝ ł Ñäĝā
ÎİŁĬçÎŃğ ł Ñäğā
ÎĹHĬŁçÎĤfl ł ÎĹHĬŁçÎĤŃfl ő ÎİŁĬçÎĤfl
ÎĹHĬŁçÎŃfl ł ÎĹHĬŁçÎĤŃfl ő ÎİŁĬçÎŃfl
ÎĹHĬŁçÎĤĝ ł ÎĹHĬŁçÎĤŃĝ ő ÎİŁĬçÎĤĝ
ÎĹHĬŁçÎŃĝ ł ÎĹHĬŁçÎĤŃĝ ő ÎİŁĬçÎŃĝ
ÎĹHĬŁçÎĤğ ł ÎĹHĬŁçÎĤŃğ ő ÎİŁĬçÎĤğ
ÎĹHĬŁçÎŃğ ł ÎĹHĬŁçÎĤŃğ ő ÎİŁĬçÎŃğ
ĴĹĨĮĻâÎĹHĬŁçÎĤfld ÎĹHĬŁçÎŃfld ÎĹHĬŁçÎĤĝd ÎĹHĬŁçÎŃĝd ÎĹHĬŁçÎĤğd ÎĹHĬŁçÎŃğã


ńńń
ĝgfl
ĝgfl ĝgfl ĝgfl ĝgfl
hġgfl ĝgfl hĝgfl hğgfl ğgfl ġgfl
```

Fig 9.19: Complete backpropagation graph.

Anim 9.01-9.19: https://nnfs.io/pro

That's the complete set of the activated neuron's partial derivatives with respect to the inputs, weights and a bias.

Recall the equation from the beginning of this chapter:

$$\frac{\partial}{\partial x_0}[ReLU(sum(mul(x_0, w_0), mul(x_1, w_1), mul(x_2, w_2), b))] =$$

$$\frac{dReLU()}{dsum()} \cdot \frac{\partial sum()}{\partial mul(x_0, w_0)} \cdot \frac{\partial mul(x_0, w_0)}{\partial x_0}$$

Since we have the complete code and we are applying the chain rule from this equation, let's see what we can optimize in these calculations. We applied the chain rule to calculate the

partial derivative of the ReLU activation function with respect to the first input, x_0. In our code, let's take the related lines of the code and simplify them:

```
ÂĹĦĬŁçÂŇfl ł ÂĹĦĬŁçÂŇŃfl ő ÂİŁĬçÂŇfl
```

where:

```
ÂİŁĬçÂŇfl ł Ńäflā
```

then:

```
ÂĹĦĬŁçÂŇfl ł ÂĹĦĬŁçÂŇŃfl ő Ńäflā
```

where:

```
ÂĹĦĬŁçÂŇŃfl ł ÂĹĦĬŁçÂŅ ő ÂĽŁİçÂŇŃfl
```

then:

```
ÂĹĦĬŁçÂŇfl ł ÂĹĦĬŁçÂŅ ő ÂĽŁİçÂŇŃfl ő Ńäflā
```

where:

```
ÂĽŁİçÂŇŃfl ł ĝ
```

then:

```
ÂĹĦĬŁçÂŇfl ł ÂĹĦĬŁçÂŅ ő ĝ ő Ńäflā ł ÂĹĦĬŁçÂŅ ő Ńäflā
```

where:

```
ÂĹĦĬŁçÂŅ ł ÂŁĞĬŁĦ ő âĝg ĨÌ Ņ ń fl ĦĬĽĦ flgã
```

then:

```
ÂĹĦĬŁçÂŇfl ł ÂŁĞĬŁĦ ő âĝg ĨÌ Ņ ń fl ĦĬĽĦ flgã ő Ńäflā
```

```
drelu_dx0 = drelu_dxw0 * dmul_dx0

dmul_dx0 = w[0]

drelu_dxw0 = drelu_dz * dsum_dxw0

dsum_dxw0 = 1

drelu_dz = dvalue * (1. if z > 0 else 0.)
```

https://nnfs.io

Fig 9.20: How to apply the chain rule for the partial derivative of ReLU w.r.t. first input

```
drelu_dx0 = dvalue * (1. if z > 0 else 0.) * w[0]
```

Fig 9.21: The chain rule applied for the partial derivative of ReLU w.r.t. first input

Anim 9.20-9.21: https://nnfs.io/com

In this equation, starting from the left-hand side, is the derivative calculated in the next layer, with respect to its inputs — this is the gradient backpropagated to the current layer, which is the derivative of the *ReLU* function, and the partial derivative of the neuron's function with respect to the x_0 input. This is all multiplied by applying the chain rule to calculate the impact of the input to the neuron on the whole function's output.

The partial derivative of a neuron's function, with respect to the weight, is the input related to this weight, and, with respect to the input, is the related weight. The partial derivative of the neuron's function with respect to the bias is always 1. We multiply them with the derivative of the subsequent function (which was *1* in this example) to get the final derivatives. We are going to code all of these derivatives in the Dense layer's class and the ReLU activation class for the backpropagation step.

All together, the partial derivatives above, combined into a vector, make up our gradients. Our gradients could be represented as:

```
ĤŇ ł äĤĹĦĬłçĤŇfld ĤĹĦĬłçĤŇĝd ĤĹĦĬłçĤŇğā  ńĩ ÍĹĞĤĨĦĮĻĹ IJĮ ĨĮĴłĻĹ
ĤŃ ł äĤĹĦĬłçĤŃfld ĤĹĦĬłçĤŃĝd ĤĹĦĬłçĤŃğā  ńĩ ÍĹĞĤĨĦĮĻĹ IJĮ ŃĦĨÍÎĻĹ
ĤĠ ł ĤĹĦĬłçĤĠ  ńĩ ÍĹĞĤĨĦĮĻ IJĮ ĠĨĞĹgggĬłĹĹĻ ĝ ĠĨĞĹ ÎĦĹĦg
```

For this single neuron example, we also won't need our ĤŇ. With many layers, we will continue backpropagating to preceding layers with the partial derivative with respect to our inputs.

Continuing the single neuron example, we can now apply these gradients to the weights to hopefully minimize the output. This is typically the purpose of the **optimizer** (discussed in the following chapter), but we can show a simplified version of this task by directly applying a negative fraction of the gradient to our weights. We apply a negative fraction to this gradient since we want to decrease the final output value, and the gradient shows the direction of the steepest ascent. For example, our current weights and bias are:

```
ĴĹĨĮĻâŃd Ġã


ńńń
ähġgfld hĝgfld ğgflā ĝgfl
```

We can then apply a fraction of the gradients to these values:

```
Ńäflā кł hflgflflĝ ő ĤŃäflā
Ńäĝā кł hflgflflĝ ő ĤŃäĝā
Ńäğā кł hflgflflĝ ő ĤŃäğā
Ġ кł hflgflflĝ ő ĤĠ

ĴĹĨĮĻâŃd Ġã


ńńń
ähġgflflĝd hflgîîíd ĝgîîìā flgîîî
```

Now, we've slightly changed the weights and bias in such a way so as to decrease the output somewhat intelligently. We can see the effects of our tweaks on the output by doing another forward pass:

```
ńĩ MłĬĻĨĴĬÑĨĮÍ ĨĮĴłĻĹ ĠÑ ŃĦĨÍÎĻĹ
ŇŃfl ł Ňäflā ő Ńäflā
ŇŃĝ ł Ňäĝā ő Ńäĝā
ŇŃğ ł Ňäğā ő Ńäğā
```

```
ńi AĤĤĨĮÍ
Ņ ł ŇŃfl κ ŇŃĝ κ ŇŃğ κ Ġ

ńi RŦLU ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
Ñ ł ÌĞŇâŅd flã
ĴĹĨĮĻâÑã


ńńń
ĥgîíĥ
```

We've successfully decreased this neuron's output from 6.000 to 5.985. Note that it does not make sense to decrease the neuron's output in a real neural network; we were doing this purely as a simpler exercise than the full network. We want to decrease the loss value, which is the last calculation in the chain of calculations during the forward pass, and it's the first one to calculate the gradient during the backpropagation. We've minimized the ReLU output of a single neuron only for the purpose of this example to show that we actually managed to decrease the value of chained functions intelligently using the derivatives, partial derivatives, and chain rule. Now, we'll apply the one-neuron example to the list of samples and expand it to an entire layer of neurons. To begin, let's set a list of 3 samples for input, where each sample consists of 4 features. For this example, our network will consist of a single hidden layer, containing 3 neurons (lists of 3 weight sets and 3 biases). We're not going to describe the forward pass again, but the backward pass, in this case, needs further explanation.

So far, we have performed an example backward pass with a single neuron, which received a singular derivative to apply the chain rule. Let's consider multiple neurons in the following layer. A single neuron of the current layer connects to all of them — they all receive the output of this neuron. What will happen during backpropagation? Each neuron from the next layer will return a partial derivative of its function with respect to this input. The neuron in the current layer will receive a vector consisting of these derivatives. We need this to be a singular value for a singular neuron. To continue backpropagation, we need to sum this vector.

Now, let's replace the current singular neuron with a layer of neurons. As opposed to a single neuron, a layer outputs a vector of values instead of a singular value. Each neuron in a layer connects to all of the neurons in the next layer. During backpropagation, each neuron from the current layer will receive a vector of partial derivatives the same way that we described for a single neuron. With a layer of neurons, it'll take the form of a list of these vectors, or a 2D array. We know that we need to perform a sum, but what should we sum and what is the result supposed to be? Each neuron is going to output a gradient of the partial derivatives with respect to all of its inputs, and all neurons will form a list of these vectors. We need to sum along the inputs — the first input to all of the neurons, the second input, and so on. We'll have to sum columns.

To calculate the partial derivatives with respect to inputs, we need the weights — the partial derivative with respect to the input equals the related weight. This means that the array of partial derivatives with respect to all of the inputs equals the array of weights. Since this array is transposed, we'll need to sum its rows instead of columns. To apply the chain rule, we need to multiply them by the gradient from the subsequent function.

In the code to show this, we take the transposed weights, which are the transposed array of the derivatives with respect to inputs, and multiply them by their respective gradients (related to given neurons) to apply the chain rule. Then we sum along with the inputs. Then we calculate the gradient for the next layer in backpropagation. The "next" layer in backpropagation is the previous layer in the order of creation of the model:

```
ÕİĴIJĹĻ ĮłİĴÑ ĞĽ ĮĴ

ﬁ PĞĽĽĦĤ ÕĮ ÍĹĞĤÕĦĮĻ ÌĹIJİ ĻÎĦ ĮĦŇĻ ĬĞÑĦĹ
ﬁ ÌIJĹ ĻÎĦ ĴłĹĴIJĽĦ IJÌ ĻÎÕĽ ĦŇĞİĴĬĦ ŃĦďĹĦ ÍIJÕĮÍ ĻIJ łĽĦ
ﬁ Ğ ŁĦĢĻIJĹ IJÌ ĝĽ
ĤŁĞĬłĦĽ ł ĮĴgĞĹĹĞÑâäägd ĝgd ĝgāāã

ﬁ WĦ ÎĞŁĦ ġ ĽĦĻĽ IJÌ ŃĦÕÍÎĻĽ h IJĮĦ ĽĦĻ ÌIJĹ ĦĞĢÎ ĮĦłĹIJĮ
ﬁ ŃĦ ÎĞŁĦ ǵ ÕĮĴłĻĽd ĻÎłĽ ǵ ŃĦÕÍÎĻĽ
ﬁ ĹĦĢĞĬĬ ĻÎĞĻ ŃĦ ĪĦĦĴ ŃĦÕÍÎĻĽ ĻĹĞĮĽĴIJĽĦĤ
ŃĦÕÍÎĻĽ ł ĮĴgĞĹĹĞÑâääflgğd flgíd hflgĥd ĝād
                    äflgĥd hflgîĝd flgğħd hflgĥād
                    ähflgğħd hflgğìd flgĝìd flgííìāāãgT

ﬁ Ľłİ ŃĦÕÍÎĻĽ IJÌ ÍÕŁĦĮ ÕĮĴłĻ
ﬁ ĞĮĤ İłĬĻÕĴĬÑ ĠÑ ĻÎĦ ĴĞĽĽĦĤ ÕĮ ÍĹĞĤÕĦĮĻ ÌIJĹ ĻÎÕĽ ĮĦłĹIJĮ
ĤŃﬁ ł ĽłİâŃĦÕÍÎĻĽäflāāáĤŁĞĬłĦĽäflā
ĤŃĝ ł ĽłİâŃĦÕÍÎĻĽäĝāāáĤŁĞĬłĦĽäflā
ĤŃğ ł ĽłİâŃĦÕÍÎĻĽäğāāáĤŁĞĬłĦĽäflā
ĤŃġ ł ĽłİâŃĦÕÍÎĻĽäġāāáĤŁĞĬłĦĽäflā

ĤÕĮĴłĻĽ ł ĮĴgĞĹĹĞÑâäĤŃﬁd ĤŃĝd ĤŃğd ĤŃġāã

ĴĹÕĮĻâĤÕĮĴłĻĽã


ńńń
ä flgǵǵ hflgġí hflgflì  ĝgġìā
```

ĤÕĮĴłĻĽ is a gradient of the neuron function with respect to inputs.

We defined the gradient of the subsequent function (dvalues) as a row vector, which we'll explain shortly. From NumPy's perspective, and since both weights and dvalues are NumPy arrays, we can simplify the dx0 to dx3 calculation. Since the weights array is formatted so that the rows contain weights related to each input (weights for all neurons for the given input), we can multiply them by the gradient vector directly:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ńĩ PĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌĹIJİ ĻÎĦ ĮĦŇĻ ĬĞÑĦĹ
ńĩ ÌIJĹ ĻÎĦ ĴŁĹĴIJĽĦ IJÌ ĻĨĨĽ ĦŇĞİĴĬĦ ŃĦďĹĦ ÍIJĨĮÍ ĻIJ ŁĽĦ
ńĩ Ğ ŁĦĢĻIJĹ IJÌ ĝĽ
ĤŁĞĬŁĦĽ ł ĮĴgĞĹĹĞÑâäägg d ggd ggāāā

ńĩ WĦ ÎĞŁĦ ġ ĽĦĻĽ IJÌ ŃĦĨÍÎĻĽ h IJĮĦ ĽĦĻ ÌIJĹ ĦĞĢÎ ĮĦŁĹIJĮ
ńĩ ŃĦ ÎĞŁĦ ǵ ĨĮĴŁĻĽd ĻÎŁĽ ǵ ŃĦĨÍÎĻĽ
ńĩ ĹĦĢĞĬĬ ĻÎĞĻ ŃĦ ĪĦĦĴ ŃĦĨÍÎĻĽ ĻĹĞĮĽĴIJĽĦĤ
ŃĦĨÍÎĻĽ ł ĮĴgĞĹĹĞÑâäägg d flgíd hflgĥd ĝād
                äflgĥd hflgîĝd flgĝħd hflgĥād
                ähflgĝħd hflgĝìd flgĝìd flgííāāāgT

ńĩ ĽŁİ ŃĦĨÍÎĻĽ IJÌ ÍĨŁĦĮ ĨĮĴŁĻ
ńĩ ĞĮĤ İŁĬĻĨĴĬÑ ĠÑ ĻÎĦ ĴĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌIJĹ ĻÎĨĽ ĮĦŁĹIJĮ
ĤŃfl ł ĽŁİâŃĦĨÍÎĻĽäflāāĤŁĞĬŁĦĽäflāā
ĤŃĝ ł ĽŁİâŃĦĨÍÎĻĽäĝāāĤŁĞĬŁĦĽäflāā
ĤŃĝ ł ĽŁİâŃĦĨÍÎĻĽäĝāāĤŁĞĬŁĦĽäflāā
ĤŃġ ł ĽŁİâŃĦĨÍÎĻĽäġāāĤŁĞĬŁĦĽäflāā

ĤĨĮĴŁĻĽ ł ĮĴgĞĹĹĞÑâäĤŃfld ĤŃĝd ĤŃĝd ĤŃġāā

ĴĹĨĮĻâĤĨĮĴŁĻĽā


ńńń
ä flgġġ hflgġí hflgflì  ĝgġìā
```

You might already see where we are going with this — the sum of the multiplication of the elements is the dot product. We can achieve the same result by using the `ĮĴgĤIJĻ` function. For this to be possible, we need to match the "inner" shapes and decide the first dimension of the result, which is the first dimension of the first parameter. We want the output of this calculation to be of the shape of the gradient from the subsequent function — recall that we have one partial derivative for each neuron and multiply it by the neuron's partial derivative with respect to its input. We then want to multiply each of these gradients with each of the partial derivatives that are related to this neuron's inputs, and we already noticed that they are rows. The dot product takes rows from the first argument and columns from the second to perform multiplication and sum; thus, we need to transpose the weights for this calculation:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ń PĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌĹIJİ ĻÎĦ ĮĦŇĻ ĬĞÑĦĹ
ń ÌIJĹ ĻÎĦ ĴŁĹĴIJĽĦ IJÌ ĻÎĨĽ ĦŇĞİĴĬĦ ŃĦďĹĦ ÍIJĨĮÍ ĻIJ ŁĽĦ
ń Ğ ŁĦĢĻIJĹ IJÌ ĝĽ
ĤŁĞĬŁĦĽ ł ĮĴgĞĹĹĞÑâäägd ĝgd ĝgāāã

ń WĦ ÎĞŁĦ ġ ĽĦĻĽ IJÌ ŃĦĨÍÎĻĽ h IJĮĦ ĽĦĻ ÌIJĹ ĦĞĢÎ ĮĦŁĹIJĮ
ń ŃĦ ÎĞŁĦ ǵ ĨĮĴŁĻĽd ĻÎŁĽ ǵ ŃĦĨÍÎĻĽ
ń ĹĦĢĞĬĬ ĻÎĞĻ ŃĦ ĪĦĦĴ ŃĦĨÍÎĻĽ ĻĹĞĮĽĴIJĽĦĤ
ŃĦĨÍÎĻĽ ł ĮĴgĞĹĹĞÑâäägg̃d flgíd hflgĥd ĝād
                äflgĥd hflgîĝd flgg̃ħd hflgĥād
                ähflgg̃ħd hflgg̃ìd flgĝìd flgíìāāãgT

ń ĽŁİ ŃĦĨÍÎĻĽ IJÌ ÍĨŁĦĮ ĨĮĴŁĻ
ń ĞĮĤ İŁĬĻĨĴĬÑ ĠÑ ĻÎĦ ĴĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌIJĹ ĻÎĨĽ ĮĦŁĹIJĮ
ĤĨĮĴŁĻĽ ł ĮĴgĤIJĻâĤŁĞĬŁĦĽäflād ŃĦĨÍÎĻĽgTã

ĴĹĨĮĻâĤĨĮĴŁĻĽã


ńńń
ä flgǵǵ hflgġí hflgflì  ĝgġìā
```

We have to account for one more thing — a batch of samples. So far, we have been using a single sample responsible for a single gradient vector that is backpropagated between layers. The row vector that we created for `ĤŁĞĬŁĦĽ` is in preparation for a batch of data. With more samples, the layer will return a list of gradients, which we *almost* handle correctly for. Let's replace the singular gradient `ĤŁĞĬŁĦĽäflā` with a full list of gradients, `ĤŁĞĬŁĦĽ`, and add more example gradients to this list:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ń PĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌIJĹ ĻÎĦ ĮĦŇĻ ĬĞÑĦĹ
ń ÌIJĹ ĻÎĦ ĴŁĹĴIJĽĦ IJÌ ĻÎĨĽ ĦŇĞİĴĬĦ ŃĦďĹĦ ÍIJĨĮÍ ĻIJ ŁĽĦ
ń ĞĮ ĞĹĹĞÑ IJÌ ĞĮ ĨĮĢĹĦİĦĮĻĞĬ ÍĹĞĤĨĦĮĻ ŁĞĬŁĦĽ
ĤŁĞĬŁĦĽ ł ĮĴgĞĹĹĞÑâäägd ĝgd ĝgād
                äg̃gd g̃gd g̃gād
                äġgd ġgd ġgāāã

ń WĦ ÎĞŁĦ ġ ĽĦĻĽ IJÌ ŃĦĨÍÎĻĽ h IJĮĦ ĽĦĻ ÌIJĹ ĦĞĢÎ ĮĦŁĹIJĮ
ń ŃĦ ÎĞŁĦ ǵ ĨĮĴŁĻĽd ĻÎŁĽ ǵ ŃĦĨÍÎĻĽ
ń ĹĦĢĞĬĬ ĻÎĞĻ ŃĦ ĪĦĦĴ ŃĦĨÍÎĻĽ ĻĹĞĮĽĴIJĽĦĤ
ŃĦĨÍÎĻĽ ł ĮĴgĞĹĹĞÑâäägg̃d flgíd hflgĥd ĝād
                äflgĥd hflgîĝd flgg̃ħd hflgĥād
                ähflgg̃ħd hflgg̃ìd flgĝìd flgíìāāãgT
```

```
ń ĽŁİ ŃĦĨÍÎĻĽ ĲÌ ÍĨĿĦĮ ĨĮĴŁĻ
ń ĞĮĤ İŁĬĻĨĴĬÑ ĠÑ ĻÎĦ ĴĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌĲĹ ĻÎĨĽ ĮĦŁĹĲĮ
ĤĨĮĴŁĻĽ ł ĮĴgĤĲĻâĤĿĞĬŁĦĽd ŃĦĨÍÎĻĽgTã

ĴĹĨĮĻâĤĨĮĴŁĻĽã


ńńń
ää ńgǵǵ hńgġí hńgńì  ĝgġìā
 ä ńgíí hńgìħ hńgĝǵ  ǧgìǵā
 ä ĝgġǧ hĝgĝǵ hńgǧĝ  ǵgĝĝāā
```

Calculating the gradients with respect to weights is very similar, but, in this case, we're going to be using gradients to update the weights, so we need to match the shape of weights, not inputs. Since the derivative with respect to the weights equals inputs, weights are transposed, so we need to transpose inputs to receive the derivative of the neuron with respect to weights. Then we use these transposed inputs as the first parameter to the dot product — the dot product is going to multiply rows by inputs, where each row, as it is transposed, contains data for a given input for all of the samples, by the columns of ĤĿĞĬŁĦĽ. These columns are related to the outputs of singular neurons for all of the samples, so the result will contain an array with the shape of the weights, containing the gradients with respect to the inputs, multiplied with the incoming gradient for all of the samples in the batch:

```
ĨİĴĲĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ń PĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌĹĲİ ĻÎĦ ĮĦŃĻ ĬĞÑĦĹ
ń ÌĲĹ ĻÎĦ ĴŁĹĴĲĽĦ ĲÌ ĻÎĨĽ ĦŃĞİĴĬĦ ŃĦďĹĦ ÍĲĨĮÍ ĻĲ ŁĽĦ
ń ĞĮ ĞĹĹĞÑ ĲÌ ĞĮ ĨĮĢĹĦİĦĮĻĞĬ ÍĹĞĤĨĦĮĻ ĿĞĬŁĦĽ
ĤĿĞĬŁĦĽ ł ĮĴgĞĹĹĞÑâäägg̃d ĝgd ĝgād
                    äǧgd ǧgd ǧgād
                    äġgd ġgd ġgāã

ń WĦ ÎĞĿĦ ġ ĽĦĻĽ ĲÌ ĨĮĴŁĻĽ h ĽĞİĴĬĦĽ
ĨĮĴŁĻĽ ł ĮĴgĞĹĹĞÑâäägd ǧd ġd ǧgĥād
                   äǧgd ĥgd hĝgd ǧād
                   ähĝgĥd ǧgìd ġgġd hńgíāãã

ń ĽŁİ ŃĦĨÍÎĻĽ ĲÌ ÍĨĿĦĮ ĨĮĴŁĻ
ń ĞĮĤ İŁĬĻĨĴĬÑ ĠÑ ĻÎĦ ĴĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌĲĹ ĻÎĨĽ ĮĦŁĹĲĮ
ĤŃĦĨÍÎĻĽ ł ĮĴgĤĲĻâĨĮĴŁĻĽgTd ĤĿĞĬŁĦĽã

ĴĹĨĮĻâĤŃĦĨÍÎĻĽã


ńńń
ää ńgĥ  ńgĥ  ńgĥā
 äǧńgĝ ǧńgĝ ǧńgĝā
 äĝńgî ĝńgî ĝńgîā
 ä ǵggĝ  ǵggĝ  ǵggĝāā
```

This output's shape matches the shape of weights because we summed the inputs for each weight and then multiplied them by the input gradient. ĤŃĦĨÍÎĻĽ is a gradient of the neuron function with respect to the weights.

For the biases and derivatives with respect to them, the derivatives come from the sum operation and always equal 1, multiplied by the incoming gradients to apply the chain rule. Since gradients are a list of gradients (a vector of gradients for each neuron for all samples), we just have to sum them with the neurons, column-wise, along axis 0.

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ſ PĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌĹIJİ ĻÎĦ ĮĦŃĻ ĬĞÑĦĹ
ſ ÌIJĹ ĻÎĦ ĴŁĹĴIJĽĦ IJÌ ĻÎĨĽ ĦŃĞİĴĬĦ ŃĦďĹĦ ÍIJĨĮÍ ĻIJ ŁĽĦ
ſ ĞĮ ĞĹĹĞÑ IJÌ ĞĮ ĨĮĢĹĦİĦĮĻĞĬ ÍĹĞĤĨĦĮĻ ĽĞĬŁĦĽ
ĤĽĞĬŁĦĽ ł ĮĴgĞĹĹĞÑâäägd ĝgd ĝgād
                    äğgd ğgd ğgād
                    äġgd ġgd ġgāāā

ſ OĮĦ ĠĨĞĽ ÌIJĹ ĦĞĢÎ ĮĦŁĹIJĮ
ſ ĠĨĞĽĦĽ ĞĹĦ ĻÎĦ ĹIJŃ ĽĦĢĻIJĹ ŃĨĻÎ Ğ ĽÎĞĴĦ âĝd ĮĦŁĹIJĮĽã
ĠĨĞĽĦĽ ł ĮĴgĞĹĹĞÑâäägd ġd flgĥāāã

ſ ĤĠĨĞĽĦĽ h ĽŁİ ĽĞĬŁĦĽd ĤIJ ĻÎĨĽ IJĽĦĹ ĽĞİĴĬĦĽ âÌĨĹĽĻ ĞŃĨĽãd ĪĦĦĴĤĨİĽ
ſ ĽĨĮĢĦ ĻÎĨĽ ĠÑ ĤĦÌĞŁĬĻ ŃĨĬĬ ĴĹIJĤŁĢĦ Ğ ĴĬĞĨĮ ĬĨĽĻ h
ſ ŃĦ ĦŃĴĬĞĨĮĦĤ ĻÎĨĽ ĨĮ ĻÎĦ ĢÎĞĴĻĦĹ ǵ
ĤĠĨĞĽĦĽ ł ĮĴgĽŁİâĤĽĞĬŁĦĽd ĞŃĨĽłfld ĪĦĦĴĤĨİĽłTĹŁĦã

ĴĹĨĮĻâĤĠĨĞĽĦĽã


ńńń
äähg ħg ħgāā
```

ĪĦĦĴĤĨİĽ lets us keep the gradient as a row vector — recall the shape of biases array.

The last thing to cover here is the derivative of the ReLU function. It equals *1* if the input is greater than *0* and *0* otherwise. The layer passes its outputs through the *ReLU()* activation during the forward pass. For the backward pass, *ReLU()* receives a gradient of the same shape. The derivative of the ReLU function will form an array of the same shape, filled with 1 when the related input is greater than 0, and 0 otherwise. To apply the chain rule, we need to multiply this array with the gradients of the following function:

```
ĨĬĴIJĹĻ ĮŁĬĴÑ ĞĽ ĮĴ

ńĩ EŃĞĬĴĬĦ ĬĞÑĦĹ IJŁĻĴŁĻ
Ņ ł ĮĴgĞĹĹĞÑâäägd ğd hġd hǵād
              äğd hìd hĝd ġād
              ähĝd ğd ĥd hĝāāã

ĤĿĞĬŁĦĽ ł ĮĴgĞĹĹĞÑâäägd ğd ġd ǵād
                    äĥd ħd ìd íād
                    äîd ĝfld ĝĝd ĝğāāã

ńĩ RĦLU ĞĢĻĨĿĞĻĨIJĮďĽ ĤĦĹĨĿĞĻĨĿĦ
ĤĹĦĬŁ ł ĮĴgŅĦĹIJĽçĬĨĬĦâŅã
ĤĹĦĬŁäŅ ń flā ł ĝ

ĴĹĨĮĻâĤĹĦĬŁã

ńĩ TĨĦ ĢĨĞĨĮ ĹŁĬĦ
ĤĹĦĬŁ ǻł ĤĿĞĬŁĦĽ

ĴĹĨĮĻâĤĹĦĬŁã


ńńń
äägĝ ĝ fl flā
 äĝ fl fl ĝā
 äfl ĝ ĝ flāā
ää ĝ  ğ  fl  flā
 ä ĥ  fl  fl  íā
 ä fl ĝfl ĝĝ  flāā
```

To calculate the ReLU derivative, we created an array filled with zeros. `ĮĴgŅĦĹIJĽçĬĨĬĦ` is a NumPy function that creates an array filled with zeros, with the shape of the array from its parameter, the `Ņ` array in our case, which is an example output of the neuron. Following the *ReLU()* derivative, we then set the values related to the inputs greater than *0* as 1. We then print this table to see and compare it to the gradients. In the end, we multiply this array with the gradient of the subsequent function and print the result.

We can now simplify this operation. Since the *ReLU()* derivative array is filled with 1s, which do not change the values multiplied by them, and 0s that zero the multiplying value, this means that we can take the gradients of the subsequent function and set to 0 all of the values that correspond to the *ReLU()* input and are equal to or less than 0:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ƒ EŃĞİĴĬĦ ĬĞÑĦĹ IJŁĻĴŁĻ
Ņ ł ĮĴgĞĹĹĞÑâäägd ğd hġd hǵād
              äğd hìd hĝd ġād
              ähĝd ğd ĥd hĝāāã

ĤĽĞĬŁĦĽ ł ĮĴgĞĹĹĞÑâäägd ğd ġd ǵād
                    äĥd ħd ìd íād
                    äîd ĝfld ĝĝd ĝğāāã

ƒ RĦLU ĞĢĻĨĽĞĻĨIJĮďĽ ĤĦĹĨĽĞĻĨĽĦ
ƒ ŃĨĻÎ ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ ĞĴĴĬĨĦĤ
ĤĹĦĬŁ ł ĤĽĞĬŁĦĽgĢIJĴÑâã
ĤĹĦĬŁäŅ l·ł flā ł fl

ĴĹĨĮĻâĤĹĦĬŁã


ńńń
ää ĝ  ğ  fl  flā
 ä ĥ  fl  fl  íā
 ä fl ĝfl ĝĝ  flāā
```

The copy of ĤĽĞĬŁĦĽ ensures that we don't modify it during the ReLU derivative calculation.

Let's combine the forward and backward pass of a single neuron with a full layer and batch-based partial derivatives. We'll minimize ReLU's output, once again, only for this example:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ƒ PĞĽĽĦĤ ĨĮ ÍĹĞĤĨĦĮĻ ÌĹIJİ ĻÎĦ ĮĦŃĻ ĬĞÑĦĹ
ƒ ÌIJĹ ĻÎĦ ĴŁĹĴIJĽĦ IJÌ ĻÎĨĽ ĦŃĞİĴĬĦ ŃĦďĹĦ ÍIJĨĮÍ ĻIJ ŁĽĦ
ƒ ĞĮ ĞĹĹĞÑ IJÌ ĞĮ ĨĮĢĹĦİĦĮĻĞĬ ÍĹĞĤĨĦĮĻ ĽĞĬŁĦĽ
ĤĽĞĬŁĦĽ ł ĮĴgĞĹĹĞÑâäägd ĝgd ĝgād
                    äğgd ğgd ğgād
                    äġgd ġgd ġgāāã

ƒ WĦ ÎĞĽĦ ġ ĽĦĻĽ IJÌ ĨĮĴŁĻĽ h ĽĞİĴĬĦĽ
ĨĮĴŁĻĽ ł ĮĴgĞĹĹĞÑâäägd ğd ġd ğgĥād
                   äğgd ĥgd hĝgd ğād
                   ähĝgĥd ğgìd ġgġd hflgíāāã
ƒ WĦ ÎĞĽĦ ġ ĽĦĻĽ IJÌ ŃĦĨÍÎĻĽ h IJĮĦ ĽĦĻ ÌIJĹ ĦĞĢÎ ĮĦŁĹIJĮ
ƒ ŃĦ ÎĞĽĦ ǵ ĨĮĴŁĻĽd ĻÎŁĽ ǵ ŃĦĨÍÎĻĽ
ƒ ĹĦĢĞĬĬ ĻÎĞĻ ŃĦ ĨĦĦĴ ŃĦĨÍÎĻĽ ĻĹĞĮĽĴIJĽĦĤ
ŃĦĨÍÎĻĽ ł ĮĴgĞĹĹĞÑâäāflgğd flgíd hflgĥd ĝād
                    äflgĥd hflgîĝd flgğħd hflgĥād
                    ähflgğħd hflgğìd flgĝìd flgíìāāãgT
```

```
ﬁ OĮĦ ĠĨĞĽ ÌIJĹ ĦĞĢÎ ĮĦŁĹIJĮ
ﬁ ĠĨĞĽĦĽ ĞĹĦ ĻÎĦ ĹIJŃ ŁĦĢĻIJĹ ŃĨĻÎ Ğ ĽÎĞĴĦ âĝd ĮĦŁĹIJĮĽã
ĠĨĞĽĦĽ ł ĮĴgĞĹĹĞÑâäägd ġd ﬂghĥāãã

ﬁ FIJĹŃĞĹĤ ĴĞĽĽ
ĬĞÑĦĹçIJŁĻĴŁĻĽ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ŃĦĨÍÎĻĽã κ ĠĨĞĽĦĽ  ﬁ DĦĮĽĦ ĬĞÑĦĹ
ĹĦĬŁçIJŁĻĴŁĻĽ ł ĮĴgĬĞŃĨĬŁĬâﬂd ĬĞÑĦĹçIJŁĻĴŁĻĽã  ﬁ RĦLU ĞĢĻĨŁĠĻĨIJĮ

ﬁ LĦĻďĽ IJĴĻĨĬĨŅĦ ĞĮĤ ĻĦĽĻ ĠĞĢĨĴĹIJĴĞÍĞĻĨIJĮ ÎĦĹĦ
ﬁ RĦLU ĞĢĻĨŁĠĻĨIJĮ h ĽĨĬŁĬĞĻĦĽ ĤĦĹĨŁĠĻĨŁĦ ŃĨĻÎ ĹĦĽĴĦĢĻ ĻIJ ĨĮĴŁĻ ŁĠĬŁĦĽ
ﬁ ÌĹIJĬ ĮĦŃĻ ĬĞÑĦĹ ĴĞĽĽĦĤ ĻIJ ĢŁĹĹĦĮĻ ĬĞÑĦĹ ĤŁĹĨĮÍ ĠĞĢĨĴĹIJĴĞÍĞĻĨIJĮ
ĤĹĦĬŁ ł ĹĦĬŁçIJŁĻĴŁĻĽgĢIJĴÑâã
ĤĹĦĬŁäĬĞÑĦĹçIJŁĻĴŁĻĽ l·ł ﬂã ł ﬂ

ﬁ DĦĮĽĦ ĬĞÑĦĹ
ﬁ ÂĨĮĴŁĻĽ h ĬŁĬĻĨĴĬÑ ĠÑ ŃĦĨÍÎĻĽ
ÂĨĮĴŁĻĽ ł ĮĴgĤIJĻâĤĹĦĬŁd ŃĦĨÍÎĻĽgTã
ﬁ ÂŃĦĨÍÎĻĽ h ĬŁĬĻĨĴĬÑ ĠÑ ĨĮĴŁĻĽ
ÂŃĦĨÍÎĻĽ ł ĮĴgĤIJĻâĨĮĴŁĻĽgTd ÂĹĦĬŁã
ﬁ ÂĠĨĞĽĦĽ h ĽŁĬ ŁĞĬŁĦĽd ÂIJ ĻÎĨĽ IJŁĦĹ ĽĞĬĴĬĦĽ âÌĨĹĽĻ ĞÑĨĽãd ĨĦĦĴÂĨĬĽ
ﬁ ĽĨĮĢĦ ĻÎĨĽ ĠÑ ÂĦÌĞŁĬĻ ŃĨĨĬ ĴĹIJÂŁĢĦ Ğ ĴĬĞĨĮ ĬĨĽĻ h
ﬁ ŃĦ ĦŃĴĬĞĨĮĦÂ ĻÎĨĽ ĨĮ ĻÎĦ ĢÎĞĴĻĦĹ ǵ
ÂĠĨĞĽĦĽ ł ĮĴgĽŁĬâÂĹĦĬŁd ĞŃĨĽłﬂd ĨĦĦĴÂĨĬĽłTĹŁĦã

ﬁ UĴÂĞĻĦ ĴĞĹĞĬĦĻĦĹĽ
ŃĦĨÍÎĻĽ κł hﬂgﬂﬂĝ ǻ ÂŃĦĨÍÎĻĽ
ĠĨĞĽĦĽ κł hﬂgﬂﬂĝ ǻ ÂĠĨĞĽĦĽ

ĴĹĨĮĻâŃĦĨÍÎĻĽã
ĴĹĨĮĻâĠĨĞĽĦĽã

ńńń
[[ 0.179515   0.5003665 -0.262746 ]
 [ 0.742093  -0.9152577 -0.2758402]
 [-0.510153   0.2529017  0.1629592]
 [ 0.971328  -0.5021842  0.8636583]]
[[1.98489  2.997739 0.497389]]
```

In this code, we replaced the plain Python functions with NumPy variants, created example data, calculated the forward and backward passes, and updated the parameters. Now we will update the dense layer and ReLU activation code with a ĠĞĢĪŃĞĹĤ method (for backpropagation), which we'll call during the backpropagation phase of our model.

```
# DHĮĽH ĬĞÑHĹ
ĢĬĞĽĽ LĞÑHĹçDHĮĽHf

    # LĞÑHĹ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ
    ÂHÌ ççĨĮĨĻççâĽHĬÌd ĨĮĴŁĻĽd ĮHŁĹIJĮĽãf
        ĽHĬÌgŃHĨÍÎĻĽ ł flgflĝ ấ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĨĮĴŁĻĽd ĮHŁĹIJĮĽã
        ĽHĬÌgĠĨĞĽHĽ ł ĮĴgŅHĹIJĽââĝd ĮHŁĹIJĮĽãã

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽãf
        ĽHĬÌgIJŁĻĴŁĻ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĽHĬÌgŃHĨÍÎĻĽã к ĽHĬÌgĠĨĞĽHĽ

# RHLU ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçRHLUf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽãf
        ĽHĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĨİŁİâfld ĨĮĴŁĻĽã
```

During the ÌIJĹŃĞĹĤ method for our LĞÑHĹçDHĮĽH class, we will want to remember what the inputs were (recall that we'll need them when calculating the partial derivative with respect to weights during backpropagation), which can be easily implemented using an object property (ĽHĬÌgĨĮĴŁĻĽ):

```
# DHĮĽH ĬĞÑHĹ
ĢĬĞĽĽ LĞÑHĹçDHĮĽHf
    ggg
    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽãf
        ggg
        ĽHĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
```

Next, we will add our backward pass (backpropagation) code that we developed previously into a new method in the layer class, which we'll call ĠĞĢĪŃĞĹĤ:

```
ĢĬĞĽĽ LĞÑĦĹçDĦIĽĦf
    ggg
    ħ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ÂĽĠĬŁĦĽãf
        ħ GĹĞĤĨĦIĻĽ IJI Ĵ ĞĹĞĠĬĦĻĦĹĽ
        ĽĦĬÌgĤŃĦĨÍÎĻĽ ł IĴgĤIJĻâĽĦĬÌgĨIĴŁĻĽgTd ÂĽĠĬŁĦĽã
        ĽĦĬÌgĤĠĨĞĽĦĽ ł IĴgĽŁĠâĤĽĠĬŁĦĽd ĞŃĨĽłfld ĪĦĦĴĤĨĠĽłTĹŁĦã
        ħ GĹĞĤĨĦIĻ IJI ĽĠĬŁĦĽ
        ĽĦĬÌgĤĨIĴŁĻĽ ł IĴgĤIJĻâĤĽĠĬŁĦĽd ĽĦĬÌgŃĦĨÍÎĻĽgTã
```

We then do the same for our ReLU class:

```
ħ RĦLU ĞĢĻĨĽĠĻĨIJI
ĢĬĞĽĽ AĢĻĨĽĠĻĨIJIçRĦLUf

    ħ FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨIĴĴŁĻĽãf
        ħ RĦĠĦĠĦĹ ĨIĴŁĻ ĽĠĬŁĦĽ
        ĽĦĬÌgĨIĴŁĻĽ ł ĨIĴŁĻĽ
        ĽĦĬÌgIJŁĻŁĻ ł IĴgĠĞŃĨĠŁĠâfld ĨIĴŁĻĽã

    ħ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ÂĽĠĬŁĦĽãf
        ħ SĨIĢĦ ŃĦ IĦĦĤ ĻIJ ĠIJĤĨÌÑ ĻÎĦ IJĹĨÍĨIĞĬ ĽĠĹĨĞĠĬĦd
        ħ ĬĦĻďĽ ĠĞĪĦ Ğ ĢIJĴÑ IJÌ ĻÎĦ ĽĠĬŁĦĽ ÌĨĹĽĻ
        ĽĦĬÌgĤĨIĴŁĻĽ ł ÂĽĠĬŁĦĽgĢIJĴÑâã

        ħ ZĦĹIJ ÍĹĞĤĨĦIĻ ŃÎĦĹĦ ĨIĴŁĻ ĽĠĬŁĦĽ ŃĦĹĦ IĦÍĞĻĨĽĦ
        ĽĦĬÌgĤĨIĴŁĻĽäĽĦĬÌgĨIĴŁĻĽ l·ł flã ł fl
```

By this point, we've covered everything we need to perform backpropagation, except for the derivative of the Softmax activation function and the derivative of the cross-entropy loss function.

Categorical Cross-Entropy loss derivative

If you are not interested in the mathematical derivation of the Categorical Cross-Entropy loss, feel free to skip to the code implementation, as derivatives are known for common loss functions, and you won't necessarily need to know how to solve them. It is a good exercise if you plan to create custom loss functions, though.

As we learned in chapter 5, the Categorical Cross-Entropy loss function's formula is:

$$L_i = -log(\hat{y}_{i,k}) \quad where\ \mathbf{k}\ is\ an\ index\ of\ "true"\ probability$$

Where L_i denotes sample loss value, *i* — *i*-th sample in a set, *k* — index of the target label (ground-true label), *y* — target values and *y-hat* — predicted values.

This formula is convenient when calculating the loss value itself, as all we need is the output of the Softmax activation function at the index of the correct class. For the purpose of the derivative calculation, we'll use the full equation mentioned back in chapter 5:

$$L_i = -\sum_j y_{i,j} log(\hat{y}_{i,j})$$

Where L_i denotes sample loss value, *i* — *i*-th sample in a set, *j* — label/output index, *y* — target values and *y-hat* — predicted values.

We'll use this full function because our current goal is to calculate the gradient, which is composed of the partial derivatives of the loss function with respect to each of its inputs (being the outputs of the Softmax activation function). This means that we cannot use the equation, which takes just the value at the index of the correct class (the first equation above). To calculate partial derivatives with respect to each of the inputs, we need an equation that takes all of them as parameters, thus the choice to use the full equation.

First, let's define the gradient equation:

$$\frac{\partial L_i}{\partial \hat{y}_{i,j}} = \frac{\partial}{\partial \hat{y}_{i,j}}[-\sum_j y_{i,j} log(\hat{y}_{i,j})] =$$

We defined the equation here as the partial derivative of the loss function with respect to each of its inputs. We already learned that the derivative of the sum equals the sum of the derivatives. We also learned that we can move constants. An example is $y_{i,j}$, as it is not what we are calculating the derivative with respect to. Let's apply these transforms:

$$= -\sum_j y_{i,j} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} log(\hat{y}_{i,j}) =$$

Now we have to solve the derivative of the logarithmic function, which is the reciprocal of its parameter, multiplied (using the chain rule) by the partial derivative of this parameter — using prime (also called Lagrange's) notation:

$$f(x) = log(h(x)) \quad \rightarrow \quad f'(x) = \frac{1}{h(x)} \cdot h'(x)$$

We can solve it further (using Leibniz's notation in this case):

$$f(x) = log(x) \quad \rightarrow \quad \frac{d}{dx} f(x) = \frac{d}{dx} log(x) = \frac{1}{x} \cdot \frac{d}{dx} x = \frac{1}{x} \cdot 1 = \frac{1}{x}$$

Let's apply this derivative:

$$= -\sum_j y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j} =$$

The partial derivative of a value with respect to this value equals 1:

$$= -\sum_j y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot 1 = -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} =$$

Since we are calculating the partial derivative with respect to the *y* given *j*, the sum is being performed over a single element and can be omitted:

$$= -\frac{y_{i,j}}{\hat{y}_{i,j}}$$

Full solution:

$$\frac{\partial L_i}{\partial \hat{y}_{i,j}} = \frac{\partial}{\partial \hat{y}_{i,j}}[-\sum_j y_{i,j} log(\hat{y}_{i,j})] = -\sum_j y_{i,j} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} log(\hat{y}_{i,j}) =$$

$$= -\sum_j y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j} = -\sum_j y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot 1 = -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} = -\frac{y_{i,j}}{\hat{y}_{i,j}}$$

The derivative of this loss function with respect to its inputs (predicted values at the i-th sample, since we are interested in a gradient with respect to the predicted values) equals the negative ground-truth vector, divided by the vector of the predicted values (which is also the output vector of the softmax function).

Categorical Cross-Entropy loss derivative code implementation

Since we derived this equation and have found that it solves to a simple division operation of 2 values, we know that, with NumPy, we can extend this operation to the sample-wise vectors of ground truth and predicted values, and further to the batch-wise arrays of them. From the coding perspective, we need to add a backward method to the Loss_CategoricalCrossentropy class. We need to pass the array of predictions and the array of true values into it and calculate the negated division of them:

```
# CĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ
ĢĬĞĽĽ LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâLIJĽĽãf
    ggg
    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĽĠĬŁĦĽd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâĤĽĞĬŁĦĽã
        # NŁİĠĦĹ IJÌ ĬĞĠĦĬĽ ĨĮ ĦĽĦĹÑ ĽĞİĴĬĦ
        # WĦďĬĬ ŁĽĦ ĻĤĦ ÌĨĹĽĻ ĽĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦİ
        ĬĞĠĦĬĽ ł ĬĦĮâĤĽĞĬŁĦĽäflãã

        # IÌ ĬĞĠĦĬĽ ĞĹĦ ĽĴĞĹĽĦd ĻŁĹĮ ĻĤĦİ ĨĮĻIJ IJĮĦhĤIJĻ ĽĦĢĻIJĹ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽĤĞĴĦã łł ĝf
            ÑçĻĹŁĦ ł ĮĴgĦÑĦâĬĞĠĦĬĽãäÑçĻĹŁĦā

        # CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł hÑçĻĹŁĦ w ĤĽĞĬŁĦĽ
        # NIJĹİĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽĞİĴĬĦĽ
```

Along with the partial derivative calculation, we are performing two additional operations. First, we're turning numerical labels into one-hot encoded vectors — prior to this, we need to check how many dimensions y_true consists of. If the shape of the labels returns a single dimension

(which means that they are shaped like a list and not like an array), they consist of discrete numbers and need to be converted to a list of one-hot encoded vectors — a two-dimensional array. If that's the case, we need to turn them into one-hot encoded vectors. We'll use the `ĮĴgĦÑĦ` method which, given a number, *n*, returns an *n*x*n* array filled with ones on the diagonal and zeros everywhere else. For example:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĮĴgĦÑĦâĥã


ńńń
ĞĹĹĞÑâäägd flgd flgd flgd flgād
       äflgd ggd flgd flgd flgād
       äflgd flgd ggd flgd flgād
       äflgd flgd flgd ggd flgād
       äflgd flgd flgd flgd ggāāā
```

We can then index this table with the numerical label to get the one-hot encoded vector that represents it:

```
ĮĴgĦÑĦâĥāäg̃ā


ńńń
ĞĹĹĞÑâäflgd ggd flgd flgd flgāā


ĮĴgĦÑĦâĥāäǵā


ńńń
ĞĹĹĞÑâäflgd flgd flgd flgd ggāā
```

If *y_true* is already one-hot encoded, we do not perform this step.

The second operation is the gradient normalization. As we'll learn in the next chapter, optimizers sum all of the gradients related to each weight and bias before multiplying them by the learning rate (or some other factor). What this means, in our case, is that the more samples we have in a dataset, the more gradient sets we'll receive at this step, and the bigger this sum will become. As a consequence, we'll have to adjust the learning rate according to each set of samples. To solve this problem, we can divide all of the gradients by the number of samples. A sum of elements divided by a count of them is their mean value (and, as we mentioned, the optimizer will perform the sum) — this way, we'll effectively normalize the gradients and make their sum's magnitude invariant to the number of samples.

Softmax activation derivative

The next calculation that we need to perform is the partial derivative of the Softmax function, which is a bit more complicated task than the derivative of the Categorical Cross-Entropy loss. Let's remind ourselves of the equation of the Softmax activation function and define the derivative:

$$S_{i,j} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \quad \rightarrow \quad \frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}}{\partial z_{i,k}}$$

Where $S_{i,j}$ denotes j-th Softmax's output of i-th sample, z — input array which is a list of input vectors (output vectors from the previous layer), $z_{i,j}$ — j-th Softmax's input of i-th sample, L — number of inputs, $z_{i,k}$ — k-th Softmax's input of i-th sample.

As we described in chapter 4, the Softmax function equals the exponentiated input divided by the sum of all exponentiated inputs. In other words, we need to exponentiate all of the values first, then divide each of them by the sum of all of them to perform the normalization. Each input to the Softmax impacts each of the outputs, and we need to calculate the partial derivative of each output with respect to each input. From the programming side of things, if we calculate the impact of one list on the other list, we'll receive a matrix of values as a result. That's exactly what we'll calculate here — we'll calculate the **Jacobian matrix** (which we'll explain later) of the vectors, which we'll dive deeper into soon.

To calculate this derivative, we need to first define the derivative of the division operation:

$$f(x) = \frac{g(x)}{h(x)} \quad \rightarrow \quad f'(x) = \frac{g'(x) \cdot h(x) - g(x) \cdot h'(x)}{[h(x)]^2}$$

In order to calculate the derivative of the division operation, we need to take the derivative of the numerator multiplied by the denominator, subtract the numerator multiplied by the derivative of the denominator from it, and then divide the result by the squared denominator.

We can now start solving the derivative:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}}{\partial z_{i,k}} =$$

Let's apply the derivative of the division operation:

$$= \frac{\frac{\partial}{\partial z_{i,k}} e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot \frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

At this step, we have two partial derivatives present in the equation. For the one on the right side of the numerator (right side of the subtraction operator):

$$\frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}}$$

We need to calculate the derivative of the sum of the constant,*e* (Euler's number), raised to power $z_{i,l}$ (where *l* denotes consecutive indices from *1* to the number of the Softmax outputs — *L*) with respect to the $z_{i,k}$. The derivative of the sum operation is the sum of derivatives, and the derivative of the constant *e* raised to power *n* (e^n) with respect to *n* equals e^n:

$$\frac{d}{dn} e^n = e^n \cdot \frac{d}{dn} n = e^n \cdot 1 = e^n$$

It is a special case when the derivative of an exponential function equals this exponential function itself, as its exponent is exactly what we are deriving with respect to, thus its derivative equals *1*. We also know that the range *1...L* contains *k* (*k* is one of the indices from this range) exactly once and then, in this case, the derivative is going to equal *e* to the power of the $z_{i,k}$ (as *j* equals *k*) and *0* otherwise (when *j* does not equal *k* as $z_{i,l}$ won't contain $z_{i,k}$ and will be treated as a constant — The derivative of the constant equals *0*):

$$\frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}} = \frac{\partial}{\partial z_{i,k}} e^{z_{i,1}} + \frac{\partial}{\partial z_{i,k}} e^{z_{i,2}} + \ldots + \frac{\partial}{\partial z_{i,k}} e^{z_{i,k}} + \ldots + \frac{\partial}{\partial z_{i,k}} e^{z_{i,L-1}} + \frac{\partial}{\partial z_{i,k}} e^{z_{i,L}}$$

$$= 0 + 0 + \ldots + e^{z_{i,k}} + \ldots + 0 + 0 = e^{z_{i,k}}$$

The derivative on the left side of the subtraction operator in the denominator is a slightly different case:

$$\frac{\partial}{\partial z_{i,k}} e^{z_{i,j}}$$

It does not contain the sum over all of the elements like the derivative we solved moments ago, so it can become either *0* if $j \neq k$ or *e* to the power of the $z_{i,j}$ if $j=k$. That means, starting from this step, we need to calculate the derivatives separately for both cases. Let's start with $j=k$.

In the case of $j=k$, the derivative on the left side is going to equal *e* to the power of the $z_{i,j}$ and the derivative on the right solves to the same value in both cases. Let's substitute them:

$$= \frac{e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

The numerator contains the constant *e* to the power of $z_{i,j}$ in both the minuend (the value we are subtracting from) and subtrahend (the value we are subtracting from the minuend) of the subtraction operation. Because of this, we can regroup the numerator to contain this value multiplied by the subtraction of their current multipliers. We can also write the denominator as a multiplication of the value instead of using the power of *2*:

$$= \frac{e^{z_{i,j}} \cdot (\sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,k}})}{\sum_{l=1}^{L} e^{z_{i,l}} \cdot \sum_{l=1}^{L} e^{z_{i,l}}} =$$

Then let's split the whole equation into 2 parts:

$$= \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot \frac{\sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}} =$$

We moved *e* from the numerator and the sum from the denominator to its own fraction, and the content of the parentheses in the numerator, and the other sum from the denominator as another fraction, both joined by the multiplication operation. Now we can further split the "right" fraction into two separate fractions:

$$= \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot (\frac{\sum_{l=1}^{L} e^{z_{i,l}}}{\sum_{l=1}^{L} e^{z_{i,l}}} - \frac{e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}}) =$$

In this case, as it's a subtraction operation, we separated both values from the numerator, dividing them both by the denominator and applying the subtraction operation between new fractions. If

we look closely, the "left" fraction turns into the Softmax function's equation, as well as the "right" one, with the middle fraction solving to *1* as the numerator and the denominator are the same values:

$$= S_{i,j} \cdot (1 - S_{i,k})$$

Note that the "left" Softmax function carries the $_j$ parameter, and the "right" one $_k$ — both came from their numerators, respectively.

Full solution:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}}{\partial z_{i,k}} = \frac{\frac{\partial}{\partial z_{i,k}} e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot \frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

$$\frac{e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} = \frac{e^{z_{i,j}} \cdot (\sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,k}})}{\sum_{l=1}^{L} e^{z_{i,l}} \cdot \sum_{l=1}^{L} e^{z_{i,l}}} =$$

$$\frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot \frac{\sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot (\frac{\sum_{l=1}^{L} e^{z_{i,l}}}{\sum_{l=1}^{L} e^{z_{i,l}}} - \frac{e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}}) =$$

$$S_{i,j} \cdot (1 - S_{i,k})$$

Now we have to go back and solve the derivative in the case of *j≠k*. In this case, the "left" derivative of the original equation solves to *0* as the whole expression is treated as a constant:

$$= \frac{0 \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

The difference is that now the whole subtrahend solves to *0*, leaving us with just the minuend in the numerator:

$$= \frac{-e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

Now, exactly like before, we can write the denominator as the multiplication of the values instead of using the power of *2*:

$$= \frac{-e^{z_{i,j}} \cdot e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}} \cdot \sum_{l=1}^{L} e^{z_{i,l}}} =$$

That lets us to split this fraction into 2 fractions, using the multiplication operation:

$$= -\frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot \frac{e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}} =$$

Now both fractions represent the Softmax function:

$$= -S_{i,j} \cdot S_{i,k}$$

Note that the left Softmax function carries the $_j$ parameter, and the "right" one has $_k$ — both came from their numerators, respectively.

Full solution:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}}{\partial z_{i,k}} = \frac{\frac{\partial}{\partial z_{i,k}} e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot \frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

$$\frac{0 \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} = \frac{-e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} = \frac{-e^{z_{i,j}} \cdot e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}} \cdot \sum_{l=1}^{L} e^{z_{i,l}}} =$$

$$-\frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot \frac{e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}} = -S_{i,j} \cdot S_{i,k}$$

As a summary, the solution of the derivative of the Softmax function with respect to its inputs is:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \begin{cases} S_{i,j} \cdot (1 - S_{i,k}) & j = k \\ -S_{i,j} \cdot S_{i,k} & j \neq k \end{cases}$$

That's not the end of the calculation that we can perform here. When left in this form, we'll have 2 separate equations to code and use in different cases, which isn't very convenient for the speed of calculations. We can, however, further morph the result of the second case of the derivative:

$$-S_{i,j} \cdot S_{i,k} = S_{i,j} \cdot (-S_{i,k}) = S_{i,j} \cdot (0 - S_{i,k})$$

In the first step, we moved the second Softmax along the minus sign into the brackets so we can add a zero inside of them and right before this value. That does not change the solution, but now:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \begin{cases} S_{i,j} \cdot (1 - S_{i,k}) & j = k \\ S_{i,j} \cdot (0 - S_{i,k}) & j \neq k \end{cases}$$

Both solutions look very similar, they differ only in a single value. Conveniently, there exists **Kronecker delta** function (which we'll explain soon) whose equation is:

$$\delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

We can apply it here, simplifying our equation further to:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = S_{i,j} \cdot (\delta_{j,k} - S_{i,k})$$

That's the final math solution to the derivative of the Softmax function's outputs with respect to each of its inputs. To make it a little bit easier to implement in Python using NumPy, let's transform the equation for the last time:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = S_{i,j} \cdot (\delta_{j,k} - S_{i,k}) = S_{i,j}\delta_{j,k} - S_{i,j}S_{i,k}$$

We basically multiplied $S_{i,j}$ by both sides of the subtraction operation from the parentheses.

Softmax activation derivative code implementation

This lets us code the solution using just two NumPy functions, which we'll explain now step by step:

Let's make up a single sample:

```
ĽIJÌĻİĞŇçIJŁĻĴŁĻ ł äflgìd flgĝd flgğā
```

And shape it as a list of samples:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ĽIJÌĻİĞŇçIJŁĻĴŁĻ ł ĮĴgĞĹĹĞÑâĽIJÌĻİĞŇçIJŁĻĴŁĻãgĹĦĽĨĞĴĦâhĝd ĝã
ĴĹĨĮĻâĽIJÌĻİĞŇçIJŁĻĴŁĻã


ńńń
ĞĹĹĞÑâääflgìād
       äflgĝād
       äflgğāāā
```

The left side of the equation is Softmax's output multiplied by the Kronecker delta. The Kronecker delta equals *1* when both inputs are equal, and *0* otherwise. If we visualize this as an array, we'll have an array of zeros with ones on the diagonal — you might remember that we already have implemented such a solution using the `ĮĴgĦÑĦ` method:

```
ĴĹĨĮĻâĮĴgĦÑĦâĽIJÌĻİĞŇçIJŁĻĴŁĻgĽÎĞĴĦäflāãã


ńńń
ĞĹĹĞÑâäägĝgd flgd flgād
       äflgd ĝgd flgād
       äflgd flgd ĝgāãã
```

Now we'll do the multiplication of both of the values from the equation part:

```
ĴĹĨĮĻâĽIJÌĻİĞŇçIJŁĻĴŁĻ ǻ ĮĴgĦÑĦâĽIJÌĻİĞŇçIJŁĻĴŁĻgĽÎĞĴĦäflāãã


ńńń
ĞĹĹĞÑâäägìd flg d flg ād
       äflg d flgĝd flg ād
       äflg d flg d flgğāãã
```

It turns out that we can gain some speed by replacing this by the `ĮĴgĤĨĞÍÌĬĞĻ` method call, which computes the same solution — the diagflat method creates an array using an input vector as the diagonal:

```
ĴĹĨĮĻâĮĴgĤĨĞÍÌĬĞĻâĽIJÌĻİĞŇçIJŁĻĴŁĻãã


ńńń
ĞĹĹĞÑâäägìd flg d flg ād
       äflg d flgĝd flg ād
       äflg d flg d flgğāãã
```

The other part of the equation is $S_{i,j}S_{i,k}$ — the multiplication of the Softmax outputs, iterating over the $_j$ and $_k$ indices respectively. Since, for each sample (the $_i$ index), we'll have to multiply the values from the Softmax function's output (in all of the combinations), we can use the dot product operation. For this, we'll just have to transpose the second argument to get its row vector form (as described in chapter 2):

```
ĴĹĨĮĻâĮĴgĤIJĻâĽIJÌĻİĞŇçIJŁĻĴŁĻd ĽIJÌĻİĞŇçIJŁĻĴŁĻgTãã


ńńń
ĞĹĹĞÑâäägǵîd flgflìd flgĝǵād
       äflgflìd flgflĝd flgflğād
       äflgĝǵd flgflğd flgflǵāãã
```

Finally, we can perform the subtraction of both arrays (following the equation):

```
ĴĹĨĮĻâĮĴgĤĨĞÍÌĬĞĻâĽIJÌĻİĞŇçIJŁĻĴŁĻã h
      ĮĴgĤIJĻâĽIJÌĻİĞŇçIJŁĻĴŁĻd ĽIJÌĻİĞŇçIJŁĻĴŁĻgTãã


ńńń
ĞĹĹĞÑâää flgğĝd hflgflìd hflgĝǵād
       ähflgflìd  flgflîd hflgflğād
       ähflgĝǵd hflgflğd  flgĝħāāã
```

The matrix result of the equation and the array solution provided by the code is called the **Jacobian matrix**. In our case, the Jacobian matrix is an array of partial derivatives in all of the combinations of both input vectors. Remember, we are calculating the partial derivatives of every output of the Softmax function with respect to each input separately. We do this because each input influences each output due to the normalization process, which takes the sum of all the exponentiated inputs. The result of this operation, performed on a batch of samples, is a list of the Jacobian matrices, which effectively forms a 3D matrix — you can visualize it as a column whose levels are Jacobian matrices being the sample-wise gradient of the Softmax function.

This raises a question — if sample-wise gradients are the Jacobian matrices, how do we perform the chain rule with the gradient back-propagated from the loss function, since it's a vector for each sample? Also, what do we do with the fact that the previous layer, which is the Dense layer, will expect the gradients to be a 2D array? Currently, we have a 3D array of the partial derivatives — a list of the Jacobian matrices. The derivative of the Softmax function with respect to any of its inputs returns a vector of partial derivatives (a row from the Jacobian matrix), as this input influences all the outputs, thus also influencing the partial derivative for each of them. We need to sum the values from these vectors so that each of the inputs for each of the samples will return a single partial derivative value instead. Because each input influences all of the outputs, the returned vector of the partial derivatives has to be summed up for the final partial derivative with respect to this input. We can perform this operation on each of the Jacobian matrices directly, applying the chain rule at the same time (applying the gradient from the loss function) using `ĮĴgĤIJĻâã` — For each sample, it'll take the row from the Jacobian matrix and multiply it by the corresponding value from the loss function's gradient. As a result, the dot product of each of these vectors and values will return a singular value, forming a vector of the partial derivatives sample-wise and a 2D array (a list of the resulting vectors) batch-wise.

Let’s code the solution:

```
ƒ SIJÌĻİĞŇ ĞĢĻÎĿĞĻÎIJĮ
ĢĬĞĽĽ AĢĻÎĿĞĻÎIJĮçSIJÌĻİĞŇƒ
    ggg
    ƒ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĿĞĬŁĦĽãƒ

        ƒ CĹĦĞĻĦ ŁĮĬĮĬĻĬĞĬĬŅĦĤ ĞĹĹĞŇ
        ĽĦĬÌgĤĬĮĴŁĻĽ ł ĮĴgĦİĴĻÑçĬĬĪĦâĤĿĞĬŁĦĽã

        ƒ EĮŁİĦĹĞĻĦ IJŁĻĴŁĻĽ ĞĮĤ ÍĹĞĤĬĦĮĻĽ
        ÌIJĹ ĬĮĤĦŇd âĽĬĮÍĬĦçIJŁĻĴŁĻd ĽĬĮÍĬĦçĤĿĞĬŁĦĽã ĬĮ ċ
            ĦĮŁİĦĹĞĻĦâŅĬĴâĽĦĬÌgIJŁĻĴŁĻd ĤĿĞĬŁĦĽããƒ
        ƒ FĬĞĻĻĦĮ IJŁĻĴŁĻ ĞĹĹĞŇ
        ĽĬĮÍĬĦçIJŁĻĴŁĻ ł ĽĬĮÍĬĦçIJŁĻĴŁĻgĹĦĽĬĞĴĦâhĝd ĝã
        ƒ CĞĬĢŁĬĞĻĦ JĞĢIJĠĬĞĮ İĞĻĹĬŇ IJÌ ĻĬĦ IJŁĻĴŁĻ ĞĮĤ
        ÏĞĢIJĠĬĞĮçİĞĻĹĬŇ ł ĮĴgĤĬĞÍÌĬĞĻâĽĬĮÍĬĦçIJŁĻĴŁĻã h ċ
                            ĮĴgĤIJĻâĽĬĮÍĬĦçIJŁĻĴŁĻd ĽĬĮÍĬĦçIJŁĻĴŁĻgTã
        ƒ CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦhŃĬĽĦ ÍĹĞĤĬĦĮĻ
        ƒ ĞĮĤ ĞĤĤ ĬĻ ĻIJ ĻĬĦ ĞĹĹĞŇ IJÌ ĽĞİĴĬĦ ÍĹĞĤĬĦĮĻĽ
        ĽĦĬÌgĤĬĮĴŁĻĽäĬĮĤĦŇã ł ĮĴgĤIJĻâÏĞĢIJĠĬĞĮçİĞĻĹĬŇd
                                        ĽĬĮÍĬĦçĤĿĞĬŁĦĽã
```

First, we created an empty array (which will become the resulting gradient array) with the same shape as the gradients that we’re receiving to apply the chain rule. The `ĮĴgĦİĴĻÑçĬĬĪĦ` method creates an empty and uninitialized array. Uninitialized means that we can expect it to contain meaningless values, but we’ll set all of them shortly anyway, so there’s no need for initialization (for example, with zeros using `ĮĴgŅĦĹIJĽâã` instead). In the next step, we’re going to iterate sample-wise over pairs of the outputs and gradients, calculating the partial derivatives as described earlier and calculating the final product (applying the chain rule) of the Jacobian matrix and gradient vector (from the passed-in gradient array), storing the resulting vector as a row in the dinput array. We’re going to store each vector in each row while iterating, forming the output array.

Common Categorical Cross-Entropy loss and Softmax activation derivative

At the moment, we have calculated the partial derivatives of the Categorical Cross-Entropy loss and Softmax activation functions, and we can finally use them, but there is still one more step that we can perform to speed the calculations up. Different books and tutorials usually mention the derivative of the loss function with respect to the Softmax inputs, or even weight and biases of the output layer directly and don't go into the details of the partial derivatives of these functions separately. This is partially because the derivatives of both functions combine to solve a simple equation — the whole code implementation is simpler and faster to execute. When we look at our current code, we perform multiple operations to calculate the gradients and even include a loop in the backward step of the activation function.

Let's apply the chain rule to calculate the partial derivative of the Categorical Cross-Entropy loss function with respect to the Softmax function inputs. First, let's define this derivative by applying the chain rule:

$$\frac{\partial L_i}{\partial z_{i,k}} = \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial S_{i,j}}{\partial z_{i,k}} =$$

This partial derivative equals the partial derivative of the loss function with respect to its inputs, multiplied (using the chain rule) by the partial derivative of the activation function with respect to its inputs. Now we need to systematize semantics — we know that the inputs to the loss function, $y\text{-}hat_{i,j}$, are the outputs of the activation function, $S_{i,j}$:

$$\hat{y}_{i,j} = S_{i,j}$$

That means that we can update the equation to the form of:

$$= \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} =$$

Now we can substitute the equation for the partial derivative of the Categorical Cross-Entropy function, but, since we are calculating the partial derivative with respect to the Softmax inputs, we'll use the one containing the sum operator over all of the outputs — it will soon become clear why. The derivative:

$$\frac{\partial L_i}{\partial \hat{y}_{i,j}} = -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}}$$

After substitution to the combined derivative's equation:

$$= -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} =$$

Now, as we calculated before, the partial derivative of the Softmax activation, before applying Kronecker delta to it:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \begin{cases} S_{i,j} \cdot (1 - S_{i,k}) & j = k \\ -S_{i,j} \cdot S_{i,k} & j \neq k \end{cases}$$

Let's actually do the substitution of the $S_{i,j}$ with *y-hat*$_{i,j}$ here as well:

$$\frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = \begin{cases} \hat{y}_{i,j} \cdot (1 - \hat{y}_{i,k}) & j = k \\ -\hat{y}_{i,j} \cdot \hat{y}_{i,k} & j \neq k \end{cases}$$

The solution is different depending on if $j=k$ or $j \neq k$. To handle for this situation, we have to split the current partial derivative following these cases — when they both match and when they do not:

$$-\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \frac{\partial \hat{y}_{i,k}}{\partial z_{i,k}} - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}}$$

For the $j \neq k$ case, we just updated the sum operator to exclude k and that's the only change:

$$-\sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}}$$

For the $j=k$ case, we do not need the sum operator as it will sum only one element, of index k. For

the same reason, we also replace *j*indices with *k*:

$$-\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \frac{\partial \hat{y}_{i,k}}{\partial z_{i,k}}$$

Back to the main equation:

$$= -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \frac{\partial \hat{y}_{i,k}}{\partial z_{i,k}} - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} =$$

Now we can substitute the partial derivatives of the activation function for both cases with the newly-defined solutions:

$$= -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \hat{y}_{i,k} \cdot (1 - \hat{y}_{i,k}) - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} (-\hat{y}_{i,j} \hat{y}_{i,k}) =$$

We can cancel out the *y-hat*$_{i,k}$ from both sides of the subtraction in the equation — both contain it as part of the multiplication operations and in their denominators. Then on the "right" side of the equation, we can replace 2 minus signs with the plus one and remove the parentheses:

$$= -y_{i,k} \cdot (1 - \hat{y}_{i,k}) + \sum_{j \neq k} y_{i,j} \hat{y}_{i,k} =$$

Now let's multiply the -$y_{i,k}$ with the content of the parentheses on the "left" side of the equation:

$$= -y_{i,k} + y_{i,k} \hat{y}_{i,k} + \sum_{j \neq k} y_{i,j} \hat{y}_{i,k} =$$

Now let's look at the sum operation — it adds up $y_{i,j}$*y-hat*$_{i,k}$ over all possible values of index *j* except for when it equals *k*. Then, on the left of this part of the equation, we have $y_{i,k}$*y-hat*$_{i,k}$, which contains $y_{i,k}$ — the exact element that is excluded from the sum. We can then join both expressions:

$$= -y_{i,k} + \sum_{j} y_{i,j} \hat{y}_{i,k} =$$

Now the sum operator iterates over all of the possible values of *j* and, since we know that $y_{i,j}$ for each *i* is the one-hot encoded vector of ground-truth values, the sum of all of its elements equals *1*. In other words, following the earlier explanation in this chapter — this sum will multiply *0* by the *y-hat*$_{i,k}$ except for a single situation, the true label, where it'll multiply *1* by this value. We can then simplify it further to:

Full solution:

$$\frac{\partial L_i}{\partial z_{i,k}} = \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} =$$

$$= -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \frac{\partial \hat{y}_{i,k}}{\partial z_{i,k}} - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \hat{y}_{i,k} \cdot (1 - \hat{y}_{i,k}) - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} (-\hat{y}_{i,j} \hat{y}_{i,k}) =$$

$$= -y_{i,k} \cdot (1 - \hat{y}_{i,k}) + \sum_{j \neq k} y_{i,j} \hat{y}_{i,k} = -y_{i,k} + y_{i,k} \hat{y}_{i,k} + \sum_{j \neq k} y_{i,j} \hat{y}_{i,k} =$$

$$= -y_{i,k} + \sum_j y_{i,j} \hat{y}_{i,k} = -y_{i,k} + \hat{y}_{i,k} = \hat{y}_{i,k} - y_{i,k}$$

As we can see, when we apply the chain rule to both partial derivatives, the whole equation simplifies significantly to the subtraction of the predicted and ground truth values. It is also multiple times faster to compute.

Common Categorical Cross-Entropy loss and Softmax activation derivative - code implementation

To code this solution, nothing in the forward pass changes — we still need to perform it on the activation function to receive the outputs and then on the loss function to calculate the loss value. For backpropagation, we'll create the backward step containing the implementation of the new equation, which calculates the combined gradient of the loss and activation functions. We'll code the solution as a separate class, which initializes both the Softmax activation and the Categorical Cross-Entropy objects, calling their forward methods respectively during the forward pass. Then the new backward pass is going to contain the new code:

```
# SIJÌĻİĞŇ ĢĬĞĽĽĨÌĨĦĹ h ĢIJİĠĨĮĦĤ SIJÌĻİĞŇ ĞĢĻĨĿĞĻĨIJĮ
# ĞĮĤ ĢĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ ÌIJĹ ÌĞĽĻĦĹ ĠĞĢĪŃĞĹĤ ĽĻĦĴ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇçLIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâãf

    # CĹĦĞĻĦĽ ĞĢĻĨĿĞĻĨIJĮ ĞĮĤ ĬIJĹĽ ÌŁĮĢĻĨIJĮ IJĠÏĦĢĻĽ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ĽĦĬÌgĞĢĻĨĿĞĻĨIJĮ ł AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇâã
        ĽĦĬÌgĬIJĹĽ ł LIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâã

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ÑçĻĹŁĦãf
        # OŁĻĴŁĻ ĬĞÑĦĹďĽ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĞĢĻĨĿĞĻĨIJĮgÌIJĹŃĞĹĤâĨĮĴŁĻĽã
        # SĦĻ ĻĤĦ IJŁĻĴŁĻ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĽĦĬÌgĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻ
        # CĞĬĢŁĬĞĻĦ ĞĮĤ ĹĦĻŁĹĮ ĬIJĹĽ ĿĞĬŁĦ
        ĹĦĻŁĹĮ ĽĦĬÌgĬIJĹĽgĢĞĬĢŁĬĞĻĦâĽĦĬÌgIJŁĻĴŁĻd ÑçĻĹŁĦã

    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĿĞĬŁĦĽd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâĤĿĞĬŁĦĽã
```

```
# IÌ ĬĞĠĦĬĽ ĞĹĦ IJĮĦhÎIJĻ ĦĮĢIJÂĦĤd
# ĻŁĹĮ ĻÎĦİ ĨĮĻIJ ĤĨĽĢĹĦĻĦ ĿĞĬŁĦĽ
ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
    ÑçĻĹŁĦ ł ĮĴgĞĹÍİĞŇâÑçĻĹŁĦd ĞŇĨĽłĝã

# CIJĴÑ ĽIJ ŃĦ ĢĞĮ ĽĞÌĦĬÑ İIJÂĨÌÑ
ĽĦĬÌgĤĨĮĴŁĻĽ ł ÂĿĞĬŁĦĽgĢIJĴÑâã
# CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
ĽĦĬÌgĤĨĮĴŁĻĽäĹĞĮÍĦâĽĞİĴĬĦĽãd ÑçĻĹŁĦã hł ĝ
# NIJĹİĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽĞİĴĬĦĽ
```

To implement the solution $y\text{-}hat_{i,k}\text{-}y_{i,k}$, instead of performing the subtraction of the full arrays, we're taking advantage of the fact that the *y* being `ÑçĻĹŁĦ` in the code consists of one-hot encoded vectors, which means that, for each sample, there is only a singular value of *1* in these vectors and the remaining positions are filled with zeros.

This means that we can use NumPy to index the prediction array with the sample number and its true value index, subtracting *1* from these values. This operation requires discrete true labels instead of one-hot encoded ones, thus the additional code that performs the transformation if needed — If the number of dimensions in the ground-truth array equals *2*, it means that it's a matrix consisting of one-hot encoded vectors. We can use `ĮĴgĞĹÍİĞŇâã`, which returns the index of the maximum value (index for *1* in this case), but we need to perform this operation sample-wise to get a vector of indices:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ÑçĻĹŁĦ ł ĮĴgĞĹĹĞŇâäägdfldflādäfldfldĝādäfldĝdflāāã
ĮĴgĞĹÍİĞŇâÑçĻĹŁĦã


ńńń
fl


ĴĹĨĮĻâĮĴgĞĹÍİĞŇâÑçĻĹŁĦd ĞŇĨĽłĝãã


ńńń
äfld ğd ĝā
```

For the last step, we normalize the gradient in exactly the same way and for the same reason as described along with the Categorical Cross-Entropy gradient normalization.

Let’s summarize the code for each of the classes that we have updated:

```
# SIJÌĻİĞŇ ĞĢĻÎĹĞĻÎIJĮ
ĢĬĞĹĹ AĢĻÎĹĞĻÎIJĮçSIJÌĻİĞŇf

    # FIJĹŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ĨĮĴŁĻĹãf
        # RĦİĦİĠĦĹ ĨĮĴŁĻ ĹĞĬŁĦĹ
        ĹĦĬÌgĨĮĴŁĻĹ ł ĨĮĴŁĻĹ

        # GĦĻ ŁĮĮIJĹİĞĬĨŅĦĤ ĴĹIJĠĞĠĨĬĨĻĨĦĹ
        ĦŇĴçĹĞĬŁĦĹ ł ĮĴgĦŇĴâĨĮĴŁĻĹ h ĮĴgİĞŇâĨĮĴŁĻĹd ĞŇĨĹłĝd
                                                ĪĦĦĴĤĨİĹłTĹŁĦãã
        # NIJĹİĞĬĨŅĦ ĻĤĦİ ÌIJĹ ĦĞĢĤ ĹĞİĴĬĦ
        ĴĹIJĠĞĠĨĬĨĻĨĦĹ ł ĦŇĴçĹĞĬŁĦĹ w ĮĴgĹŁİâĦŇĴçĹĞĬŁĦĹd ĞŇĨĹłĝd
                                                ĪĦĦĴĤĨİĹłTĹŁĦã

        ĹĦĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĨĬĨĻĨĦĹ

    # BĞĢĨŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĹĦĬÌd ĤĹĞĬŁĦĹãf

        # CĹĦĞĻĦ ŁĮĨĮĨĻĨĞĬĨŅĦĤ ĞĹĹĞŇ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĮĴgĦİĴĻŇçĬĨĨĦâĤĹĞĬŁĦĹã

        # EĮŁİĦĹĞĻĦ IJŁĻĴŁĻĹ ĞĮĤ ÍĹĞĤĨĦĮĻĹ
        ÌIJĹ ĨĮĤĦŇd âĹĨĮÍĬĦçIJŁĻĴŁĻd ĹĨĮÍĬĦçĤĹĞĬŁĦĹã ĨĮ ċ
                ĦĮŁİĦĹĞĻĦâŅĨĴâĹĦĬÌgIJŁĻĴŁĻd ĤĹĞĬŁĦĹããf
            # FĬĞĻĻĦĮ IJŁĻĴŁĻ ĞĹĹĞŇ
            ĹĨĮÍĬĦçIJŁĻĴŁĻ ł ĹĨĮÍĬĦçIJŁĻĴŁĻgĹĦĹĬĞĴĦâhĝd ĝã
            # CĞĬĢŁĬĞĻĦ JĞĢIJĠĨĞĮ İĞĻĹĨŇ IJÌ ĻĤĦ IJŁĻĴŁĻ ĞĮĤ
            ÏĞĢIJĠĨĞĮçİĞĻĹĨŇ ł ĮĴgĤĨĞÍÌĬĞĻâĹĨĮÍĬĦçIJŁĻĴŁĻã h ċ
                                ĮĴgĤIJĻâĹĨĮÍĬĦçIJŁĻĴŁĻd ĹĨĮÍĬĦçIJŁĻĴŁĻgTã
            # CĞĬĢŁĬĞĻĦ ĹĞİĴĬĦhŃĨĹĦ ÍĹĞĤĨĦĮĻ
            # ĞĮĤ ĞĤĤ ĨĻ ĻIJ ĻĤĦ ĞĹĹĞŇ IJÌ ĹĞİĴĬĦ ÍĹĞĤĨĦĮĻĹ
            ĹĦĬÌgĤĨĮĴŁĻĹäĨĮĤĦŇã ł ĮĴgĤIJĻâÏĞĢIJĠĨĞĮçİĞĻĹĨŇd
                                        ĹĨĮÍĬĦçĤĹĞĬŁĦĹã


# CĹIJĹĹhĦĮĻĹIJĴŇ ĬIJĹĹ
ĢĬĞĹĹ LIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴŇâLIJĹĹãf

    # FIJĹŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ŇçĴĹĦĤd ŇçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĹĞİĴĬĦĹ ĨĮ Ğ ĠĞĻĢĤ
        ĹĞİĴĬĦĹ ł ĬĦĮâŇçĴĹĦĤã
```

```
        fi CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ĤĨŁĨĹĨIJĮ ĠÑ fl
        fi CĬĨĴ ĠIJĻĨ ĹĨĤĦĹ ĻIJ ĮIJĻ ĤĹĞÍ ĬĦĞĮ ĻIJŃĞĹĤĹ ĞĮÑ ŁĞĬŁĦ
        ÑçĴĹĦĤçĢĬĨĴĴĦĤ ł ĮĴgĢĬĨĴâÑçĴĹĦĤd ĝĦhìd ĝ h ĝĦhìã

        fi PĹIJĠĞĠĨĬĨĻĨĦĹ ÌIJĹ ĻĞĹÍĦĻ ŁĞĬŁĦĹ h
        fi IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĹ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĹĤĞĴĦã łł ĝf
            ĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĹ ł ÑçĴĹĦĤçĢĬĨĴĴĦĤä
                ĹĞĮÍĦâĹĞĬĴĬĦĹãd
                ÑçĻĹŁĦ
            ã

        fi MĞĹĨ ŁĞĬŁĦĹ h IJĮĬÑ ÌIJĹ IJĮĦhĤIJĻ ĦĮĢIJĤĦĤ ĬĞĠĦĬĹ
        ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĹĤĞĴĦã łł ğf
            ĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĹ ł ĮĴgĹŁİâ
                ÑçĴĹĦĤçĢĬĨĴĴĦĤ ấ ÑçĻĹŁĦd
                ĞÑĨĹłĝ
            ã

        fi LIJĹĹĦĹ
        ĮĦÍĞĻĨŁĦçĬIJÍçĬĨĨĦĬĨĨIJIJĤĹ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĹã
        ĹĦĻŁĹĮ ĮĦÍĞĻĨŁĦçĬIJÍçĬĨĨĦĬĨĨIJIJĤĹ

    fi BĞĢĨŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĹĦĬÌd ĤŁĞĬŁĦĹd ÑçĻĹŁĦãf

        fi NŁİĠĦĹ IJÌ ĹĞİĴĬĦĹ
        ĹĞİĴĬĦĹ ł ĬĦĮâĤŁĞĬŁĦĹã
        fi NŁİĠĦĹ IJÌ ĬĞĠĦĬĹ ĨĮ ĦŁĦĹÑ ĹĞİĴĬĦ
        fi WĦďĬĬ ŁĹĦ ĻĤĦ ÌĨĹĹĻ ĹĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦİ
        ĬĞĠĦĬĹ ł ĬĦĮâĤŁĞĬŁĦĹäflãã

        fi IÌ ĬĞĠĦĬĹ ĞĹĦ ĹĴĞĹĹĦd ĻŁĹĮ ĻĤĦİ ĨĮĻIJ IJĮĦhĤIJĻ ŁĦĢĻIJĹ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĹĤĞĴĦã łł ĝf
            ÑçĻĹŁĦ ł ĮĴgĦÑĦâĬĞĠĦĬĹãäÑçĻĹŁĦã

        fi CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł hÑçĻĹŁĦ w ĤŁĞĬŁĦĹ
        fi NIJĹİĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĹĦĬÌgĤĨĮĴŁĻĹ w ĹĞİĴĬĦĹ


fi SIJÌĻİĞŃ ĢĬĞĹĹĨÌÌĨĦĹ h ĢIJİĠĨĮĦĤ SIJÌĻİĞŃ ĞĢĻĨŁĞĻĨIJĮ
fi ĞĮĤ ĢĹIJĹĹhĦĮĻĹIJĴÑ ĬIJĹĹ ÌIJĹ ÌĞĹĻĦĹ ĠĞĢĨŃĞĹĤ ĹĻĦĴ
ĢĬĞĹĹ AĢĻĨŁĞĻĨIJĮçSIJÌĻİĞŃçLIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑâãf

    fi CĹĦĞĻĦĹ ĞĢĻĨŁĞĻĨIJĮ ĞĮĤ ĬIJĹĹ ÌŁĮĢĻĨIJĮ IJĠÏĦĢĻĹ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌãf
        ĹĦĬÌgĞĢĻĨŁĞĻĨIJĮ ł AĢĻĨŁĞĻĨIJĮçSIJÌĻİĞŃâã
        ĹĦĬÌgĬIJĹĹ ł LIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑâã
```

```
ƒ FIJĹŃĞĹÎ ĴĞĽĽ
Ĥ̀Ĥ̀ ÌIJĹŃĞĹÎâĽĤĬÌd ĨĮĴŁĻĽd ÑçĻĹŁĤãf
    ƒ OŁĻĴŁĻ ĬĞÑĤĹďĽ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    ĽĤĬÌgĞĢĻĨĿĞĻĨIJĮgÌIJĹŃĞĹÎâĨĮĴŁĻĽã
    ƒ SĤĻ ĻÎĤ IJŁĻĴŁĻ
    ĽĤĬÌgIJŁĻĴŁĻ ł ĽĤĬÌgĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻ
    ƒ CĞĬĢŁĬĞĻĤ ĞĮÎ ĹĤĻŁĹĮ ĬIJĽĽ ĿĞĬŁĤ
    ĹĤĻŁĹĮ ĽĤĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĤâĽĤĬÌgIJŁĻĴŁĻd ÑçĻĹŁĤã

ƒ BĞĢĪŃĞĹÎ ĴĞĽĽ
Ĥ̀Ĥ̀ ĠĞĢĪŃĞĹÎâĽĤĬÌd ĤĿĞĬŁĤĽd ÑçĻĹŁĤãf

    ƒ NŁİĠĤĹ IJÌ ĽĞİĴĬĤĽ
    ĽĞİĴĬĤĽ ł ĬĤĮâĤĿĞĬŁĤĽã

    ƒ IÌ ĬĞĠĤĬĽ ĞĹĤ IJĮĤhÎIJĻ ĤĮĢIJÎĤĤd
    ƒ ĻŁĹĮ ĻÎĤİ ĨĮĻIJ ÎĨĽĢĹĤĻĤ ĿĞĬŁĤĽ
    ĨÌ ĬĤĮâÑçĻĹŁĤgĽÎĞĴĤã łł ğf
        ÑçĻĹŁĤ ł ĮĴgĞĹÍİĞÑâÑçĻĹŁĤd ĞÑĨĽłĝã

    ƒ CIJĴÑ ĽIJ ŃĤ ĢĞĮ ĽĞÌĤĬÑ İIJÎĨÌÑ
    ĽĤĬÌgÎĨĮĴŁĻĽ ł ĤĿĞĬŁĤĽgĢIJĴÑâã
    ƒ CĞĬĢŁĬĞĻĤ ÍĹĞÎĨĤĮĻ
    ĽĤĬÌgÎĨĮĴŁĻĽäĹĞĮÍĤâĽĞİĴĬĤĽãd ÑçĻĹŁĤã hł ĝ
    ƒ NIJĹİĞĬĨŅĤ ÍĹĞÎĨĤĮĻ
    ĽĤĬÌgÎĨĮĴŁĻĽ ł ĽĤĬÌgÎĨĮĴŁĻĽ w ĽĞİĴĬĤĽ
```

We can now test if the combined backward step returns the same values compared to when we backpropagate gradients through both of the functions separately. For this example, let's make up an output of the Softmax function and some target values. Next, let's backpropagate them using both solutions:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ

ĮĮÌĽgĨĮĨĻâã

ĽIJÌĻİĞÑçIJŁĻĴŁĻĽ ł ĮĴgĞĹĹĞÑâääflgìd flgĝd flgğād
                              äflgĝd flgĥd flgǵād
                              äflgflğd flgîd flgflíāāã

ĢĬĞĽĽçĻĞĹÍĤĻĽ ł ĮĴgĞĹĹĞÑâäfld ĝd ĝāã

ĽIJÌĻİĞÑçĬIJĽĽ ł AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞÑçLIJĽĽçCĞĻĤÍIJĹĨĢĞĬCĹIJĽĽĤĮĻĹIJĴÑâã
ĽIJÌĻİĞÑçĬIJĽĽgĠĞĢĪŃĞĹÎâĽIJÌĻİĞÑçIJŁĻĴŁĻĽd ĢĬĞĽĽçĻĞĹÍĤĻĽã
ĤĿĞĬŁĤĽĝ ł ĽIJÌĻİĞÑçĬIJĽĽgĤĨĮĴŁĻĽ

ĞĢĻĨĿĞĻĨIJĮ ł AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞÑâã
ĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻ ł ĽIJÌĻİĞÑçIJŁĻĴŁĻĽ
ĬIJĽĽ ł LIJĽĽçCĞĻĤÍIJĹĨĢĞĬCĹIJĽĽĤĮĻĹIJĴÑâã
ĬIJĽĽgĠĞĢĪŃĞĹÎâĽIJÌĻİĞÑçIJŁĻĴŁĻĽd ĢĬĞĽĽçĻĞĹÍĤĻĽã
ĞĢĻĨĿĞĻĨIJĮgĠĞĢĪŃĞĹÎâĬIJĽĽgĤĨĮĴŁĻĽã
ĤĿĞĬŁĤĽğ ł ĞĢĻĨĿĞĻĨIJĮgĤĨĮĴŁĻĽ
```

```
ĴĹĨĮĻâďGĹĞĤĨĦĮĻĽf ĢIJĠĨĮĦĤ ĬIJĽĽ ĞĮĤ ĞĢĻĨĿĞĻĨIJĮfďã
ĴĹĨĮĻâĤĿĞĬŁĦĽĝã
ĴĹĨĮĻâďGĹĞĤĨĦĮĻĽf ĽĦĴĞĹĞĻĦ ĬIJĽĽ ĞĮĤ ĞĢĻĨĿĞĻĨIJĮfďã
ĴĹĨĮĻâĤĿĞĬŁĦĽĝã


ńńń
GĹĞĤĨĦĮĻĽf ĢIJĠĨĮĦĤ ĬIJĽĽ ĞĮĤ ĞĢĻĨĿĞĻĨIJĮf
äähflgĝ          flgflġġġġġġ  flgflħħħħħħìā
 ä flgflġġġġġġġ hflgĝħħħħħħì  flgĝġġġġġġġā
 ä flgflflħħħħħì hflgflġġġġġġ  flgflğħħħħħìāā
GĹĞĤĨĦĮĻĽf ĽĦĴĞĹĞĻĦ ĬIJĽĽ ĞĮĤ ĞĢĻĨĿĞĻĨIJĮf
äähflgflîîîîîîî  flgflġġġġġġġ  flgflħħħħħħìā
 ä flgflġġġġġġġ hflgĝħħħħħħì  flgĝġġġġġġġā
 ä flgflflħħħħħì hflgflġġġġġġ  flgflğħħħħħìāā
```

The results are the same. The small difference between values in both arrays results from the precision of floating-point values in raw Python and NumPy. To answer the question of how many times faster this solution is, we can take advantage of Python's timeit module, running both solutions multiple times and combining the execution times. A full description of the timeit module and the code used here is outside of the scope of this book, but we include this code purely to show the speed deltas:

```
ĨĬĴIJĹĻ ĮŁĬĴÑ ĞĽ ĮĴ
ÌĹIJĬ ĻĨĬĦĨĻ ĨĬĴIJĹĻ ĻĨĬĦĨĻ
ĨĬĴIJĹĻ ĮĮÌĽ

ĮĮÌĽgĨĮĨĻâã

ĽIJÌĻĬĞÑçIJŁĻĴŁĻĽ ł ĮĴgĞĹĹĞÑâääflgìd flgĝd flgğād
                              äflgĝd flghd flgġād
                              äflgflğd flgîd flgflíāāã

ĢĬĞĽĽçĻĞĹÍĦĻĽ ł ĮĴgĞĹĹĞÑâäfld ĝd ĝāã


ÂĦÌ Ìĝâãf
    ĽIJÌĻĬĞÑçĬIJĽĽ ł AĢĻĨĿĞĻĨIJĮçSIJÌĻĬĞÑçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã
    ĽIJÌĻĬĞÑçĬIJĽĽgĠĞĢĪŃĞĹÂâĽIJÌĻĬĞÑçIJŁĻĴŁĻĽd ĢĬĞĽĽçĻĞĹÍĦĻĽã
    ÂĿĞĬŁĦĽĝ ł ĽIJÌĻĬĞÑçĬIJĽĽgÂĨĮĴŁĻĽ


ÂĦÌ Ìğâãf
    ĞĢĻĨĿĞĻĨIJĮ ł AĢĻĨĿĞĻĨIJĮçSIJÌĻĬĞÑâã
    ĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻ ł ĽIJÌĻĬĞÑçIJŁĻĴŁĻĽ
    ĬIJĽĽ ł LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã
    ĬIJĽĽgĠĞĢĪŃĞĹÂâĽIJÌĻĬĞÑçIJŁĻĴŁĻĽd ĢĬĞĽĽçĻĞĹÍĦĻĽã
    ĞĢĻĨĿĞĻĨIJĮgĠĞĢĪŃĞĹÂâĬIJĽĽgÂĨĮĴŁĻĽã
    ÂĿĞĬŁĦĽğ ł ĞĢĻĨĿĞĻĨIJĮgÂĨĮĴŁĻĽ
```

```
Ļĝĝĝ
```

And finally add the backward step and printing gradients:

```
# BĞĢĪŃĞĹĤ ĴĞĽĽ
ĬIJĹĽçĞĢĻĨŁĞĻĨIJĮgĠĞĢĪŃĞĹĤâĬIJĹĽçĞĢĻĨŁĞĻĨIJĮgIJŁĻĴŁĻd Ñã
ĤĦĮĹĦğgĠĞĢĪŃĞĹĤâĬIJĹĽçĞĢĻĨŁĞĻĨIJĮgĤĨĮĴŁĻĹã
ĞĢĻĨŁĞĻĨIJĮĝgĠĞĢĪŃĞĹĤâĤĦĮĹĦğgĤĨĮĴŁĻĹã
ĤĦĮĹĦĝgĠĞĢĪŃĞĹĤâĞĢĻĨŁĞĻĨIJĮĝgĤĨĮĴŁĻĹã

# PĹĨĮĻ ÍĹĞĤĨĦĮĻĹ
ĴĹĨĮĻâĤĦĮĹĦĝgĤŃĦĨÍÎĻĹã
ĴĹĨĮĻâĤĦĮĹĦĝgĤĠĨĞĹĦĹã
ĴĹĨĮĻâĤĦĮĹĦğgĤŃĦĨÍÎĻĹã
ĴĹĨĮĻâĤĦĮĹĦğgĤĠĨĞĹĦĹã
```

Full model code:

```
# CĹĦĞĻĦ ĤĞĻĞĹĦĻ
Xd Ñ ł ĹĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĹĦĽłġã

# CĹĦĞĻĦ DĦĮĹĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĹ ĞĮĤ ġ IJŁĻĴŁĻ ŁĞĬŁĦĹ
ĤĦĮĹĦĝ ł LĞÑĦĹçDĦĮĹĦâğd ġã

# CĹĦĞĻĦ RĦLU ĞĢĻĨŁĞĻĨIJĮ âĻIJ ĠĦ ŁĹĦĤ ŃĨĻÎ DĦĮĹĦ ĬĞÑĦĹãf
ĞĢĻĨŁĞĻĨIJĮĝ ł AĢĻĨŁĞĻĨIJĮçRĦLUâã

# CĹĦĞĻĦ ĹĦĢIJĮĤ DĦĮĹĦ ĬĞÑĦĹ ŃĨĻÎ ġ ĨĮĴŁĻ ÌĦĞĻŁĹĦĹ âĞĹ ŃĦ ĻĞĨĦ IJŁĻĴŁĻ
# IJÌ ĴĹĦŁĨIJŁĹ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ġ IJŁĻĴŁĻ ŁĞĬŁĦĹ âIJŁĻĴŁĻ ŁĞĬŁĦĹã
ĤĦĮĹĦğ ł LĞÑĦĹçDĦĮĹĦâġd ġã

# CĹĦĞĻĦ SIJÌĻĬĞŃ ĢĬĞĹĹĨÌÌĨĦĹďĹ ĢIJĬĠĨĮĦĤ ĬIJĹĹ ĞĮĤ ĞĢĻĨŁĞĻĨIJĮ
ĬIJĹĹçĞĢĻĨŁĞĻĨIJĮ ł AĢĻĨŁĞĻĨIJĮçSIJÌĻĬĞŃçLIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑâã

# PĦĹÌIJĹĬ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ĤĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĹ ĬĞÑĦĹ
ĤĦĮĹĦĝgÌIJĹŃĞĹĤâXã

# PĦĹÌIJĹĬ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨŁĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
# ĻĞĪĦĹ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ĤĦĮĹĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨŁĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĹĦĝgIJŁĻĴŁĻã

# PĦĹÌIJĹĬ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽĦĢIJĮĤ DĦĮĹĦ ĬĞÑĦĹ
# ĻĞĪĦĹ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨŁĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
ĤĦĮĹĦğgÌIJĹŃĞĹĤâĞĢĻĨŁĞĻĨIJĮĝgIJŁĻĴŁĻã
```

```
# PHLÌIJLİ Ğ ÌIJLŃĞĹĤ ĴĞĹĹ ĻÎĹIJŁÍÎ ĻÎH ĞĢĻÎĹĞĻÎIJĮwĬIJĹĹ ÌŁĮĢĻÎIJĮ
# ĻĞĪHĹ ĻÎH IJŁĻĴŁĻ IJÌ ĹHĢIJĮĤ ĤHĮĹH ĬĞÑHĹ ÎHĹH ĞĮĤ ĹHĻŁĹĮĹ ĬIJĹĹ
ĬIJĹĹ ł ĬIJĹĹçĞĢĻÎĹĞĻÎIJĮgÌIJĹŃĞĹĤâĤHĮĹHğgIJŁĻĴŁĻd Ñã

# LHĻďĹ ĹHH IJŁĻĴŁĻ IJÌ ĻÎH ÌÎĹĹĻ ÌHŃ ĹĞİĴĬHĹf
ĴĹÎĮĻâĬIJĹĹçĞĢĻÎĹĞĻÎIJĮgIJŁĻĴŁĻäfĥãã

# PĹÎĮĻ ĬIJĹĹ ĹĞĬŁH
ĴĹÎĮĻâďĬIJĹĹfďd ĬIJĹĹã

# CĞĬĢŁĬĞĻH ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻÎĹĞĻÎIJĮğ ĞĮĤ ĻĞĹĹHĻĹ
# ĢĞĬĢŁĬĞĻH ĹĞĬŁHĹ ĞĬIJĮĹ ÌÎĹĹĻ ĞÑÎĹ
ĴĹHĤÎĢĻÎIJĮĹ ł ĮĴgĞĹĹİĞÑâĬIJĹĹçĞĢĻÎĹĞĻÎIJĮgIJŁĻĴŁĻd ĞÑÎĹłĝã
ÎÌ ÌHĮâÑgĹĴĞĴHã łł ğf
    Ñ ł ĮĴgĞĹĹİĞÑâÑd ĞÑÎĹłĝã
ĞĢĢŁĹĞĢÑ ł ĮĴgİHĞĮâĴĹHĤÎĢĻÎIJĮĹłłÑã

# PĹÎĮĻ ĞĢĢŁĹĞĢÑ
ĴĹÎĮĻâďĞĢĢfďd ĞĢĢŁĹĞĢÑã

# BĞĢĪŃĞĹĤ ĴĞĹĹ
ĬIJĹĹçĞĢĻÎĹĞĻÎIJĮgĠĞĢĪŃĞĹĤâĬIJĹĹçĞĢĻÎĹĞĻÎIJĮgIJŁĻĴŁĻd Ñã
ĤHĮĹHğgĠĞĢĪŃĞĹĤâĬIJĹĹçĞĢĻÎĹĞĻÎIJĮgĤÎĮĴŁĻĹã
ĞĢĻÎĹĞĻÎIJĮĝgĠĞĢĪŃĞĹĤâĤHĮĹHğgĤÎĮĴŁĻĹã
ĤHĮĹHĝgĠĞĢĪŃĞĹĤâĞĢĻÎĹĞĻÎIJĮĝgĤÎĮĴŁĻĹã

# PĹÎĮĻ ÍĹĞĤÎHĮĻĹ
ĴĹÎĮĻâĤHĮĹHĝgĤŃHÎÍÎĻĹã
ĴĹÎĮĻâĤHĮĹHĝgĤĠÎĞĹHĹã
ĴĹÎĮĻâĤHĮĹHğgĤŃHÎÍÎĻĹã
ĴĹÎĮĻâĤHĮĹHğgĤĠÎĞĹHĹã


ńńń
ääflgġġġġġġġġ flgġġġġġġġġ flgġġġġġġġġġã
 äflgġġġġġġĝh flgġġġġġġġğ  flgġġġġġġĝhġã
 äflgġġġġġğíì flgġġġġġğî  flgġġġġġġĝíã
 äflgġġġġġğh  flgġġġġġğhġ flgġġġġġġìã
 äflgġġġġġğġġ flgġġġġġğğ  flgġġġġġĥğíāã
ĬIJĹĹf ĝgflîíħĝflġ
ĞĢĢf flgġġ
ää ĝgĥìħĥġĥíHhflġ  ìgíġħíĥìĥHhflĥ  ġgìġğġġflġHhflĥã
 ä ĝgíĝħĝflġħHhflġ  ĝgĝflġĥĥìĝHhflĥ hġgġflîħġĝħHhflĥãã
äähġgħflĥĥġġìHhflġ  îgħħĝĝìğğHhflĥ hĝgflġħìĝġğHhflġãã
ää ĥgġġĝflîĥìHhflĥ  ĝgflìġĝĝġğHhflġ hĝgħĝíğğġħHhflġã
 ähġgflìîĝġĝîHhflĥ hìgĝħìíĝflflHhflĥ  ĝgĝğġħîġġHhflġã
 ähĥgġflĝĝğîîHhflĥ  ígħíĝìğíħHhflĥ hġgğíflĥîîġHhflĥãã
äähĝgflìġğìîġHhflĥ hîgġĥîflîġĝHhflħ  ğgflflğìħğħHhflĥãã
```

Full code up to this point:

```
ĨĬĴIJĹĻ ĮŁĬĴÑ ĞĽ ĮĴ
ĨĬĴIJĹĻ ĮĮÌĽ
ÌĹIJĬ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĨĬĴIJĹĻ ĽĴĨĹĞĬçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


ſi DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ſi LĞÑĦĹ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽãf
        ſi IĮĨĻĨĞĬĨŅĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍÎĻĽ ł flgflĝ ấ ĮĴgĹĞĮĤIJĠgĹĞĮĤĮâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã

    ſi FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ſi RĦİĦĠĦĹ ĨĮĴŁĻ ĽĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ſi CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ĽĞĬŁĦĽ ÌĹIJĬ ĨĮĴŁĻĽd ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĽĦĬÌgŃĦĨÍÎĻĽã κ ĽĦĬÌgĠĨĞĽĦĽ

    ſi BĞĢĨŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤĽĞĬŁĦĽãf
        ſi GĹĞĤĨĦĮĻĽ IJĮ ĴĞĹĞĬĦĻĦĹĽ
        ĽĦĬÌgĤŃĦĨÍÎĻĽ ł ĮĴgĤIJĻâĽĦĬÌgĨĮĴŁĻĽgTd ĤĽĞĬŁĦĽã
        ĽĦĬÌgĤĠĨĞĽĦĽ ł ĮĴgĽŁĬâĤĽĞĬŁĦĽd ĞŃĨĽłfld ĨĦĦĴĤĨĬĽłTĹŁĦã
        ſi GĹĞĤĨĦĮĻ IJĮ ĽĞĬŁĦĽ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĮĴgĤIJĻâĤĽĞĬŁĦĽd ĽĦĬÌgŃĦĨÍÎĻĽgTã


ſi RĦLU ĞĢĻĨĽĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĽĞĻĨIJĮçRĦLUf

    ſi FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
```

```
        # RHIHIGHL IIJJLL LGILHL
        LHIIgIIJJLLL L IIJJLLL
        # CGIGLIGLH IJLLJLL LGILHL IILIJI IIJJLLL
        LHIIgIJLLJLL L IJgIGNIILIâfld IIJJLLLã

    # BGGIINGLÂ JGLL
    ÂHI GGGIINGLÂâLHIId ÂLGILHLãf
        # SIJGH NH JHHÂ LIJ IJHÂIIN IJLIIIIJGI LGLIGGIHd
        # IHLdL IGIH G GJJÑ IJI LGILHL IIILLL
        LHIIgÂIJJLLL L ÂLGILHLgGJJÑâã

        # ZHLIJ ILGÂIHJL NIHLH IJJLL LGILHL NHLH JHIGLIL·H
        LHIIgÂIJJLLäLHIIgIJJLLL l·L flã L fl


# SIJILIGN GGLIL·GLIIJJ
GIGLL AGLIL·GLIIJJçSIJILIGNf

    # FIJLNGLÂ JGLL
    ÂHI IIJLNGLÂâLHIId IJJLLLãf
        # RHIHIGHL IJJLL LGILHL
        LHIIgIJJLLL L IJJLLL

        # GHL LJJIJLIGIIINHHÂ JLIJGGGIIIILIHL
        HNJçLGILHL L JgHNJâIJJLLL h JgIGNâIJJLLLd GNILLLgd
                                                IHHJÂIILLLTLLHãã
        # NIJLIGIINH LIHI IIJL HGGÂ LGIJIH
        JLIJGGGIIIILIHL L HNJçLGILHL w JgLLIâHNJçLGILHLd GNILLLgd
                                                IHHJÂIILLLTLLHã

        LHIIgIJLLJLL L JLIJGGGIIIILIHL

    # BGGIINGLÂ JGLL
    ÂHI GGGIINGLÂâLHIId ÂLGILHLãf

        # CLHGLH LJJIJLIGIIINHHÂ GLLGN
        LHIIgÂIJJLLL L JgHIJLNçIIIHâÂLGILHLã

        # EJLIHLGLH IJLLJLLL GJÂ LLGÂIHJLL
        IIJL IJÂHHNd âLIJJIIHçIJLLJLLd LIJJIIHçÂLGILHLã IJ i
                HJLIHLGLHâNIJâLHIIgIJLLJLLd ÂLGILHLããf
            # FIGLLHJ IJLLJLL GLLGN
            LIJJIIHçIJLLJLL L LIJJIIHçIJLLJLLgLHLIGJHâhgd gã
            # CGIGLIGLH JGGIJGIGJ IGLLIN IJI LIH IJLLJLL GJÂ
            IGGIJGIGJçIGLLIN L JgGÂIGIIIGLâLIJJIIHçIJLLJLLã h i
                                JgGÂIJLâLIJJIIHçIJLLJLLd LIJJIIHçIJLLJLLgTã
```

```
            # CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦhŃĬĽĦ ÍĹĞĤĬĦĮĻ
            # ĞĮĤ ĞĤĤ ĬĻ ĻIJ ĻĬĦ ĞĹĹĞÑ IJÌ ĽĞİĴĬĦ ÍĹĞĤĬĦĮĻĽ
            ĽĦĬÌgĤĬĮĴŁĻĽäĬĮĤĦÑā ł ĮĴgĤIJĻâÏĞĢIJĠĬĞĮçİĞĻĹĬÑd
                                        ĽĬĮÍĬĦçĤĽĞĬŁĦĽã


# CIJİİIJĮ ĬIJĽĽ ĢĬĞĽĽ
ĢĬĞĽĽ LIJĽĽf

    # CĞĬĢŁĬĞĻĦĽ ĻĬĦ ĤĞĻĞ ĞĮĤ ĹĦÍŁĬĞĹĬŅĞĻĬIJĮ ĬIJĽĽĦĽ
    # ÍĬĽĦĮ İIJĤĦĬ IJŁĻĴŁĻ ĞĮĤ ÍĹIJŁĮĤ ĻĹŁĻÎ ĽĞĬŁĦĽ
    ĤĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñãf

        # CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦ ĬIJĽĽĦĽ
        ĽĞİĴĬĦçĬIJĽĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹĤâIJŁĻĴŁĻd Ñã

        # CĞĬĢŁĬĞĻĦ İĦĞĮ ĬIJĽĽ
        ĤĞĻĞçĬIJĽĽ ł ĮĴgİĦĞĮâĽĞİĴĬĦçĬIJĽĽĦĽã

        # RĦĻŁĹĮ ĬIJĽĽ
        ĹĦĻŁĹĮ ĤĞĻĞçĬIJĽĽ


# CĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ
ĢĬĞĽĽ LIJĽĽçCĞĻĦÍIJĹĬĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâLIJĽĽãf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ ĬĮ Ğ ĠĞĻĢĤ
        ĽĞİĴĬĦĽ ł ĬĦĮâÑçĴĹĦĤã

        # CĬĬĴ ĤĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ĤĬŁĬĽĬIJĮ ĠÑ fl
        # CĬĬĴ ĠIJĻĤ ĽĬĤĦĽ ĻIJ ĮIJĻ ĤĹĞÍ İĦĞĮ ĻIJŃĞĹĤĽ ĞĮÑ ŁĞĬŁĦ
        ÑçĴĹĦĤçĢĬĬĴĴĦĤ ł ĮĴgĢĬĬĴâÑçĴĹĦĤd ĝĦhìd ĝ h ĝĦhìã

        # PĹIJĠĞĠĬĬĬĻĬĦĽ ÌIJĹ ĻĞĹÍĦĻ ŁĞĬŁĦĽ h
        # IJĮĬÑ ĬÌ ĢĞĻĦÍIJĹĬĢĞĬ ĬĞĠĦĬĽ
        ĬÌ ĬĦĮâÑçĻĹŁĦgĽĬĞĴĦã łł ĝf
            ĢIJĹĹĦĢĻçĢIJĮÌĬĤĦĮĢĦĽ ł ÑçĴĹĦĤçĢĬĬĴĴĦĤä
                ĹĞĮÍĦâĽĞİĴĬĦĽãd
                ÑçĻĹŁĦ
            ã
```

```
        # MĞĽĨ L·ĞĬŁĦĽ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJĤĦĤ ĬĞĠĦĬĽ
        ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĽ ł ĮĴgĽŁÌâ
                ÑçĴĹĦĤçĢĬĨĴĴĦĤ ǻ ÑçĻĹŁĦd
                ĞŇĨĽłĝ
            ã

        # LIJĹĽĦĽ
        ĮĦÍĞĻĨL·ĦçĬIJÍçĬĨĨĦĬĨĨIJIJĤĽ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĽã
        ĹĦĻŁĹĮ ĮĦÍĞĻĨL·ĦçĬIJÍçĬĨĨĦĬĨĨIJIJĤĽ

    # BĞĢĨŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤL·ĞĬŁĦĽd ÑçĻĹŁĦãf

        # NŁĬĠĦĹ IJÌ ĽĞĬĴĬĦĽ
        ĽĞĬĴĬĦĽ ł ĬĦĮâĤL·ĞĬŁĦĽã
        # NŁĬĠĦĹ IJÌ ĬĞĠĦĬĽ ĨĮ ĦL·ĦĹÑ ĽĞĬĴĬĦ
        # WĦďĬĬ ŁĽĦ ĻĤĦ ÌĨĹĽĻ ĽĞĬĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦĬ
        ĬĞĠĦĬĽ ł ĬĦĮâĤL·ĞĬŁĦĽäflãã

        # IÌ ĬĞĠĦĬĽ ĞĹĦ ĽĴĞĹĽĦd ĻŁĹĮ ĻĤĦĬ ĨĮĻIJ IJĮĦhÎIJĻ L·ĦĢĻIJĹ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ÑçĻĹŁĦ ł ĮĴgĦÑĦâĬĞĠĦĬĽãäÑçĻĹŁĦã

        # CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł hÑçĻĹŁĦ w ĤL·ĞĬŁĦĽ
        # NIJĹĬĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽĞĬĴĬĦĽ


# SIJÌĻĬĞÑ ĢĬĞĽĽĨÌÌĨĦĹ h ĢIJĬĠĨĮĦĤ SIJÌĻĬĞÑ ĞĢĻĨL·ĞĻĨIJĮ
# ĞĮĤ ĢĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ ÌIJĹ ÌĞĽĻĦĹ ĠĞĢĨŃĞĹĤ ĽĻĦĴ
ĢĬĞĽĽ AĢĻĨL·ĞĻĨIJĮçSIJÌĻĬĞÑçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâãf

    # CĹĦĞĻĦĽ ĞĢĻĨL·ĞĻĨIJĮ ĞĮĤ ĬIJĽĽ ÌŁĮĢĻĨIJĮ IJĠÏĦĢĻĽ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ĽĦĬÌgĞĢĻĨL·ĞĻĨIJĮ ł AĢĻĨL·ĞĻĨIJĮçSIJÌĻĬĞÑâã
        ĽĦĬÌgĬIJĽĽ ł LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ÑçĻĹŁĦãf
        # OŁĻĴŁĻ ĬĞÑĦĹďĽ ĞĢĻĨL·ĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĞĢĻĨL·ĞĻĨIJĮgÌIJĹŃĞĹĤâĨĮĴŁĻĽã
        # SĦĻ ĻĤĦ IJŁĻĴŁĻ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĽĦĬÌgĞĢĻĨL·ĞĻĨIJĮgIJŁĻĴŁĻ
        # CĞĬĢŁĬĞĻĦ ĞĮĤ ĹĦĻŁĹĮ ĬIJĽĽ L·ĞĬŁĦ
        ĹĦĻŁĹĮ ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâĽĦĬÌgIJŁĻĴŁĻd ÑçĻĹŁĦã
```

```
    fi BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎHÌ ĠĞĢĪŃĞĹÎâĽHĬÌd ÎĽĞĬŁHĽd ÑçĻĹŁHãf

        fi NŁĬĠHĹ IJÌ ĽĞĬĴĬHĽ
        ĽĞĬĴĬHĽ ł ĬHĮâÎĽĞĬŁHĽã

        fi IÌ ĬĞĠHĬĽ ĞĹH IJĮHhÎIJĻ HĮĢIJÎHÎd
        fi ĻŁĹĮ ĻÎHĬ ĨĮĻIJ ÎĨĽĢĹHĻH ĽĞĬŁHĽ
        ĨÌ ĬHĮâÑçĻĹŁHgĽÎĞĴHã łł ğf
            ÑçĻĹŁH ł ĮĴgĞĹÍĬĞÑâÑçĻĹŁHd ĞÑĨĽłĝã

        fi CIJĴÑ ĽIJ ŃH ĢĞĮ ĽĞÌHĬÑ ĬIJÎĨÌÑ
        ĽHĬÌgÎĨĮĴŁĻĽ ł ÎĽĞĬŁHĽgĢIJĴÑâã
        fi CĞĬĢŁĬĞĻH ÍĹĞÎĨHĮĻ
        ĽHĬÌgÎĨĮĴŁĻĽäĹĞĮÍHâĽĞĬĴĬHĽãd ÑçĻĹŁHã hł ĝ
        fi NIJĹĬĞĬĨŅH ÍĹĞÎĨHĮĻ
        ĽHĬÌgÎĨĮĴŁĻĽ ł ĽHĬÌgÎĨĮĴŁĻĽ w ĽĞĬĴĬHĽ


fi CĹHĞĻH ÎĞĻĞĽHĻ
Xd Ñ ł ĽĴĨĹĞĬçÎĞĻĞâĽĞĬĴĬHĽłĝflfld ĢĬĞĽĽHĽłġã

fi CĹHĞĻH DHĮĽH ĬĞÑHĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌHĞĻŁĹHĽ ĞĮÎ ġ IJŁĻĴŁĻ ĽĞĬŁHĽ
ÎHĮĽHĝ ł LĞÑHĹçDHĮĽHâğd ġã

fi CĹHĞĻH RHLU ĞĢĻĨĽĞĻĨIJĮ âĻIJ ĠH ŁĽHÎ ŃĨĻÎ DHĮĽH ĬĞÑHĹãf
ĞĢĻĨĽĞĻĨIJĮĝ ł AĢĻĨĽĞĻĨIJĮçRHLUâã

fi CĹHĞĻH ĽHĢIJĮÎ DHĮĽH ĬĞÑHĹ ŃĨĻÎ ġ ĨĮĴŁĻ ÌHĞĻŁĹHĽ âĞĽ ŃH ĻĞĬH IJŁĻĴŁĻ
fi IJÌ ĴĹHĽĨIJŁĽ ĬĞÑHĹ ÎHĹHã ĞĮÎ ġ IJŁĻĴŁĻ ĽĞĬŁHĽ âIJŁĻĴŁĻ ĽĞĬŁHĽã
ÎHĮĽHğ ł LĞÑHĹçDHĮĽHâġd ġã

fi CĹHĞĻH SIJÌĻĬĞÑ ĢĬĞĽĽĨÌÌHĹďĽ ĢIJĬĠĨĮHÎ ĬIJĽĽ ĞĮÎ ĞĢĻĨĽĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨĽĞĻĨIJĮ ł AĢĻĨĽĞĻĨIJĮçSIJÌĻĬĞÑçLIJĽĽçCĞĻHÍIJĹĨĢĞĬCĹIJĽĽHĮĻĹIJĴÑâã

fi PHĹÌIJĹĬ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ÎĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑHĹ
ÎHĮĽHĝgÌIJĹŃĞĹÎâXã

fi PHĹÌIJĹĬ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
fi ĻĞĬHĽ ĻÎH IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ÎHĮĽH ĬĞÑHĹ ÎHĹH
ĞĢĻĨĽĞĻĨIJĮĝgÌIJĹŃĞĹÎâÎHĮĽHĝgIJŁĻĴŁĻã

fi PHĹÌIJĹĬ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽHĢIJĮÎ DHĮĽH ĬĞÑHĹ
fi ĻĞĬHĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑHĹ ĞĽ ĨĮĴŁĻĽ
ÎHĮĽHğgÌIJĹŃĞĹÎâĞĢĻĨĽĞĻĨIJĮĝgIJŁĻĴŁĻã

fi PHĹÌIJĹĬ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĻÎH ĞĢĻĨĽĞĻĨIJĮwĬIJĽĽ ÌŁĮĢĻĨIJĮ
fi ĻĞĬHĽ ĻÎH IJŁĻĴŁĻ IJÌ ĽHĢIJĮÎ ÎHĮĽH ĬĞÑHĹ ÎHĹH ĞĮÎ ĹHĻŁĹĮĽ ĬIJĽĽ
ĬIJĽĽ ł ĬIJĽĽçĞĢĻĨĽĞĻĨIJĮgÌIJĹŃĞĹÎâÎHĮĽHğgIJŁĻĴŁĻd Ñã
```

```
ƒ LĦĻďĽ ĽĦĦ IJŁĻĴŁĻ IJÌ ĻÎĦ ÌĨĹĽĻ ÌĦŃ ĽĞİĴĬĦĽf
ĴĹĨĮĻâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻäfĥāã

ƒ PĹĨĮĻ ĬIJĽĽ ĿĞĬŁĦ
ĴĹĨĮĻâďĬIJĽĽfďd ĬIJĽĽã

ƒ CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĿĞĻĨIJĮğ ĞĮĤ ĻĞĹĹĦĻĽ
ƒ ĢĞĬĢŁĬĞĻĦ ĿĞĬŁĦĽ ĞĬIJĮĹ ÌĨĹĽĻ ĞÑĨĽ
ĴĹĦĤĨĢĻĨIJĮĽ ł ĮĴgĞĹĹİĞÑâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻd ĞÑĨĽłĝã
ĨÌ ĬĦĮâÑgĽÎĞĴĦã łł ğf
    Ñ ł ĮĴgĞĹĹİĞÑâÑd ĞÑĨĽłĝã
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦĤĨĢĻĨIJĮĽłłÑã

ƒ PĹĨĮĻ ĞĢĢŁĹĞĢÑ
ĴĹĨĮĻâďĞĢĢfďd ĞĢĢŁĹĞĢÑã

ƒ BĞĢĪŃĞĹĤ ĴĞĽĽ
ĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻd Ñã
ĤĦĮĽĦğgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgĤĨĮĴŁĻĽã
ĞĢĻĨĿĞĻĨIJĮĝgĠĞĢĪŃĞĹĤâĤĦĮĽĦğgĤĨĮĴŁĻĽã
ĤĦĮĽĦĝgĠĞĢĪŃĞĹĤâĞĢĻĨĿĞĻĨIJĮĝgĤĨĮĴŁĻĽã

ƒ PĹĨĮĻ ÍĹĞĤĨĦĮĻĽ
ĴĹĨĮĻâĤĦĮĽĦĝgĤŃĦĨÍÎĻĽã
ĴĹĨĮĻâĤĦĮĽĦĝgĤĠĨĞĽĦĽã
ĴĹĨĮĻâĤĦĮĽĦğgĤŃĦĨÍÎĻĽã
ĴĹĨĮĻâĤĦĮĽĦğgĤĠĨĞĽĦĽã
```

At this point, thanks to gradients and backpropagation using the chain rule, we're able to adjust the weights and biases with the goal of lowering loss, but we'd be doing it in a very rudimentary way. This process of adjusting weights and biases using gradients to decrease loss is the job of the optimizer, which is the subject of the next chapter.

Supplementary Material: https://nnfs.io/ch9
Chapter code, further resources, and errata for this chapter.

Chapter 10

Optimizers

Once we have calculated the gradient, we can use this information to adjust weights and biases to decrease the measure of loss. In a previous toy example, we showed how we could successfully decrease a neuron's activation function's (ReLU) output in this manner. Recall that we subtracted a fraction of the gradient for each weight and bias parameter. While very rudimentary, this is still a commonly used optimizer called **Stochastic Gradient Descent (SGD)**. As you will soon discover, most optimizers are just variants of SGD.

Stochastic Gradient Descent (SGD)

There are some naming conventions with this optimizer that can be confusing, so let's walk through those first. You might hear the following names:

- Stochastic Gradient Descent, SGD
- Vanilla Gradient Descent, Gradient Descent, GD, or Batch Gradient Descent, BGD
- Mini-batch Gradient Descent, MBGD

The first name, **Stochastic Gradient Descent**, historically refers to an optimizer that fits a single sample at a time. The second optimizer, **Batch Gradient Descent**, is an optimizer used to fit a whole dataset at once. The last optimizer, **Mini-batch Gradient Descent**, is used to fit slices of a dataset, which we'd call batches in our context. The naming convention can be confusing here for multiple reasons.

First, in the context of deep learning and this book, we call slices of data **batches**, where, historically, the term to refer to slices of data in the context of Stochastic Gradient Descent was **mini-batches**. In our context, it does not matter if the batch contains a single sample, a slice of the dataset, or the full dataset — as a batch of the data. Additionally, with the current code, we are fitting the full dataset; following this naming convention, we would use **Batch Gradient Descent**. In a future chapter, we'll introduce data slices, or **batches**, so we should start by using the **Mini-batch Gradient Descent** optimizer. That said, current naming trends and conventions with Stochastic Gradient Descent in use with deep learning today have merged and normalized all of these variants, to the point where we think of the **Stochastic Gradient Descent** optimizer as one that assumes a batch of data, whether that batch happens to be a single sample, every sample in a dataset, or some subset of the full dataset at a time.

In the case of Stochastic Gradient Descent, we choose a learning rate, such as *1.0*. We then subtract the *learning_rate · parameter_gradients* from the actual parameter values. If our learning rate is 1, then we're subtracting the exact amount of gradient from our parameters. We're going to start with 1 to see the results, but we'll be diving more into the learning rate shortly. Let's create the SGD optimizer class code. The initialization method will take hyper-parameters, starting with the learning rate, for now, storing them in the class' properties. The `ŁĴĤĞĻĦçĴĞĹĞİĽ` method, given a layer object, performs the most basic optimization, the same way that we performed it in the previous chapter — it multiplies the gradients stored in the layers by the negated learning rate

and adds the result to the layer's parameters. It seems that, in the previous chapter, we performed SGD optimization without knowing it. The full class so far:

```
class OptimizerSGD:

    def __init__(self, learning_rate=1.0, decay=0.0,
    def pre_update_params(self):
        if self.decay:
            self.current_learning_rate = self.learning_rate*(1.
    def update_params(self, layer):
        if hasattr(layer, 'weight_cache'):
            layer.weight_cache += layer.dweights**2
            layer.bias_cache += layer.dbiases**2
        layer.weights += -self.current_learning_rate*layer.dwe
        layer.bias += -self.current_learning_rate*layer.dbiase
```

To use this, we need to create an optimizer object:

```
optimizer = OptimizerSGD(decay=1e-3)
```

Then update our network layer's parameters after calculating the gradient using:

```
optimizer.pre_update_params()
optimizer.update_params(layer)
```

Recall that the layer object contains its parameters (weights and biases) and also, at this stage, the gradient that is calculated during backpropagation. We store these in the layer's properties so that the optimizer can make use of them. In our main neural network code, we'd bring the optimization in after backpropagation. Let's make a 1x64 densely-connected neural network (1 hidden layer with 64 neurons) and use the same dataset as before:

```
# Create dense layer
X, y = sine_data()
dense1 = Layer_Dense(1, 64)

# Create ReLU activation (to be applied to hidden layer)
activation1 = Activation_ReLU()

# Create dense layer with 64 input features (as we take output
# of previous layer here) and 1 output value
dense2 = Layer_Dense(64, 1)

# Create Mean Squared Error loss function
loss_function = Loss_MeanSquaredError()

# Create SGD optimizer with learning rate 1.0 and some decay
optimizer = Optimizer_SGD(learning_rate=1.0, decay=1e-3)

# Forward pass through layer 1
dense1.forward(X)

# Forward through activation function
activation1.forward(dense1.output)

# Forward pass through layer 2
dense2.forward(activation1.output)

# Calculate loss from output of activation2 (so loss function
# of the last layer)
loss = loss_function.calculate(dense2.output, y)

# Print loss value
print('loss:', loss)

# Backpropagation
loss_function.backward(dense2.output, y)
dense2.backward(loss_function.dinputs)
activation1.backward(dense2.dinputs)
dense1.backward(activation1.dinputs)

# Print gradients
print(dense1.dweights)
print(dense1.dbiases)
print(dense2.dweights)
print(dense2.dbiases)
```

The next step is to create the optimizer's object:

```
# CĹĦĞĻĦ IJĴĻĨİĨĨŅĦĹ
IJĴĻĨİĨĨŅĦĹ = OĴĻĨİĨĨŅĦĹçSGDâã
```

Then perform a **forward pass** of our sample data:

```
# PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ĤĞĻĞ ĻĨĹIJŁÍĨĨ ĻĨĨĽ ĬĞÑĦĹ
ĤĦĮĽĦĝgÌIJĹŃĞĹĤâXã

# PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĨĹIJŁÍĨĨ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
# ĻĞĪĦĽ ĻĨĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ĤĦĮĽĦ ĬĞÑĦĹ ĨĦĹĦ
ĞĢĻĨĿĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĽĦĝgIJŁĻĴŁĻã

# PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĨĹIJŁÍĨĨ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ
# ĻĞĪĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
ĤĦĮĽĦĝgÌIJĹŃĞĹĤâĞĢĻĨĿĞĻĨIJĮĝgIJŁĻĴŁĻã

# PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĨĹIJŁÍĨĨ ĻĨĦ ĞĢĻĨĿĞĻĨIJĮwĬIJĽĽ ÌŁĮĢĻĨIJĮ
# ĻĞĪĦĽ ĻĨĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮĤ ĤĦĮĽĦ ĬĞÑĦĹ ĨĦĹĦ ĞĮĤ ĹĦĻŁĹĮĽ ĬIJĽĽ
ĬIJĽĽ = ĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgÌIJĹŃĞĹĤâĤĦĮĽĦĝgIJŁĻĴŁĻd Ñã

# LĦĻďĽ ĴĹĨĮĻ ĬIJĽĽ ĿĞĬŁĦ
ĴĹĨĮĻâďĬIJĽĽfďd ĬIJĽĽã

# CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĿĞĻĨIJĮğ ĞĮĤ ĻĞĹĹĦĻĽ
# ĢĞĬĢŁĬĞĻĦ ĿĞĬŁĦĽ ĞĬIJĮÍ ÌĨĹĽĻ ĞÑĨĽ
ĴĹĦĤĨĢĻĨIJĮĽ = ĮĴgĞĹÍİĞÑâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻd ĞÑĨĽŁĝã
ĨÌ ĬĦĮâÑgĽĴĞĴĦã ŁŁ ğf
    Ñ = ĮĴgĞĹÍİĞÑâÑd ĞÑĨĽŁĝã
ĞĢĢŁĹĞĢÑ = ĮĴgİĦĞĮâĴĹĦĤĨĢĻĨIJĮĽŁŁÑã

ĴĹĨĮĻâďĞĢĢfďd ĞĢĢŁĹĞĢÑã
```

Next, we do our **backward pass**, which is also called **backpropagation**:

```
# BĞĢĪŃĞĹĤ ĴĞĽĽ
ĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻd Ñã
ĤĦĮĽĦĝgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgĤĨĮĴŁĻĽã
ĞĢĻĨĿĞĻĨIJĮĝgĠĞĢĪŃĞĹĤâĤĦĮĽĦĝgĤĨĮĴŁĻĽã
ĤĦĮĽĦĝgĠĞĢĪŃĞĹĤâĞĢĻĨĿĞĻĨIJĮĝgĤĨĮĴŁĻĽã
```

Then we finally use our optimizer to update weights and biases:

```
# UĴĤĞĻĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
IJĴĻĨİĨĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĤĦĮĽĦĝã
IJĴĻĨİĨĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĤĦĮĽĦĝã
```

This is everything we need to train our model! But why would we only perform this optimization once, when we can perform it lots of times by leveraging Python's looping capabilities? We will repeatedly perform a forward pass, backward pass, and optimization until we reach some stopping point. Each full pass through all of the training data is called an **epoch**. In most deep learning tasks, a neural network will be trained for multiple epochs, though the ideal scenario would be to have a perfect model with ideal weights and biases after only one epoch. To add multiple epochs of training into our code, we will initialize our model and run a loop around all the code performing the forward pass, backward pass, and optimization calculations:

```
fi CĹĦĞĻĦ ÎĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçÂĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

fi CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴłĻ ÌĦĞĻłĹĦĽ ĞĮÂ ħǵ IJłĻĴłĻ ŁĞĬłĦĽ
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħǵã

fi CĹĦĞĻĦ RĦLU ĞĢĻĨŁĞĻĨIJĮ âĻIJ ĠĦ łĽĦÂ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨŁĞĻĨIJĮĝ ł AĢĻĨŁĞĻĨIJĮçRĦLUâã

fi CĹĦĞĻĦ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴłĻ ÌĦĞĻłĹĦĽ âĞĽ ŃĦ ĻĞĪĦ IJłĻĴłĻ
fi IJÌ ĴĹĦŁĨIJłĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮÂ ġ IJłĻĴłĻ ŁĞĬłĦĽ âIJłĻĴłĻ ŁĞĬłĦĽã
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâħǵd ġã

fi CĹĦĞĻĦ SIJÌĻİĞŃ ĢĬĞĽĽĨÌĨĦĹďĽ ĢIJİĠĨĮĦÂ ĬIJĽĽ ĞĮÂ ĞĢĻĨŁĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨŁĞĻĨIJĮ ł AĢĻĨŁĞĻĨIJĮçSIJÌĻİĞŃçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

fi CĹĦĞĻĦ IJĴĻĨİĨŅĦĹ
IJĴĻĨİĨŅĦĹ ł OĴĻĨİĨŅĦĹçSGDâã

fi TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝflflflĝãf

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ IJÌ IJłĹ ĻĹĞĨĮĨĮÍ ÂĞĻĞ ĻÎĹIJłÍÎ ĻÎĨĽ ĬĞÑĦĹ
    ÂĦĮĽĦĝgÌIJĹŃĞĹÂâXã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĞĢĻĨŁĞĻĨIJĮ ÌłĮĢĻĨIJĮ
    fi ĻĞĪĦĽ ĻÎĦ IJłĻĴłĻ IJÌ ÌĨĹĽĻ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨŁĞĻĨIJĮĝgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJłĻĴłĻã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ
    fi ĻĞĪĦĽ IJłĻĴłĻĽ IJÌ ĞĢĻĨŁĞĻĨIJĮ ÌłĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴłĻĽ
    ÂĦĮĽĦĝgÌIJĹŃĞĹÂâĞĢĻĨŁĞĻĨIJĮĝgIJłĻĴłĻã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĻÎĦ ĞĢĻĨŁĞĻĨIJĮwĬIJĽĽ ÌłĮĢĻĨIJĮ
    fi ĻĞĪĦĽ ĻÎĦ IJłĻĴłĻ IJÌ ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮÂ ĹĦĻłĹĮĽ ĬIJĽĽ
    ĬIJĽĽ ł ĬIJĽĽçĞĢĻĨŁĞĻĨIJĮgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJłĻĴłĻd Ñã
```

```
ƒ CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĿĞĻĨIJĮğ ĞĮĤ ĻĞĹÍĦĻĹ
ƒ ĢĞĬĢŁĬĞĻĦ ĿĞĬŁĦĹ ĞĬIJĮÍ ÌĨĹĹĻ ĞÑĨĹ
ĴĹĦĤĬĢĻĨIJĮĹ ł ĮĴgĞĹÍİĞÑâĬIJĹĹçĞĢĻĨĿĞĻĨIJĮglJŁĻĴŁĻd ĞÑĨĹłĝã
ĨÌ ĬĦĮâÑgĹÎĞĴĦã łł ğf
    Ñ ł ĮĴgĞĹÍİĞÑâÑd ĞÑĨĹłĝã
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦĤĬĢĻĨIJĮĹłłÑã

ĨÌ ĮIJĻ ĦĴIJĢÎ ķ ĝflflf
    ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
          ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
          ÌďĬIJĹĹf ăĬIJĹĹfgġÌåďã

ƒ BĞĢĪŃĞĹĤ ĴĞĹĹ
ĬIJĹĹçĞĢĻĨĿĞĻĨIJĮgĠĞĢĪŃĞĹĤâĬIJĹĹçĞĢĻĨĿĞĻĨIJĮglJŁĻĴŁĻd Ñã
ĤĦĮĹĦğgĠĞĢĪŃĞĹĤâĬIJĹĹçĞĢĻĨĿĞĻĨIJĮgĤĨĮĴŁĻĹã
ĞĢĻĨĿĞĻĨIJĮĝgĠĞĢĪŃĞĹĤâĤĦĮĹĦğgĤĨĮĴŁĻĹã
ĤĦĮĹĦĝgĠĞĢĪŃĞĹĤâĞĢĻĨĿĞĻĨIJĮĝgĤĨĮĴŁĻĹã

ƒ UĴĤĞĻĦ ŃĦĨÍÎĻĹ ĞĮĤ ĠĨĞĹĦĹ
IJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĹâĤĦĮĹĦĝã
IJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĹâĤĦĮĹĦğã
```

This gives us an update of where we are (epochs), the model's accuracy, and loss every 100 epochs. Initially, we can see consistent improvement:

```
ĦĴIJĢÎf fld ĞĢĢf flgġħfld ĬIJĹĹf ĝgflîî
ĦĴIJĢÎf ĝflfld ĞĢĢf flgǵflfld ĬIJĹĹf ĝgflíì
ĦĴIJĢÎf ğflfld ĞĢĢf flgǵĝìd ĬIJĹĹf ĝgflìì
ggg
ĦĴIJĢÎf ĝflflfld ĞĢĢf flgǵflìd ĬIJĹĹf ĝgflĥí
ggg
ĦĴIJĢÎf ğflflfld ĞĢĢf flgǵflġd ĬIJĹĹf ĝgflġí
ĦĴIJĢÎf ĝĝflfld ĞĢĢf flgǵǵìd ĬIJĹĹf ĝgflĝĝ
ĦĴIJĢÎf ĝğflfld ĞĢĢf flgǵħìd ĬIJĹĹf ĝgflĝġ
ĦĴIJĢÎf ĝġflfld ĞĢĢf flgǵġìd ĬIJĹĹf ĝgflflĥ
ĦĴIJĢÎf ĝǵflfld ĞĢĢf flgǵîìd ĬIJĹĹf flgîîġ
ĦĴIJĢÎf ĝĥflfld ĞĢĢf flgĥĝġd ĬIJĹĹf flgîíĝ
ggg
ĦĴIJĢÎf îĥflfld ĞĢĢf flgĥîfld ĬIJĹĹf flgíħĥ
ĦĴIJĢÎf îħflfld ĞĢĢf flgħğìd ĬIJĹĹf flgíħġ
ĦĴIJĢÎf îìflfld ĞĢĢf flgħġfld ĬIJĹĹf flgíġfl
ĦĴIJĢÎf îíflfld ĞĢĢf flgħħġd ĬIJĹĹf flgíǵǵ
ĦĴIJĢÎf îîflfld ĞĢĢf flgħğìd ĬIJĹĹf flgíğfl
ĦĴIJĢÎf ĝflflfld ĞĢĢf flgħġġd ĬIJĹĹf flgíǵí
```

Additionally, we've prepared animations to help visualize the training process and to convey the impact of various optimizers and their hyperparameters. The left part of the animation canvas

contains dots, where color represents each of the 3 classes of the data, the coordinates are features, and the background colors show the model prediction areas. Ideally, the points' colors and the background should match if the model classifies correctly. The surrounding area should also follow the data's "trend" — which is what we'd call generalization — the ability of the model to correctly predict unseen data. The colorful squares on the right show weights and biases — red for positive and blue for negative values. The matching areas right below the Dense 1 bar and next to the Dense 2 bar show the updates that the optimizer performs to the layers. The updates might look overly strong compared to the weights and biases, but that's because we've visually normalized them to the maximum value, or else they would be almost invisible since the updates are quite small at a time. The other 3 graphs show the loss, accuracy, and current learning rate values in conjunction with the training time, epochs in this case.

Fig 10.01: Model training with Stochastic Gradient Descent optimizer.

Epilepsy Warning, there are quick flashing colors in the animation:

Anim 10.01: https://nnfs.io/pup

Our neural network mostly stays stuck at around a loss of 1 and later 0.85-0.9, and an accuracy around 0.60. The animation also has a “flashy wiggle” effect, which most likely means we chose too high of a learning rate. Given that loss didn’t decrease much, we can assume that this learning rate, being too high, also caused the model to get stuck in a **local minimum**, which we’ll learn more about soon. Iterating over more epochs doesn’t seem helpful at this point, which tells us that we’re likely stuck with our optimization. Does this mean that this is the most we can get from our optimizer on this dataset?

Recall that we’re adjusting our weights and biases by applying some fraction, in this case, *1.0*, to the gradient and subtracting this from the weights and biases. This fraction is called the **learning rate** (LR) and is the primary adjustable parameter for the optimizer as it decreases loss. To gain an intuition for adjusting, planning, or initially setting the learning rate, we should first understand how the learning rate affects the optimizer and output of the loss function.

Learning Rate

So far, we have a gradient of a model and the loss function with respect to all of the parameters, and we want to apply a fraction of this gradient to the parameters in order to descend the loss value.

In most cases, we won't apply the negative gradient as is, as the direction of the function's steepest descent will be continuously changing, and these values will usually be too big for meaningful model improvements to occur. Instead, we want to perform small steps — calculating the gradient, updating parameters by a negative fraction of this gradient, and repeating this in a loop. Small steps ensure that we are following the direction of the steepest descent, but these steps can also be too small, causing learning stagnation — we'll explain this shortly.

Let's forget, for a while, that we are performing gradient descent of an n-dimensional function (our loss function), where n is the number parameters (weights and biases) that the model contains, and assume that we have just one dimension to the loss function (a singular input). Our goal for the following images and animations is to visualize some concepts and gain an intuition; thus, we will not use or present certain optimizer settings, and instead will be considering things in more general terms. That said, we've used a real SGD optimizer on a real function to prepare all of the following examples. Here's the function where we want to determine what input to it will result in the lowest possible output:

Fig 10.02: Example function to minimize the output.

We can see the **global minimum** of this function, which is the lowest possible *y* value that this function can output. This is the goal — to minimize the function's output to find the global minimum. The values of the axes are not important in this case. The goal is only to show the function and the learning rate concept. Also, remember that this one-dimensional function example is being used merely to aid in visualization. It would be easy to solve this function with simpler math than what is required to solve the much larger n-dimensional loss function for neural networks, where n (which is the number of weights and biases) can be in the millions or even billions (or more). When we have millions of, or more, dimensions, gradient descent is the best-known way to search for a global minimum.

We'll start descending from the left side of this graph. With an example learning rate:

Fig 10.03: Stuck in the first local minimum.

Anim 10.03: https://nnfs.io/and

The learning rate turned out to be too small. Small updates to the parameters caused stagnation in the model's learning — the model got stuck in a local minimum. The **local minimum** is a minimum that is near where we look but isn't necessarily the global minimum, which is the absolute lowest point for a function. With our example here, as well as with optimizing full neural networks, we do not know where the global minimum is. How do we know if we've reached the global minimum or at least gotten close? The loss function measures how far the model is with its predictions to the real target values, so, as long as the loss value is not *0* or very close to *0*, and the model stopped learning, we're at some local minimum. In reality, we almost never approach a loss of *0* for various reasons. One reason for this may be imperfect neural network hyperparameters. Another reason for this may be insufficient data. If you did reach a loss of 0 with a neural network, you should find it suspicious, for reasons we'll get into later in this book.

We can try to modify the learning rate:

Fig 10.04: Stuck in the second local minimum.

Anim 10.04: https://nnfs.io/xor

This time, the model escaped this local minimum but got stuck at another one. Let's see one more example after another learning rate change:

Fig 10.05: Stuck in the third local minimum, near the global minimum.

Anim 10.05: https://nnfs.io/tho

This time the model got stuck at a local minimum near the global minimum. The model was able to escape the "deeper" local minimums, so it might be counter-intuitive why it is stuck here. Remember, the model follows the direction of steepest descent of the loss function, no matter how large or slight the descent is. For this reason, we'll introduce momentum and the other techniques to prevent such situations.

Momentum, in an optimizer, adds to the gradient what, in the physical world, we could call inertia — for example, we can throw a ball uphill and, with a small enough hill or big enough applied force, the ball can roll-over to the other side of the hill. Let's see how this might look with the model in training:

Fig 10.06: Reached the global minimum, too low learning rate.

Anim 10.06: https://nnfs.io/pog

We used a very small learning rate here with a large momentum. The color change from green, through orange to red presents the advancement of the gradient descent process, the steps. We can see that the model achieved the goal and found the global minimum, but this took many steps. Can this be done better?

Fig 10.07: Reached the global minimum, better learning rate.

Anim 10.07: https://nnfs.io/jog

And even further:

Fig 10.08: Reached the global minimum, significantly better learning rate.

Anim 10.08: https://nnfs.io/mog

With these examples, we were able to find the global minimum in about 200, 100, and 50 steps, respectively, by modifying the learning rate and the momentum. It's possible to significantly shorten the training time by adjusting the parameters of the optimizer. However, we have to be careful with these hyper-parameter adjustments, as this won't necessarily always help the model:

Fig 10.09: Unstable model, learning rate too big.

Anim 10.09: https://nnfs.io/log

With the learning rate set too high, the model might not be able to find the global minimum. Even, at some point, if it does, further adjustments could cause it to jump out of this minimum. We'll see this behavior later in this chapter — try to take a close look at results and see if you can find it, as well as the other issues we've described, from the different optimizers as we work through them.

In this case, the model was "jumping" around some minimum and what this might mean is that we should try to lower the learning rate, raise the momentum, or possibly apply a learning rate decay (lowering the learning rate during training), which we'll describe in this chapter. If we set the learning rate far too high:

Fig 10.10: Unstable model, learning rate significantly too big.

Anim 10.10: https://nnfs.io/sog

In this situation, the model starts “jumping” around, and moves in what we might observe as random directions. This is an example of “overshooting,” with every step — the direction of a change is correct, but the amount of the gradient applied is too large. In an extreme situation, we could cause a **gradient explosion**:

Fig 10.11: Broken model, learning rate critically too big.

Anim 10.11: https://nnfs.io/bog

A gradient explosion is a situation where the parameter updates cause the function’s output to rise instead of fall, and, with each step, the loss value and gradient become larger. At some point, the floating-point variable limitation causes an overflow as it cannot hold values of this size anymore, and the model is no longer able to train. It’s crucial to recognize this situation forming during training, especially for large models, where the training can take days, weeks, or more. It is possible to tune the model’s hyper-parameters in time to save the model and to continue training.

When we choose the learning rate and the other hyper-parameters correctly, the learning process can be relatively quick:

Fig 10.12: Model learned, good learning rate, can be better.

Anim 10.12: https://nnfs.io/cog

This time it took significantly less time for the model to find the global minimum, but it can always be better:

Fig 10.13: An efficient learning example.

Anim 10.13: https://nnfs.io/rog

This time the model needed just a few steps to find the global minimum. The challenge is to choose the hyper-parameters correctly, and it is not always an easy task. It is usually best to start with the optimizer defaults, perform a few steps, and observe the training process when tuning different settings. It is not always possible to see meaningful results in a short-enough period of time, and, in this case, it's good to have the ability to update the optimizer's settings during training. How you choose the learning rate, and other hyper-parameters, depends on the model, data, including the amount of data, the parameter initialization method, etc. There is no single, best way to set hyper-parameters, but experience usually helps. As we mentioned, one

of the challenges during the training of a neural network model is to choose the right settings. The

difference can be anything from a model not learning at all to learning very well.

For a summary of learning rates — if we plot the loss along an axis of steps:

Fig 10.14: Graphs of the loss in a function of steps, different rates

We can see various examples of relative learning rates and what loss will ideally look like as a graph over time (steps) of training.

Knowing what the learning rate should be to get the most out of your training process isn't possible, but a good rule is that your initial training will benefit from a larger learning rate to take initial steps faster. If you start with steps that are too small, you might get stuck in a local minimum and be unable to leave it due to not making large enough updates to the parameters. For example, what if we make the learning rate 0.85 rather than 1.0 with the SGD optimizer?

```
fi CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

fi CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴłĻ ÌĦĞĻłĹĦĽ ĞĮĤ ħǵ IJłĻĴłĻ ĽĞĬłĦĽ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħǵã

fi CĹĦĞĻĦ RĦLU ĞĢĻĨĽĞĻĨIJĮ âĻIJ ĠĦ łĽĦĤ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĽĞĻĨIJĮĝ ł AĢĻĨĽĞĻĨIJĮçRĦLUâã
```

```
ﬁ CĹĦĞĻĦ ĽĦĢIJĮÎ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJŁĻĴŁĻ
ﬁ IJÌ ĴĹĦŁĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮÎ ġ IJŁĻĴŁĻ ŁĞĬŁĦĽ âIJŁĻĴŁĻ ŁĞĬŁĦĽã
ÎĦĮĽĦǧ ł LĞÑĦĹçDĦĮĽĦâħǵd ġã

ﬁ CĹĦĞĻĦ SIJÌĻİĞÑ ĢĬĞĽĽĨÌĨĦĹďĽ ĢIJİĠĨĮĦÎ ĬIJĽĽ ĞĮÎ ĞĢĻĨŁĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨŁĞĻĨIJĮ ł AĢĻĨŁĞĻĨIJĮçSIJÌĻİĞÑçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

ﬁ CĹĦĞĻĦ IJĴĻĨİĨĨĢĦĹ
IJĴĻĨİĨĨĢĦĹ ł OĴĻĨİĨĨĢĦĹçSGDâĬĦĞĹĮĨĮÍçĹĞĻĦłgíĥã

ﬁ TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝﬁﬁﬁĝãf

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ÎĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑĦĹ
    ÎĦĮĽĦĝgÌIJĹŃĞĹÎâXã

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨŁĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    ﬁ ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĽ ÎĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨŁĞĻĨIJĮĝgÌIJĹŃĞĹÎâÎĦĮĽĦĝgIJŁĻĴŁĻã

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽĦĢIJĮÎ DĦĮĽĦ ĬĞÑĦĹ
    ﬁ ĻĞĨĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨŁĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĽ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
    ÎĦĮĽĦĝgÌIJĹŃĞĹÎâĞĢĻĨŁĞĻĨIJĮĝgIJŁĻĴŁĻã

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĻÎĦ ĞĢĻĨŁĞĻĨIJĮwĬIJĽĽ ÌŁĮĢĻĨIJĮ
    ﬁ ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮÎ ÎĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮÎ ĹĦĻŁĹĮĽ ĬIJĽĽ
    ĬIJĽĽ ł ĬIJĽĽçĞĢĻĨŁĞĻĨIJĮgÌIJĹŃĞĹÎâÎĦĮĽĦĝgIJŁĻĴŁĻd Ñã

    ﬁ CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨŁĞĻĨIJĮǧ ĞĮÎ ĻĞĹĹĦĻĽ
    ﬁ ĢĞĬĢŁĬĞĻĦ ŁĞĬŁĦĽ ĞĬIJĮÍ ÌĨĹĽĽ ĞÑĨĽ
    ĴĹĦÎĨĢĻĨIJĮĽ ł ĮĴgĞĹÍİĞÑâĬIJĽĽçĞĢĻĨŁĞĻĨIJĮgIJŁĻĴŁĻd ĞÑĨĽłĝã
    ĨÌ ĬĦĮâÑgĽÎĞĴĦã łł ǧf
        Ñ ł ĮĴgĞĹÍİĞÑâÑd ĞÑĨĽłĝã
    ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦÎĨĢĻĨIJĮĽłłÑã

    ĨÌ ĮIJĻ ĦĴIJĢÎ ķ ĝﬁﬁf
        ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
              ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĽĽf ăĬIJĽĽfgġÌåďã

    ﬁ BĞĢĬŃĞĹÎ ĴĞĽĽ
    ĬIJĽĽçĞĢĻĨŁĞĻĨIJĮgĠĞĢĬŃĞĹÎâĬIJĽĽçĞĢĻĨŁĞĻĨIJĮgIJŁĻĴŁĻd Ñã
    ÎĦĮĽĦĝgĠĞĢĬŃĞĹÎâĬIJĽĽçĞĢĻĨŁĞĻĨIJĮgÎĨĮĴŁĻĽã
    ĞĢĻĨŁĞĻĨIJĮĝgĠĞĢĬŃĞĹÎâÎĦĮĽĦĝgÎĨĮĴŁĻĽã
    ÎĦĮĽĦĝgĠĞĢĬŃĞĹÎâĞĢĻĨŁĞĻĨIJĮĝgÎĨĮĴŁĻĽã

    ﬁ UĴÎĞĻĦ ŃĦĨÍÎĮĽ ĞĮÎ ĠĨĞĽĦĽ
    IJĴĻĨİĨĨĢĦĹgŁĴÎĞĻĦçĴĞĹĞİĽâÎĦĮĽĦĝã
    IJĴĻĨİĨĨĢĦĹgŁĴÎĞĻĦçĴĞĹĞİĽâÎĦĮĽĦĝã
```

```
ńńń
ĦĴIJĢÎf fld ĞĢĢf flgġħfld ĬIJĹĹf ĝgflîî
ĦĴIJĢÎf ĝflfld ĞĢĢf flgǵflġd ĬIJĹĹf ĝgflîĝ
ggg
ĦĴIJĢÎf ğflflfld ĞĢĢf flgǵġìd ĬIJĹĹf ĝgflĥġ
ĦĴIJĢÎf ğĝflfld ĞĢĢf flgǵǵġd ĬIJĹĹf ĝgflğħ
ĦĴIJĢÎf ğğflfld ĞĢĢf flgġìì d ĬIJĹĹf ĝgflĥfl
ĦĴIJĢÎf ğġflfld ĞĢĢf flgǵġġd ĬIJĹĹf ĝgflĝħ
ĦĴIJĢÎf ğǵflfld ĞĢĢf flgǵħfld ĬIJĹĹf ĝgflflfl
ĦĴIJĢÎf ğĥflfld ĞĢĢf flgǵîġd ĬIJĹĹf ĝgflĝfl
ĦĴIJĢÎf ğħflfld ĞĢĢf flgĥğìd ĬIJĹĹf flgîîí
ĦĴIJĢÎf ğìflfld ĞĢĢf flgĥğġd ĬIJĹĹf flgîìì
ggg
ĦĴIJĢÎf ìĝflfld ĞĢĢf flgĥìì d ĬIJĹĹf flgîǵĝ
ĦĴIJĢÎf ìğflfld ĞĢĢf flgĥĥfld ĬIJĹĹf flgîğĝ
ĦĴIJĢÎf ìġflfld ĞĢĢf flgĥîġd ĬIJĹĹf flgîǵġ
ĦĴIJĢÎf ìǵflfld ĞĢĢf flgĥîġd ĬIJĹĹf flgîǵfl
ĦĴIJĢÎf ìĥflfld ĞĢĢf flgĥĥì d ĬIJĹĹf flgîflì
ĦĴIJĢÎf ìħflfld ĞĢĢf flgĥîfld ĬIJĹĹf flgîǵî
ĦĴIJĢÎf ììflfld ĞĢĢf flgĥîfld ĬIJĹĹf flgîġĥ
ggg
ĦĴIJĢÎf îĝflfld ĞĢĢf flgĥîì d ĬIJĹĹf flgíħfl
ĦĴIJĢÎf îğflfld ĞĢĢf flgħġfld ĬIJĹĹf flgíǵğ
ggg
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgħĥì d ĬIJĹĹf flgíĝħ
```

Fig 10.15: Model training with SGD optimizer and lowered learning rate.

Epilepsy Warning (quick flashing colors).

Anim 10.15: https://nnfs.io/cup

As you can see, the neural network did slightly better in terms of accuracy, and it achieved a lower loss; lower loss is not always associated with higher accuracy. Remember, even if we desire the best accuracy out of our model, the optimizer's task is to decrease loss, not raise accuracy directly. Loss is the mean value of all of the sample losses, and some of them could drop significantly, while others might rise just slightly, changing the prediction for them from a correct to an incorrect class at the same time. This would cause a lower mean loss in general, but also more incorrectly predicted samples, which will, at the same time, lower the accuracy. A likely reason for this model's lower accuracy is that it found another local minimum by chance — the descent path has changed, due to smaller steps. In a direct comparison of these two models in training, different learning rates did not show that the lower this learning rate value is, the better. In most cases, we want to start with a larger learning rate and decrease the learning rate over time/steps.

A commonly-used solution to keep initial updates large and explore various learning rates during training is to implement a **learning rate decay**.

Learning Rate Decay

The idea of a **learning rate decay** is to start with a large learning rate, say 1.0 in our case, and then decrease it during training. There are a few methods for doing this. One is to decrease the learning rate in response to the loss across epochs — for example, if the loss begins to level out/ plateau or starts "jumping" over large deltas. You can either program this behavior-monitoring logically or simply track your loss over time and manually decrease the learning rate when you deem it appropriate. Another option, which we will implement, is to program a **Decay Rate**, which steadily decays the learning rate per batch or epoch.

Let's plan to decay per step. This can also be referred to as **1/t decaying** or **exponential decaying**. Basically, we're going to update the learning rate each step by the reciprocal of the step count fraction. This fraction is a new hyper-parameter that we'll add to the optimizer, called the **learning rate decay**. How this decaying works is it takes the step and the decaying ratio and multiplies them. The further in training, the bigger the step is, and the bigger result of this multiplication is. We then take its reciprocal (the further in training, the lower the value) and multiply the initial learning rate by it. The added *1* makes sure that the resulting algorithm never raises the learning rate. For example, for the first step, we might divide 1 by the learning rate, *0.001* for example, which will result in a current learning rate of *1000*. That's definitely not what we wanted. 1 divided by the 1+fraction ensures that the result, a fraction of the starting learning rate, will always be less than or equal to 1, decreasing over time. That's the desired result — start with the current learning rate and make it smaller with time. The code for determining the current decay rate:

```
ĽĻĞĹĻĨĮĺçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĝg
ĬĦĞĹĮĨĮĺçĹĞĻĦçĤĦĢĞÑ ł flgĝ
ĽĻĦĴ ł ĝ

ĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĽĻĞĹĻĨĮĺçĬĦĞĹĮĨĮĺçĹĞĻĦ ǻ ċ
                 âĝg w âĝ κ ĬĦĞĹĮĨĮĺçĹĞĻĦçĤĦĢĞÑ ǻ ĽĻĦĴãã
ĴĹĨĮĻâĬĦĞĹĮĨĮĺçĹĞĻĦã


ńńń
flgîflîflîflîflîflîflîflîĝ
```

In practice, 0.1 would be considered a fairly aggressive decay rate, but this should give you a sense of the concept. If we are on step 20:

```
ĽĻĞĹĻĨĮĺçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĝg
ĬĦĞĹĮĨĮĺçĹĞĻĦçĤĦĢĞÑ ł flgĝ
ĽĻĦĴ ł ğfl

ĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĽĻĞĹĻĨĮĺçĬĦĞĹĮĨĮĺçĹĞĻĦ ấ ċ
                 âĝg w âĝ κ ĬĦĞĹĮĨĮĺçĹĞĻĦçĤĦĢĞÑ ấ ĽĻĦĴãã
ĴĹĨĮĻâĬĦĞĹĮĨĮĺçĹĞĻĦã


ńńń
flgġġġġġġġġġġġġġġġ
```

We can also simulate this in a loop, which is more comparable to how we will be applying learning rate decay:

```
ĽĻĞĹĻĨĮĺçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĝg
ĬĦĞĹĮĨĮĺçĹĞĻĦçĤĦĢĞÑ ł flgĝ

ÌIJĹ ĽĻĦĴ ĨĮ ĹĞĮĺĦâğflãf
    ĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĽĻĞĹĻĨĮĺçĬĦĞĹĮĨĮĺçĹĞĻĦ ấ ċ
                     âĝg w âĝ κ ĬĦĞĹĮĨĮĺçĹĞĻĦçĤĦĢĞÑ ấ ĽĻĦĴãã
    ĴĹĨĮĻâĬĦĞĹĮĨĮĺçĹĞĻĦã


ńńń
ĝgfl
flgîflîflîflîflîflîflîĝ
flgíġġġġġġġġġġġġġġ
flgìĥîğġflìĥîğġflìĥîğ
flgìĝĝğíĤìĝĝğíĤìĝĝġ
flghhhhhhhhhhhhhhhh
flgĥğĤ
flgĤííğġĤğîġĝĝìĥġì
flgĤĤĤĤĤĤĤĤĤĤĤĤĤĤĤĤ
flgĤğĥġĝĤìííîġìġĥíġğ
flgĤ
flgġìĥĝîflġìĥĝîflġìĥĝĥ
flgġĤġĤġĤġĤġĤġĤġĤġĤġ
flgġġġìíğĥflíĥîĤĥĤğğ
flgġĝĥĥĥĥĥĥĥĥĥĥĥĥġ
flgġ
flgġíġĥĝĤġíġĥĝĤġíġĥ
flgġìflġìflġìflġìflġìflġĤ
flgġĤìĝĝğíĤìĝĝğíĤìĝĤ
flgġġġíğìĤíĥğflĥííĥĤ
```

This learning rate decay scheme lowers the learning rate each step using the mentioned formula. Initially, the learning rate drops fast, but the change in the learning rate lowers each step, letting the model sit as close as possible to the minimum. The model needs small updates near the end of training to be able to get as close to this point as possible. We can now update our SGD optimizer class to allow for the learning rate decay:

```
ƒ SGD IJĴĻĨĬĨŅĦĹ
ĢĬĞĹĹ OĴĻĨĬĨŅĦĹçSGDf

    ƒ IĮĨĻĨĞĬĨŅĦ IJĴĻĨĬĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮĺĹd
    ƒ ĬĦĞĹĮĨĮĺ ĹĞĻĦ IJÌ ĝg ĨĹ ĤĦÌĞŁĬĻ ÌIJĹ ĻĴĨĹ IJĴĻĨĬĨŅĦĹ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮĺçĹĞĻĦłĝgd ĤĦĢĞÑłflgãf
        ĹĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĹĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl

    ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴĤĞĻĦĹ
    ĤĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgĤĦĢĞÑf
            ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĹĦĬÌgĤĦĢĞÑ ắ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    ƒ UĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĹ
    ĤĦÌ łĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌd ĬĞÑĦĹãf
        ĬĞÑĦĹgŃĦĨĬĬĻĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ĬĞÑĦĹgĤŃĦĨĬĬĻĹ
        ĬĞÑĦĹgĠĨĞĹĦĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ĬĞÑĦĹgĤĠĨĞĹĦĹ

    ƒ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴĤĞĻĦĹ
    ĤĦÌ ĴIJĹĻçŁĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌãf
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κł ĝ
```

We've updated a few things in the SGD class. First, in the `ççĨĮĨĻçç` method, we added handling for the current learning rate, and `ĹĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ` is now the initial learning rate. We also added attributes to track the decay rate and the number of iterations that the optimizer has gone through. Next, we added a new method called `ĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĹ`. This method, if we have a decay rate other than 0, will update our `ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ` using the prior formula. The `łĴĤĞĻĦçĴĞĹĞĬĹ` method remains unchanged, but we do have a new `ĴIJĹĻçŁĴĤĞĻĦçĴĞĹĞĬĹ` method that will add to our `ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ` tracking. With our updated SGD optimizer class, we've added printing the current learning rate, and added pre and post optimizer method calls. Let's use a decay rate of 1e-2 (0.01) and train our model again:

```
ﬠ CĹĦĞĻĦ ÂĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçÂĞĻĞâĽĞİĴĬĦĽłĝﬠﬠd ĢĬĞĽĽĦĽłġã

ﬠ CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴłĻ ÌĦĞĻłĹĦĽ ĞĮÂ ħǵ IJłĻĴłĻ ĽĞĬłĦĽ
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħǵã

ﬠ CĹĦĞĻĦ RĦLU ĞĢĻĨĽĞĻĨIJĮ âĻIJ ĠĦ łĽĦÂ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĽĞĻĨIJĮĝ ł AĢĻĨĽĞĻĨIJĮçRĦLUâã

ﬠ CĹĦĞĻĦ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴłĻ ÌĦĞĻłĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJłĻĴłĻ
ﬠ IJÌ ĴĹĦĽĨIJłĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮÂ ġ IJłĻĴłĻ ĽĞĬłĦĽ âIJłĻĴłĻ ĽĞĬłĦĽã
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâħǵd ġã

ﬠ CĹĦĞĻĦ SIJÌĻİĞŃ ĢĬĞĽĽĨÌÌĦĹďĽ ĢIJİĠĨĮĦÂ ĬIJĽĽ ĞĮÂ ĞĢĻĨĽĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨĽĞĻĨIJĮ ł AĢĻĨĽĞĻĨIJĮçSIJÌĻİĞŃçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

ﬠ CĹĦĞĻĦ IJĴĻĨİĨŅĦĹ
IJĴĻĨİĨŅĦĹ ł OĴĻĨİĨŅĦĹçSGDâÂĦĢĞÑłĝĦhğã

ﬠ TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝﬠﬠﬠĝãf

    ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ IJÌ IJłĹ ĻĹĞĨĮĨĮÍ ÂĞĻĞ ĻÎĹIJłÍÎ ĻÎĨĽ ĬĞÑĦĹ
    ÂĦĮĽĦĝgÌIJĹŃĞĹÂâXã

    ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĞĢĻĨĽĞĻĨIJĮ ÌłĮĢĻĨIJĮ
    ﬠ ĻĞĨĦĽ ĻÎĦ IJłĻĴłĻ IJÌ ÌĨĹĽĻ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨĽĞĻĨIJĮĝgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJłĻĴłĻã

    ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ
    ﬠ ĻĞĨĦĽ IJłĻĴłĻĽ IJÌ ĞĢĻĨĽĞĻĨIJĮ ÌłĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴłĻĽ
    ÂĦĮĽĦğgÌIJĹŃĞĹÂâĞĢĻĨĽĞĻĨIJĮĝgIJłĻĴłĻã

    ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĻÎĦ ĞĢĻĨĽĞĻĨIJĮwĬIJĽĽ ÌłĮĢĻĨIJĮ
    ﬠ ĻĞĨĦĽ ĻÎĦ IJłĻĴłĻ IJÌ ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮÂ ĹĦĻłĹĮĽ ĬIJĽĽ
    ĬIJĽĽ ł ĬIJĽĽçĞĢĻĨĽĞĻĨIJĮgÌIJĹŃĞĹÂâÂĦĮĽĦğgIJłĻĴłĻd Ñã

    ﬠ CĞĬĢłĬĞĻĦ ĞĢĢłĹĞĢÑ ÌĹIJİ IJłĻĴłĻ IJÌ ĞĢĻĨĽĞĻĨIJĮğ ĞĮÂ ĻĞĹĹĦĻĽ
    ﬠ ĢĞĬĢłĬĞĻĦ ĽĞĬłĦĽ ĞĬIJĮĹ ÌĨĹĽĻ ĞŃĨĽ
    ĴĹĦÂĨĢĻĨIJĮĽ ł ĮĴgĞĹĹİĞŃâĬIJĽĽçĞĢĻĨĽĞĻĨIJĮgIJłĻĴłĻd ĞŃĨĽłĝã
    ĨÌ ĬĦĮâÑgĽÎĞĴĦã łł ğf
        Ñ ł ĮĴgĞĹĹİĞŃâÑd ĞŃĨĽłĝã
    ĞĢĢłĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦÂĨĢĻĨIJĮĽłłÑã

    ĨÌ ĮIJĻ ĦĴIJĢÎ ķ ĝﬠﬠf
        ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
            ÌďĞĢĢf ăĞĢĢłĹĞĢÑfgġÌåd ď κ
            ÌďĬIJĽĽf ăĬIJĽĽfgġÌåd ď κ
            ÌďĬĹf ăIJĴĻĨİĨŅĦĹgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦåďã
```

```
ﬁ BĞĢĪŃĞĹĤ ĴĞĽĽ
ĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻd Ñã
ĤĦĮĽĦğgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgĤĨĮĴŁĻĽã
ĞĢĻĨĿĞĻĨIJĮĝgĠĞĢĪŃĞĹĤâĤĦĮĽĦğgĤĨĮĴŁĻĽã
ĤĦĮĽĦĝgĠĞĢĪŃĞĹĤâĞĢĻĨĿĞĻĨIJĮĝgĤĨĮĴŁĻĽã

ﬁ UĴĤĞĻĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
IJĴĻĨĬĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĽâã
IJĴĻĨĬĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞĬĽâĤĦĮĽĦĝã
IJĴĻĨĬĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞĬĽâĤĦĮĽĦğã
IJĴĻĨĬĨŅĦĹgĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞĬĽâã
```

ńńń

```
ĦĴIJĢÎf fld ĞĢĢf flgġħfld ĬIJĽĽf ĝgflîîd ĬĹf ĝgfl
ĦĴIJĢÎf ĝflfld ĞĢĢf flgǵflġd ĬIJĽĽf ĝgflîĥd ĬĹf flgĥflĝĥĝĝĥħĝíĝǵflìflġ
ĦĴIJĢÎf ĝflfld ĞĢĢf flgġîìd ĬIJĽĽf ĝgflíǵd ĬĹf flgġġǵǵǵíĝħflĥġĥĝĝìflħ
ĦĴIJĢÎf ġflfld ĞĢĢf flgǵflfld ĬIJĽĽf ĝgflífld ĬĹf flgĝĥflħĝħĥħħǵĝħflǵflĝ
ĦĴIJĢÎf ǵflfld ĞĢĢf flgǵflìd ĬIJĽĽf ĝgflìíd ĬĹf flgĝflflǵflflíflĝħflġĝflħǵ
ĦĴIJĢÎf ĥflfld ĞĢĢf flgǵĝfld ĬIJĽĽf ĝgflìíd ĬĹf flgĝħħîǵǵîflíĝíflġflflĥ
ĦĴIJĢÎf ħflfld ĞĢĢf flgǵĝfld ĬIJĽĽf ĝgflììd ĬĹf flgĝǵǵflħĝĥĝħǵĥĝflìǵġí
ĦĴIJĢÎf ìflfld ĞĢĢf flgǵĝìd ĬIJĽĽf ĝgflììd ĬĹf flgĝĝĥĝĥħǵǵĥĥĥħîǵħĝ
ĦĴIJĢÎf íflfld ĞĢĢf flgǵĝġd ĬIJĽĽf ĝgflììd ĬĹf flgĝĝĝĝĝǵìflĥĝĝíflġĝĝǵ
ĦĴIJĢÎf îflfld ĞĢĢf flgǵĝfld ĬIJĽĽf ĝgflììd ĬĹf flgĝflflĝflflĝflflĝflflĝflflî
ĦĴIJĢÎf ĝflflfld ĞĢĢf flgǵĝìd ĬIJĽĽf ĝgflììd ĬĹf flgflîflîîĝíĝflìġìflġġħħ
ggg
ĦĴIJĢÎf ĝflflfld ĞĢĢf flgǵĝfld ĬIJĽĽf ĝgflìħd ĬĹf flgflǵìħǵĝìġǵĝĥîĝĝġǵíħ
ggg
ĦĴIJĢÎf ġflflfld ĞĢĢf flgǵĝġd ĬIJĽĽf ĝgflìĥd ĬĹf flgflġĝĝħíǵìġìflĝĝîġĝîġ
ggg
ĦĴIJĢÎf ǵflflfld ĞĢĢf flgǵflìd ĬIJĽĽf ĝgflìĥd ĬĹf flgflĝǵġîħĝîǵĝîġìflĥìí
ggg
ĦĴIJĢÎf ĥflflfld ĞĢĢf flgǵflġd ĬIJĽĽf ĝgflìǵd ĬĹf flgflĝîħĝĝħííĥħħġíĥĥħħ
ggg
ĦĴIJĢÎf ìflflfld ĞĢĢf flgǵflfld ĬIJĽĽf ĝgflìġd ĬĹf flgflĝǵflíħǵîĝflĥĥflìíĝíĝ
ggg
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgġîìd ĬIJĽĽf ĝgflìĝd ĬĹf flgflflîîflĝîìflǵîĝĝĝìîġġ
```

Fig 10.16: Model training with SGD optimizer and and learning rate decay set too high.

Epilepsy Warning (quick flashing colors)

Anim 10.16: https://nnfs.io/zuk

This model definitely got stuck, and the reason is almost certainly because the learning rate decayed far too quickly and became too small, trapping the model in some local minimum. This is most likely why, rather than wiggling, our accuracy and loss stopped changing *at all*.

We can, instead, try to decay a bit slower by making our decay a smaller number. For example, let's go with 1e-3 (0.001):

```
ﬠ CĹĦĞĻĦ ÂĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçÂĞĻĞâĽĞİĴĬĦĽłĝﬠﬠd ĢĬĞĽĽĦĽłġã

ﬠ CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴłĻ ÌĦĞĻłĹĦĽ ĞĮÂ ħǵ IJłĻĴłĻ ĽĞĬłĦĽ
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħǵã

ﬠ CĹĦĞĻĦ RĦLU ĞĢĻĨĽĞĻĨIJĮ âĻIJ ĠĦ łĽĦÂ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĽĞĻĨIJĮĝ ł AĢĻĨĽĞĻĨIJĮçRĦLUâã

ﬠ CĹĦĞĻĦ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴłĻ ÌĦĞĻłĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJłĻĴłĻ
ﬠ IJÌ ĴĹĦĽĨIJłĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮÂ ġ IJłĻĴłĻ ĽĞĬłĦĽ âIJłĻĴłĻ ĽĞĬłĦĽã
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâħǵd ġã

ﬠ CĹĦĞĻĦ SIJÌĻİĞŃ ĢĬĞĽĽĨÌÌĦĹďĽ ĢIJİĠĨĮĦÂ ĬIJĽĽ ĞĮÂ ĞĢĻĨĽĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨĽĞĻĨIJĮ ł AĢĻĨĽĞĻĨIJĮçSIJÌĻİĞŃçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

ﬠ CĹĦĞĻĦ IJĴĻĨİĨŅĦĹ
IJĴĻĨİĨŅĦĹ ł OĴĻĨİĨŅĦĹçSGDâÂĦĢĞÑłĝĦhġã

ﬠ TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝﬠﬠﬠĝãf

    ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ IJÌ IJłĹ ĻĹĞĨĮĨĮÍ ÂĞĻĞ ĻÎĹIJłÍÎ ĻÎĨĽ ĬĞÑĦĹ
    ÂĦĮĽĦĝgÌIJĹŃĞĹÂâXã

    ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĞĢĻĨĽĞĻĨIJĮ ÌłĮĢĻĨIJĮ
    ﬠ ĻĞĨĦĽ ĻÎĦ IJłĻĴłĻ IJÌ ÌĨĹĽĻ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨĽĞĻĨIJĮĝgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJłĻĴłĻã

    ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ
    ﬠ ĻĞĨĦĽ IJłĻĴłĻĽ IJÌ ĞĢĻĨĽĞĻĨIJĮ ÌłĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴłĻĽ
    ÂĦĮĽĦğgÌIJĹŃĞĹÂâĞĢĻĨĽĞĻĨIJĮĝgIJłĻĴłĻã

    ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĻÎĦ ĞĢĻĨĽĞĻĨIJĮwĬIJĽĽ ÌłĮĢĻĨIJĮ
    ﬠ ĻĞĨĦĽ ĻÎĦ IJłĻĴłĻ IJÌ ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮÂ ĹĦĻłĹĮĽ ĬIJĽĽ
    ĬIJĽĽ ł ĬIJĽĽçĞĢĻĨĽĞĻĨIJĮgÌIJĹŃĞĹÂâÂĦĮĽĦğgIJłĻĴłĻd Ñã

    ﬠ CĞĬĢłĬĞĻĦ ĞĢĢłĹĞĢÑ ÌĹIJİ IJłĻĴłĻ IJÌ ĞĢĻĨĽĞĻĨIJĮğ ĞĮÂ ĻĞĹĹĦĻĽ
    ﬠ ĢĞĬĢłĬĞĻĦ ĽĞĬłĦĽ ĞĬIJĮĹ ÌĨĹĽĻ ĞŃĨĽ
    ĴĹĦÂĨĢĻĨIJĮĽ ł ĮĴgĞĹĹİĞŃâĬIJĽĽçĞĢĻĨĽĞĻĨIJĮgIJłĻĴłĻd ĞŃĨĽłĝã
    ĨÌ ĬĦĮâÑgĽÎĞĴĦã łł ğf
        Ñ ł ĮĴgĞĹĹİĞŃâÑd ĞŃĨĽłĝã
    ĞĢĢłĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦÂĨĢĻĨIJĮĽłłÑã

    ĨÌ ĮIJĻ ĦĴIJĢÎ ķ ĝﬠﬠf
        ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
            ÌďĞĢĢf ăĞĢĢłĹĞĢÑfgġÌåd ď κ
            ÌďĬIJĽĽf ăĬIJĽĽfgġÌåd ď κ
            ÌďĬĹf ăIJĴĻĨİĨŅĦĹgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦåďã
```

```
ńı BĞĢĪŃĞĹĤ ĴĞĽĽ
ĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻd Ñã
ĤĦĮĽĦğgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨĿĞĻĨIJĮgĤĨĮĴŁĻĽã
ĞĢĻĨĿĞĻĨIJĮĝgĠĞĢĪŃĞĹĤâĤĦĮĽĦğgĤĨĮĴŁĻĽã
ĤĦĮĽĦĝgĠĞĢĪŃĞĹĤâĞĢĻĨĿĞĻĨIJĮĝgĤĨĮĴŁĻĽã

ńı UĴĤĞĻĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
IJĴĻĨĬĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĽâã
IJĴĻĨĬĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞĬĽâĤĦĮĽĦĝã
IJĴĻĨĬĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞĬĽâĤĦĮĽĦğã
IJĴĻĨĬĨŅĦĹgĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞĬĽâã
```

ńńń

```
ĦĴIJĢÎf fld ĞĢĢf flgġħfld ĬIJĽĽf ĝgflîîd ĬĹf ĝgfl
ĦĴIJĢÎf ĝflfld ĞĢĢf flgǵflfld ĬIJĽĽf ĝgfllíd ĬĹf flgîflîîĝíĝflìġìflġġħì
ĦĴIJĢÎf ğflfld ĞĢĢf flgǵğġd ĬIJĽĽf ĝgflìíd ĬĹf flgíġǵflğíġĤħîħǵĝġħì
ggg
ĦĴIJĢÎf ĝìflfld ĞĢĢf flgǵĤfld ĬIJĽĽf ĝgflğĤd ĬĹf flgġìflĤflìĤîĤǵflĤìflĤí
ĦĴIJĢÎf ĝíflfld ĞĢĢf flgǵìfld ĬIJĽĽf ĝgflĝìd ĬĹf flgġĤìğìflǵĤġìġġǵìħğì
ĦĴIJĢÎf ĝîflfld ĞĢĢf flgǵħfld ĬIJĽĽf ĝgflflíd ĬĹf flgġǵǵîǵħĤġġğíìġǵflĤ
ĦĴIJĢÎf ğflflfld ĞĢĢf flgǵħġd ĬIJĽĽf ĝgflflfld ĬĹf flgġġġǵǵǵǵíĝǵîġíġĝğì
ĦĴIJĢÎf ğĝflfld ĞĢĢf flgǵîfld ĬIJĽĽf ĝgflflĤd ĬĹf flgġğğħíǵìġìflĝĝîĝîġǵ
ggg
ĦĴIJĢÎf ġğflfld ĞĢĢf flgǵîġd ĬIJĽĽf flgîíġd ĬĹf flgğġíĝĤĝîǵflîġíġĝíħĤ
ggg
ĦĴIJĢÎf Ĥflflfld ĞĢĢf flgĤììd ĬIJĽĽf flgîflfld ĬĹf flgĝħħîǵǵîflìǵíǵĤíğ
ggg
ĦĴIJĢÎf ħflflfld ĞĢĢf flgħġġd ĬIJĽĽf flgíħfld ĬĹf flgĝǵğíììĤĤġîġħğìħħ
ggg
ĦĴIJĢÎf íflflfld ĞĢĢf flgħǵìd ĬIJĽĽf flgìîîd ĬĹf flgĝĝĝĝğġǵĤíĝħğflĝìîî
ggg
ĦĴIJĢÎf îíflfld ĞĢĢf flgħħġd ĬIJĽĽf flgììġd ĬĹf flgflîğħflĝĝħħììǵìflĝġì
ĦĴIJĢÎf îîflfld ĞĢĢf flgħħġd ĬIJĽĽf flgììğd ĬĹf flgflîĝìĤĝĤġħíġíğǵğflğ
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgħħìd ĬIJĽĽf flgììĝd ĬĹf flgflîflîĝìġĤħĝğǵğíġîġ
```

Fig 10.17: Model training with SGD optimizer and more proper learning rate decay.

Epilepsy Warning (quick flashing colors)

Anim 10.17: https://nnfs.io/muk

In this case, we've achieved our lowest loss and highest accuracy thus far, but it still should be possible to find parameters that will give us even better results. For example, you may suspect that the initial learning rate is too high. It can make for a great exercise to attempt to find better settings. Feel free to try!

Stochastic Gradient Descent with learning rate decay can do fairly well but is still a fairly basic optimization method that only follows a gradient without any additional logic that could potentially help the model find the **global minimum** to the loss function. One option for improving the SGD optimizer is to introduce **momentum**.

Stochastic Gradient Descent with Momentum

Momentum creates a rolling average of gradients over some number of updates and uses this average with the unique gradient at each step. Another way of understanding this is to imagine a ball going down a hill — even if it finds a small hole or hill, momentum will let it go straight through it towards a lower minimum — the bottom of this hill. This can help in cases where you're stuck in some local minimum (a hole), bouncing back and forth. With momentum, a model is more likely to pass through local minimums, further decreasing loss. Simply put, momentum may still point towards the global gradient descent direction.

Recall this situation from the beginning of this chapter:

With regular updates, the SGD optimizer might determine that the next best step is one that keeps the model in a local minimum. Remember that the gradient points toward the current steepest loss ascent for that step — taking the negative of the gradient vector flips it toward the current steepest descent, which may not necessarily follow descent towards the global minimum — the current steepest descent may point towards a local minimum. So this step may decrease loss for that update but might not get us out of the local minimum. We might wind up with a gradient

that points in one direction and then the opposite direction in the next update; the gradient could continue to bounce back and forth around a local minimum like this, keeping the optimization of the loss stuck. Instead, momentum uses the previous update's direction to influence the next update's direction, minimizing the chances of bouncing around and getting stuck.

Recall another example shown in this chapter:

We utilize momentum by setting a parameter between 0 and 1, representing the fraction of the previous parameter update to retain, and subtracting (adding the negative) our actual gradient, multiplied by the learning rate (like before), from it. The update contains a portion of the gradient from preceding steps as our momentum (direction of previous changes) and only a portion of the current gradient; together, these portions form the actual change to our parameters and the bigger the role that momentum takes in the update, the slower the update can change the direction. When we set the momentum fraction too high, the model might stop learning at all since the direction of the updates won't be able to follow the global gradient descent. The code for this is as follows:

```
ŃĦĨÍÎĻçŁĴĤĞĻĦĹ ł ĹĦĬÌgİIJİĦĮĻŁİ ǻ ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻŁİĹ h ċ
                ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ǻ ĬĞÑĦĹgĤŃĦĨÍÎĻĹ
```

The hyperparameter, `ĹĦĬÌgİIJİĦĮĻŁİ`, is chosen at the start and the `ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻŁİĹ` start as all zeros but are altered during training as:

```
ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻŁİĹ ł ŃĦĨÍÎĻçŁĴĤĞĻĦĹ
```

This means that the momentum is always the previous update to the parameters. We will perform the same operations as the above with the biases. We can then update our SGD optimizer class' łĴĤĞĻĦçĴĞĹĞİĽ method with the momentum calculation, applying with the parameters, and retaining them for the next steps as an alternative chain of operations to the current code. The difference is that we only calculate the updates and we add these updates with the common code:

```
# UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
ÂĦÌ łĴĤĞĻĦçĴĞĹĞİĽâĹĦĬÌd ĬĞÑĦĹãf

    # IÌ ŃĦ łĽĦ İIJİĦĮĻłİ
    ĨÌ ĽĦĬÌgİIJİĦĮĻłİf

        # IÌ ĬĞÑĦĹ ÂIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ İIJİĦĮĻłİ ĞĹĹĞÑĽd ĢĹĦĞĻĦ ĻĨĦİ
        # ÌĨĬĬĦÂ ŃĨĻĨ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ĨĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍĨĻçİIJİĦĮĻłİĽďãf
            ĬĞÑĦĹgŃĦĨÍĨĻçİIJİĦĮĻłİĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨÍĨĻĽã
            # IÌ ĻĨĦĹĦ ĨĽ ĮIJ İIJİĦĮĻłİ ĞĹĹĞÑ ÌIJĹ ŃĦĨÍĨĻĽ
            # TĨĦ ĞĹĹĞÑ ÂIJĦĽĮďĻ ĦŃĨĽĻ ÌIJĹ ĠĨĞĽĦĽ ÑĦĻ ĦĨĻĨĦĹg
            ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

        # BłĨĬÂ ŃĦĨÍĨĻ łĴĤĞĻĦĽ ŃĨĻĨ İIJİĦĮĻłİ h ĻĞĨĦ ĴĹĦĽĨIJłĽ
        # łĴĤĞĻĦĽ İłĬĻĨĴĨĬĦÂ ĠÑ ĹĦĻĞĨĮ ÌĞĢĻIJĹ ĞĮÂ łĴĤĞĻĦ ŃĨĻĨ
        # ĢłĹĹĦĮĻ ÍĹĞÂĨĦĮĻĽ
        ŃĦĨÍĨĻçłĴĤĞĻĦĽ ł ċ
            ĽĦĬÌgİIJİĦĮĻłİ ő ĬĞÑĦĹgŃĦĨÍĨĻçİIJİĦĮĻłİĽ h ċ
            ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ő ĬĞÑĦĹgÂŃĦĨÍĨĻĽ
        ĬĞÑĦĹgŃĦĨÍĨĻçİIJİĦĮĻłİĽ ł ŃĦĨÍĨĻçłĴĤĞĻĦĽ

        # BłĨĬÂ ĠĨĞĽ łĴĤĞĻĦĽ
        ĠĨĞĽçłĴĤĞĻĦĽ ł ċ
            ĽĦĬÌgİIJİĦĮĻłİ ő ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ h ċ
            ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ő ĬĞÑĦĹgÂĠĨĞĽĦĽ
        ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ ł ĠĨĞĽçłĴĤĞĻĦĽ

    # VĞĮĨĬĬĞ SGD łĴĤĞĻĦĽ âĞĽ ĠĦÌIJĹĦ İIJİĦĮĻłİ łĴĤĞĻĦã
    ĦĬĽĦf
        ŃĦĨÍĨĻçłĴĤĞĻĦĽ ł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ő ċ
                            ĬĞÑĦĹgÂŃĦĨÍĨĻĽ
        ĠĨĞĽçłĴĤĞĻĦĽ ł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ő ċ
                           ĬĞÑĦĹgÂĠĨĞĽĦĽ

    # UĴĤĞĻĦ ŃĦĨÍĨĻĽ ĞĮÂ ĠĨĞĽĦĽ łĽĨĮÍ ĦĨĻĨĦĹ
    # LĞĮĨĬĬĞ IJĹ İIJİĦĮĻłİ łĴĤĞĻĦĽ
    ĬĞÑĦĹgŃĦĨÍĨĻĽ κł ŃĦĨÍĨĻçłĴĤĞĻĦĽ
    ĬĞÑĦĹgĠĨĞĽĦĽ κł ĠĨĞĽçłĴĤĞĻĦĽ
```

Making our full SGD optimizer class:

```
# SGD IJĴĻĨĬĨŅĦĹ
ĢĬĞĽĹ OĴĻĨĬĨŅĦĹçSGDf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨĬĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĺĽd
    # ĬĦĞĹĮĨĮĺ ĹĞĻĦ IJĬ ĝg ĨĽ ĤĦÌĞŁĬĻ ÌIJĹ ĻĴĨĽ IJĴĻĨĬĨŅĦĹ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮĺçĹĞĻĦłĝgd ĤĦĢĞÑłflgd ĬIJĬĦĮĻŁĬłflgãf
        ĽĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĽĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĬIJĬĦĮĻŁĬ ł ĬIJĬĦĮĻŁĬ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĽâĹĦĬÌãf
        ĨÌ ĽĦĬÌgĤĦĢĞÑf
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĽĦĬÌgĤĦĢĞÑ ắ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    # UĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĽ
    ĤĦÌ ŁĴĤĞĻĦçĴĞĹĞĬĽâĹĦĬÌd ĬĞÑĦĹãf

        # IÌ ŃĦ ŁĽĦ ĬIJĬĦĮĻŁĬ
        ĨÌ ĽĦĬÌgĬIJĬĦĮĻŁĬf

            # IÌ ĬĞÑĦĹ ĤIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĬIJĬĦĮĻŁĬ ĞĹĹĞÑĽd ĢĹĦĞĻĦ ĻĨĦĬ
            # ÌĨĬĬĦĤ ŃĨĻĨ ŅĦĹIJĽ
            ĨÌ ĮIJĻ ĨĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĺĨĻçĬIJĬĦĮĻŁĬĽďãf
                ĬĞÑĦĹgŃĦĨĺĨĻçĬIJĬĦĮĻŁĬĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨĺĨĻĽã
                # IÌ ĻĨĦĹĦ ĨĽ ĮIJ ĬIJĬĦĮĻŁĬ ĞĹĹĞÑ ÌIJĹ ŃĦĨĺĨĻĽ
                # TĨĦ ĞĹĹĞÑ ĤIJĦĽĮďĻ ĦŃĨĽĻ ÌIJĹ ĠĨĞĽĦĽ ÑĦĻ ĦĨĻĨĦĹg
                ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻŁĬĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

            # BŁĨĬĤ ŃĦĨĺĨĻ ŁĴĤĞĻĦĽ ŃĨĻĨ ĬIJĬĦĮĻŁĬ h ĻĞĨĦ ĴĹĦŁĨIJŁĽ
            # ŁĴĤĞĻĦĽ ĬŁĬĻĨĴĬĨĦĤ ĠÑ ĹĦĻĞĨĮ ÌĞĢĻIJĹ ĞĮĤ ŁĴĤĞĻĦ ŃĨĻĨ
            # ĢŁĹĹĦĮĻ ĺĹĞĤĨĦĮĻĽ
            ŃĦĨĺĨĻçŁĴĤĞĻĦĽ ł ċ
                ĽĦĬÌgĬIJĬĦĮĻŁĬ ắ ĬĞÑĦĹgŃĦĨĺĨĻçĬIJĬĦĮĻŁĬĽ h ċ
                ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ĬĞÑĦĹgĤŃĦĨĺĨĻĽ
            ĬĞÑĦĹgŃĦĨĺĨĻçĬIJĬĦĮĻŁĬĽ ł ŃĦĨĺĨĻçŁĴĤĞĻĦĽ

            # BŁĨĬĤ ĠĨĞĽ ŁĴĤĞĻĦĽ
            ĠĨĞĽçŁĴĤĞĻĦĽ ł ċ
                ĽĦĬÌgĬIJĬĦĮĻŁĬ ắ ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻŁĬĽ h ċ
                ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ĬĞÑĦĹgĤĠĨĞĽĦĽ
            ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻŁĬĽ ł ĠĨĞĽçŁĴĤĞĻĦĽ
```

```
        ƒ VĞĮĨĬĬĞ SGD ŁĴĤĞĻĦĽ âĞĽ ĠĦÌIJĹĦ ĬIJĬĦĮĻŁĬ ŁĴĤĞĻĦã
        ĦĬĽĦf
            ŃĦĨÍÎĻçŁĴĤĞĻĦĽ ł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                                ĬĞÑĦĹgĤŃĦĨÍÎĻĽ
            ĠĨĞĽçŁĴĤĞĻĦĽ ł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                              ĬĞÑĦĹgĤĠĨĞĽĦĽ

        ƒ UĴĤĞĻĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ ŁĽĨĮÍ ĦĨĻÎĦĹ
        ƒ ĽĞĮĨĬĬĞ IJĹ ĬIJĬĦĮĻŁĬ ŁĴĤĞĻĦĽ
        ĬĞÑĦĹgŃĦĨÍÎĻĽ κł ŃĦĨÍÎĻçŁĴĤĞĻĦĽ
        ĬĞÑĦĹgĠĨĞĽĦĽ κł ĠĨĞĽçŁĴĤĞĻĦĽ

    ƒ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞĬĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ
```

Let's show an example illustrating how adding momentum changes the learning process. Keeping the same starting **learning rate** (1) and **decay** (1e-3) from the previous training attempt and using a momentum of 0.5:

```
ƒ CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞĬĴĬĦĽłĝƒƒd ĢĬĞĽĽĦĽłġã

ƒ CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮĤ ħǵ IJŁĻĴŁĻ ĽĞĬŁĦĽ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħǵã

ƒ CĹĦĞĻĦ RĦLU ĞĢĻĨĽĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦĤ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĽĞĻĨIJĮĝ ł AĢĻĨĽĞĻĨIJĮçRĦLUâã

ƒ CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĬĦ IJŁĻĴŁĻ
ƒ IJÌ ĴĹĦĽĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ġ IJŁĻĴŁĻ ĽĞĬŁĦĽ âIJŁĻĴŁĻ ĽĞĬŁĦĽã
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâħǵd ġã

ƒ CĹĦĞĻĦ SIJÌĻĬĞŃ ĢĬĞĽĽĨÌÌĦĹďĽ ĢIJĬĠĨĮĦĤ ĬIJĽĽ ĞĮĤ ĞĢĻĨĽĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨĽĞĻĨIJĮ ł AĢĻĨĽĞĻĨIJĮçSIJÌĻĬĞŃçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

ƒ CĹĦĞĻĦ IJĴĻĨĬĨŅĦĹ
IJĴĻĨĬĨŅĦĹ ł OĴĻĨĬĨŅĦĹçSGDâĤĦĢĞÑłĝĦhġd ĬIJĬĦĮĻŁĬłflĝhã

ƒ TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝƒƒƒĝãf

    ƒ PĦĹÌIJĹĬ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ĤĞĻĞ ĻĬĹIJŁÍÎ ĻÎĨĽ ĬĞÑĦĹ
    ĤĦĮĽĦĝgÌIJĹŃĞĹĤâXã

    ƒ PĦĹÌIJĹĬ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĬĹIJŁÍÎ ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    ƒ ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĽĻ ĤĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨĽĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĽĦĝgIJŁĻĴŁĻã
```

```
ſı PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĹĹ ĻÎĹIJŁÍÎ ĹĦĢIJĮÂ DĦĮĹĦ ĬĞÑĦĹ
ſı ĻĞĪĦĹ IJŁĻĴŁĻĹ IJÌ ĞĢĻĪĿĞĻĪIJĮ ÌŁĮĢĻĪIJĮ IJÌ ÌĪĹĹĻ ĬĞÑĦĹ ĞĹ ĪĮĴŁĻĹ
ÂĦĮĹĦğgÌIJĹŃĞĹÂâĞĢĻĪĿĞĻĪIJĮĝgIJŁĻĴŁĻã

ſı PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĹĹ ĻÎĹIJŁÍÎ ĻÎĦ ĞĢĻĪĿĞĻĪIJĮwĬIJĹĹ ÌŁĮĢĻĪIJĮ
ſı ĻĞĪĦĹ ĻÎĦ IJŁĻĴŁĻ IJÌ ĹĦĢIJĮÂ ÂĦĮĹĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮÂ ĹĦĻŁĹĮĹ ĬIJĹĹ
ĬIJĹĹ ł ĬIJĹĹçĞĢĻĪĿĞĻĪIJĮgÌIJĹŃĞĹÂâÂĦĮĹĦğgIJŁĻĴŁĻd Ñã

ſı CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĪĿĞĻĪIJĮğ ĞĮÂ ĻĞĹÍĦĻĹ
ſı ĢĞĬĢŁĬĞĻĦ ĿĞĬŁĦĹ ĞĬIJĮÍ ÌĪĹĹĻ ĞÑĪĹ
ĴĹĦÂĪĞĻĪIJĮĹ ł ĮĴgĞĹÍİĞÑâĬIJĹĹçĞĢĻĪĿĞĻĪIJĮgIJŁĻĴŁĻd ĞÑĪĹłĝã
ĪÌ ĬĦĮâÑgĹÎĞĴĦã łł ğf
    Ñ ł ĮĴgĞĹÍİĞÑâÑd ĞÑĪĹłĝã
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦÂĪĞĻĪIJĮĹłłÑã

ĪÌ ĮIJĻ ĦĴIJĢÎ ķ ĝflflf
    ĴĹĪĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
        ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
        ÌďĬIJĹĹf ăĬIJĹĹfgġÌåd ď κ
        ÌďĬĹf ăIJĴĻĪİĪŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĪĮÍçĹĞĻĦåďã

ſı BĞĢĪŃĞĹÂ ĴĞĹĹ
ĬIJĹĹçĞĢĻĪĿĞĻĪIJĮgĠĞĢĪŃĞĹÂâĬIJĹĹçĞĢĻĪĿĞĻĪIJĮgIJŁĻĴŁĻd Ñã
ÂĦĮĹĦğgĠĞĢĪŃĞĹÂâĬIJĹĹçĞĢĻĪĿĞĻĪIJĮgÂĪĮĴŁĻĹã
ĞĢĻĪĿĞĻĪIJĮĝgĠĞĢĪŃĞĹÂâÂĦĮĹĦğgÂĪĮĴŁĻĹã
ÂĦĮĹĦĝgĠĞĢĪŃĞĹÂâĞĢĻĪĿĞĻĪIJĮĝgÂĪĮĴŁĻĹã

ſı UĴÂĞĻĦ ŃĦĪÍÎĻĹ ĞĮÂ ĠĪĞĹĦĹ
IJĴĻĪİĪŅĦĹgĴĹĦçŁĴÂĞĻĦçĴĞĹĞİĹâã
IJĴĻĪİĪŅĦĹgŁĴÂĞĻĦçĴĞĹĞİĹâÂĦĮĹĦĝã
IJĴĻĪİĪŅĦĹgŁĴÂĞĻĦçĴĞĹĞİĹâÂĦĮĹĦğã
IJĴĻĪİĪŅĦĹgĴIJĹĻçŁĴÂĞĻĦçĴĞĹĞİĹâã


ńńń
ĦĴIJĢÎf fld ĞĢĢf flgġħfld ĬIJĹĹf ĝgflîîd ĬĹf ĝgfl
ĦĴIJĢÎf ĝflfld ĞĢĢf flgǵğìd ĬIJĹĹf ĝgflìíd ĬĹf flgîflîîĝíĝflìġìflġġħì
ĦĴIJĢÎf ğflfld ĞĢĢf flgǵğġd ĬIJĹĹf ĝgflìĥd ĬĹf flgíġǵflğíġĥħîħǵĝġħì
ggg
ĦĴIJĢÎf ĝíflfld ĞĢĢf flgǵíġd ĬIJĹĹf flgîìíd ĬĹf flgġĥìğìflǵĥġìġġǵìħğì
ĦĴIJĢÎf ĝîflfld ĞĢĢf flgĥǵìd ĬIJĹĹf flgîíǵd ĬĹf flgġǵǵîǵħĥġğǵílġǵflĥ
ggg
ĦĴIJĢÎf ġĝflfld ĞĢĢf flgĥîġd ĬIJĹĹf flgííġd ĬĹf flgğǵǵîħĝîǵĝîġìflĥìí
ĦĴIJĢÎf ġğflfld ĞĢĢf flgĥìfld ĬIJĹĹf flgíìíd ĬĹf flgğǵíĝĥĝîǵflîġíǵĝíħĥ
ĦĴIJĢÎf ġġflfld ĞĢĢf flgĥħġd ĬIJĹĹf flgíħġd ĬĹf flgğǵğħĝğğǵĥǵflǵĥíğğĥ
ĦĴIJĢÎf ġǵflfld ĞĢĢf flgħflìd ĬIJĹĹf flgíħfld ĬĹf flgğğìġğǵǵîĝîflìğĥĝħĥ
ggg
ĦĴIJĢÎf ǵħflfld ĞĢĢf flgħìfld ĬIJĹĹf flgìħĝd ĬĹf flgĝìíħflġğğğflğĝìííħ
ĦĴIJĢÎf ǵìflfld ĞĢĢf flgħîfld ĬIJĹĹf flgìǵîd ĬĹf flgĝìĥǵħîǵíflĥîǵflíħĥ
```

```
ggg
ĦĴIJĢÎf ħflflfld ĞĢĢf flgìġġd ĬIJĹĹf flgħħĝd ĬĹf flgĝĝĝíììĥĥġîġħĝìħħ
ggg
ĦĴIJĢÎf íflflfld ĞĢĢf flgìħġd ĬIJĹĹf flgĥíħd ĬĹf flgĝĝĝĝĝĝĝĥíĝħĝflĝìîî
ggg
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgíflfld ĬIJĹĹf flgĥġîd ĬĹf flgflîflîĝìġĥħĝĝĝĝíġîġ
```

Fig 10.18: Model training with SGD optimizer, learning rate decay and Momentum.

Epilepsy Warning (quick flashing colors)

Anim 10.18: https://nnfs.io/ram

The model achieved the lowest loss and highest accuracy that we've seen so far, but can we do even better? Sure we can! Let's try to set the momentum to 0.9:

```
ſi CĹĦĞĻĦ ÂĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçÂĞĻĞâĽĞİĴĬĦĽłĝſſſd ĢĬĞĽĽĦĽłġã

ſi CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴłĻ ÌĦĞĻłĹĦĽ ĞĮÂ ħǵ IJłĻĴłĻ L·ĞĬłĦĽ
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħǵã

ſi CĹĦĞĻĦ RĦLU ĞĢĻĨL·ĞĻĨIJĮ âĻIJ ĠĦ łĽĦÂ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨL·ĞĻĨIJĮĝ ł AĢĻĨL·ĞĻĨIJĮçRĦLUâã

ſi CĹĦĞĻĦ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴłĻ ÌĦĞĻłĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJłĻĴłĻ
ſi IJÌ ĴĹĦL·ĨIJłĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮÂ ġ IJłĻĴłĻ L·ĞĬłĦĽ âIJłĻĴłĻ L·ĞĬłĦĽã
ÂĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâħǵd ġã

ſi CĹĦĞĻĦ SIJÌĻİĞŃ ĢĬĞĽĽĨÌĨĦĹďĽ ĢIJİĠĨĮĦÂ ĬIJĽĽ ĞĮÂ ĞĢĻĨL·ĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨL·ĞĻĨIJĮ ł AĢĻĨL·ĞĻĨIJĮçSIJÌĻİĞŃçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

ſi CĹĦĞĻĦ IJĴĻĨİĨĨĢĦĹ
IJĴĻĨİĨĨĢĦĹ ł OĴĻĨİĨĨĢĦĹçSGDâĦĦĢĞÑłĝĦhġd İIJİĦĮĻłİłſlgîã

ſi TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝſſſſĝãf

    ſi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ IJÌ IJłĹ ĻĹĞĨĮĨĮÍ ÂĞĻĞ ĻÎĹIJłÍÎ ĻÎĨĽ ĬĞÑĦĹ
    ÂĦĮĽĦĝgÌIJĹŃĞĹÂâXã

    ſi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĞĢĻĨL·ĞĻĨIJĮ ÌłĮĢĻĨIJĮ
    ſi ĻĞĨĦĽ ĻÎĦ IJłĻĴłĻ IJÌ ÌĨĹĽĻ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨL·ĞĻĨIJĮĝgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJłĻĴłĻã

    ſi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ
    ſi ĻĞĨĦĽ IJłĻĴłĻĽ IJÌ ĞĢĻĨL·ĞĻĨIJĮ ÌłĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴłĻĽ
    ÂĦĮĽĦğgÌIJĹŃĞĹÂâĞĢĻĨL·ĞĻĨIJĮĝgIJłĻĴłĻã

    ſi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJłÍÎ ĻÎĦ ĞĢĻĨL·ĞĻĨIJĮwĬIJĽĽ ÌłĮĢĻĨIJĮ
    ſi ĻĞĨĦĽ ĻÎĦ IJłĻĴłĻ IJÌ ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮÂ ĹĦĻłĹĮĽ ĬIJĽĽ
    ĬIJĽĽ ł ĬIJĽĽçĞĢĻĨL·ĞĻĨIJĮgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJłĻĴłĻd Ñã

    ſi CĞĬĢłĬĞĻĦ ĞĢĢłĹĞĢÑ ÌĹIJİ IJłĻĴłĻ IJÌ ĞĢĻĨL·ĞĻĨIJĮğ ĞĮÂ ĻĞĹĹĦĻĽ
    ſi ĢĞĬĢłĬĞĻĦ L·ĞĬłĦĽ ĞĬIJĮĹ ÌĨĹĽĻ ĞÑĨĽ
    ĴĹĦÂĨĞĻĨIJĮĽ ł ĮĴgĞĹĹİĞÑâĬIJĽĽçĞĢĻĨL·ĞĻĨIJĮgIJłĻĴłĻd ĞÑĨĽłĝã
    ĞĢĢłĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦÂĨĞĻĨIJĮĽłłÑã
```

```
ÎÌ ĮIJĻ ĦĴIJĢÎ ķ ĝflflf
    ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
          ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
          ÌďĬIJĹĹf ăĬIJĹĹfgġÌåd ď κ
          ÌďĬĹf ăIJĴĻĨÌĨÑĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

ﬁ BĞĢĪŃĞĹÂ ĴĞĹĹ
ĬIJĹĹçĞĢĻĨŁĞĻĨIJĮgĠĞĢĪŃĞĹÂâĬIJĹĹçĞĢĻĨŁĞĻĨIJĮgIJŁĻĴŁĻd Ñã
ÂĦĮĹĦğgĠĞĢĪŃĞĹÂâĬIJĹĹçĞĢĻĨŁĞĻĨIJĮgÂĨĮĴŁĻĹã
ĞĢĻĨŁĞĻĨIJĮĝgĠĞĢĪŃĞĹÂâÂĦĮĹĦğgÂĨĮĴŁĻĹã
ÂĦĮĹĦĝgĠĞĢĪŃĞĹÂâĞĢĻĨŁĞĻĨIJĮĝgÂĨĮĴŁĻĹã

ﬁ UĴÂĞĻĦ ŃĦĨÍÎĻĹ ĞĮÂ ĠĨĞĹĦĹ
IJĴĻĨÌĨÑĦĹgĴĹĦçŁĴÂĞĻĦçĴĞĹĞÌĹâã
IJĴĻĨÌĨÑĦĹgŁĴÂĞĻĦçĴĞĹĞÌĹâÂĦĮĹĦĝã
IJĴĻĨÌĨÑĦĹgŁĴÂĞĻĦçĴĞĹĞÌĹâÂĦĮĹĦğã
IJĴĻĨÌĨÑĦĹgĴIJĹĻçŁĴÂĞĻĦçĴĞĹĞÌĹâã


ńńń
ĦĴIJĢÎf fld ĞĢĢf flgġħfld ĬIJĹĹf ĝgflîîd ĬĹf ĝgfl
ĦĴIJĢÎf ĝflfld ĞĢĢf flgġġġd ĬIJĹĹf ĝgflĥġd ĬĹf flgîflîîĝíĝflìġìflġġħì
ĦĴIJĢÎf ğflfld ĞĢĢf flgġîìd ĬIJĹĹf flgîîîd ĬĹf flgíġġflğíġĥħîħġĝġħì
ĦĴIJĢÎf ġflfld ĞĢĢf flgħflġd ĬIJĹĹf flgíĝfld ĬĹf flgìħîíğĝîġflìğġħġġħ
ĦĴIJĢÎf ġflfld ĞĢĢf flgìflfld ĬIJĹĹf flgìflfld ĬĹf flgìĝġìîħğíġflĥîġğíĝ
ĦĴIJĢÎf ĥflfld ĞĢĢf flgìĥfld ĬIJĹĹf flgĥîĥd ĬĹf flgħħìĝĝĝġflìħflĥflì
ĦĴIJĢÎf ħflfld ĞĢĢf flgíĝfld ĬIJĹĹf flgġîħd ĬĹf flgħğĥġîflíħîğîġġflíġ
ĦĴIJĢÎf ìflfld ĞĢĢf flgíĝfld ĬIJĹĹf flgġħħd ĬĹf flgĥííĥíĝĥĝíĥġflġĝìí
ĦĴIJĢÎf íflfld ĞĢĢf flgíġìd ĬIJĹĹf flgġíġd ĬĹf flgĥĥĥíħġġħîflîġîġĝĝ
ĦĴIJĢÎf îflfld ĞĢĢf flgíĥfld ĬIJĹĹf flgġħġd ĬĹf flgĥğħĥîğîġġħĥġĥĥĥ
ĦĴIJĢÎf ĝflflfld ĞĢĢf flgííìd ĬIJĹĹf flgġġġd ĬĹf flgĥflflğĥflĝğĥflħğĥġĝğ
ggg
ĦĴIJĢÎf ğğflfld ĞĢĢf flgîflfld ĬIJĹĹf flgğġğd ĬĹf flgġĝğĥîìħíħìììĝĝìíħ
ggg
ĦĴIJĢÎf ğîflfld ĞĢĢf flgîĝfld ĬIJĹĹf flgğĝħd ĬĹf flgğĥħġìħflĝîġíğĝììġħ
ggg
ĦĴIJĢÎf ġíflfld ĞĢĢf flgîğfld ĬIJĹĹf flgğflğd ĬĹf flgğflíġìħìġĥĝĥĥğġflħí
ggg
ĦĴIJĢÎf ìĝflfld ĞĢĢf flgîġfld ĬIJĹĹf flgĝíĝd ĬĹf flgĝğġġìğflġġĥíġġîġĝġ
ggg
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgîġġd ĬIJĹĹf flgĝìġd ĬĹf flgflîflîĝìġĥħĝğġğíġîġ
```

Fig 10.19: Model training with SGD optimizer, learning rate decay and Momentum (tuned).

Epilepsy Warning (quick flashing colors)

Anim 10.19: https://nnfs.io/map

This is a decent enough example of how momentum can prove useful. The model achieved an accuracy of almost 88% in the first 1000 epochs and improved further, ending with an accuracy of 93.3% and a loss of 0.173. These results are a great improvement. The SGD optimizer with momentum is usually one of 2 main choices for an optimizer in practice next to the Adam optimizer, which we'll talk about shortly. First, we have 2 other optimizers to talk about. The next modification to Stochastic Gradient Descent is **AdaGrad**.

AdaGrad

AdaGrad, short for **adaptive gradient**, institutes a per-parameter learning rate rather than a globally-shared rate. The idea here is to normalize updates made to the features. During the training process, some weights can rise significantly, while others tend to not change by much. It is usually better for weights to not rise too high compared to the other weights, and we'll talk about this with regularization techniques. AdaGrad provides a way to normalize parameter updates by keeping a history of previous updates — the bigger the sum of the updates is, in either direction (positive or negative), the smaller updates are made further in training. This lets less-frequently updated parameters to keep-up with changes, effectively utilizing more neurons for training. The concept of AdaGrad can be contained in the following two lines of code:

```
ĢĞĢÎĦ κł ĴĞĹİçÍĹĞĤĨĦĮĻ ăă ğ
ĴĞĹİçŁĴĤĞĻĦĽ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ ă ĴĞĹİçÍĹĞĤĨĦĮĻ w âĽĶĹĻâĢĞĢÎĦã κ ĦĴĽã
```

The `ĢĞĢÎĦ` holds a history of squared gradients, and the `ĴĞĹİçŁĴĤĞĻĦĽ` is a function of the learning rate multiplied by the gradient (basic SGD so far) and then is divided by the square root of the cache plus some **epsilon** value. The division operation performed with a constantly rising cache might also cause the learning to stall as updates become smaller with time, due to the monotonic nature of updates. That's why this optimizer is not widely used, except for some specific applications. The **epsilon** is a **hyperparameter** (pre-training control knob setting) preventing division by 0. The epsilon value is usually a small value, such as *1e-7*, which we'll be defaulting to. You might also notice that we are summing the squared value, only to calculate the square root later, which might look counter-intuitive as to why we do this. We are adding squared values and taking the square root, which is not the same as just adding the value, for example:

$$\sqrt{1^2 + 3^2} = \sqrt{1 + 9} = \sqrt{10} \approx 2.16$$

$$1 + 3 = 4$$

The resulting cache value grows slower, and in a different way, taking care of the negative numbers (we would not want to divide the update by the negative number and flip its sign). Overall, the impact is the learning rates for parameters with smaller gradients are decreased slowly, while the parameters with larger gradients have their learning rates decreased faster.

To implement AdaGrad, we start by copying and pasting our SGD optimizer class, changing the name, adding a property for **epsilon** with a default of 1e-7 to the ççĨĮĨĻçç method, and removing the momentum. Next, inside the ŁĴĤĞĻĦçĴĞĹĞİĽ method, we'll replace the momentum code with:

```
ń UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
ÎĦÌ ŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

    ń IÌ ĬĞÑĦĹ ĤIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢÎĦ ĞĹĹĞÑĽd
    ń ĢĹĦĞĻĦ ĻÎĦİ ÌĨĬĬĦĤ ŃĨĻÎ ŅĦĹIJĽ
    ĨÌ ĮIJĻ ÎĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍÎĻçĢĞĢÎĦďãf
        ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgŃĦĨÍÎĻĽã
        ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgĠĨĞĽĦĽã

    ń UĴĤĞĻĦ ĢĞĢÎĦ ŃĨĻÎ ĽĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ÍĹĞĤĨĦĮĻĽ
    ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ κł ĬĞÑĦĹgĤŃĦĨÍÎĻĽááğ
    ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ κł ĬĞÑĦĹgĤĠĨĞĽĦĽááğ

    ń VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ IJĹİĞĬĨŅĞĻĨIJĮ
    ń ŃĨĻÎ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢÎĦ
    ĬĞÑĦĹgŃĦĨÍÎĻĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ á ċ
                     ĬĞÑĦĹgĤŃĦĨÍÎĻĽ w ċ
                     âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
    ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ á ċ
                    ĬĞÑĦĹgĤĠĨĞĽĦĽ w ċ
                    âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
```

We added the cache and its updates, then added dividing the updates by the square root of the cache. Full code for the AdaGrad optimizer:

```
ń AĤĞÍĹĞĤ IJĴĻĨİĨŅĦĹ
ĢĬĞĽĽ OĴĻĨİĨŅĦĹçAĤĞÍĹĞĤf

    ń IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮÍĽ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ĤĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìãf
        ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ

    ń CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgĤĦĢĞÑf
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ á ċ
                âĝg w âĝg κ ĽĦĬÌgĤĦĢĞÑ á ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã
```

```
ƒ UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
ĤĦÌ ŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

    ƒ IÌ ĬĞÑĦĹ ĤIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĴĦ ĞĹĹĞÑĽd
    ƒ ĢĹĦĞĻĦ ĻĴĦİ ÌĨĬĬĦĤ ŃĨĻĴ ŅĦĹIJĽ
    ĨÌ ĮIJĻ ĴĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĬĴĻçĢĞĢĴĦďãf
        ĬĞÑĦĹgŃĦĨĬĴĻçĢĞĢĴĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨĬĴĻĽã
        ĬĞÑĦĹgĠĨĞĽçĢĞĢĴĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

    ƒ UĴĤĞĻĦ ĢĞĢĴĦ ŃĨĻĴ ĽĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ĬĹĞĤĨĦĮĻĽ
    ĬĞÑĦĹgŃĦĨĬĴĻçĢĞĢĴĦ κł ĬĞÑĦĹgĤŃĦĨĬĴĻĽăăğ
    ĬĞÑĦĹgĠĨĞĽçĢĞĢĴĦ κł ĬĞÑĦĹgĤĠĨĞĽĦĽăăğ

    ƒ VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹİĞĬĨŅĞĻĨIJĮ
    ƒ ŃĨĻĴ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĴĦ
    ĬĞÑĦĹgŃĦĨĬĴĻĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ă ċ
                     ĬĞÑĦĹgĤŃĦĨĬĴĻĽ w ċ
                     âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨĬĴĻçĢĞĢĴĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
    ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ă ċ
                    ĬĞÑĦĹgĤĠĨĞĽĦĽ w ċ
                    âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢĴĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã

ƒ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
ĤĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
    ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ
```

Testing this optimizer now with decaying set to *1e-4* as well as *1e-5* works better than *1e-3*, which we have used previously. This optimizer with our dataset works better with lesser decaying:

```
ƒ CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflld ĢĬĞĽĽĦĽłġã

ƒ CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻĴ ğ ĨĮĴłĻ ÌĦĞĻłĹĦĽ ĞĮĤ ħǵ IJłĻĴłĻ ŁĞĬłĦĽ
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâğd ħǵã

ƒ CĹĦĞĻĦ RĦLU ĞĢĻĨŁĞĻĨIJĮ âĻIJ ĠĦ łĽĦĤ ŃĨĻĴ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨŁĞĻĨIJĮĝ ł AĢĻĨŁĞĻĨIJĮçRĦLUâã

ƒ CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻĴ ħǵ ĨĮĴłĻ ÌĦĞĻłĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJłĻĴłĻ
ƒ IJÌ ĴĹĦŁĨIJłĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ġ IJłĻĴłĻ ŁĞĬłĦĽ âIJłĻĴłĻ ŁĞĬłĦĽã
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâħǵd ġã

ƒ CĹĦĞĻĦ SIJÌĻİĞŇ ĢĬĞĽĽĨÌÌĦĹďĽ ĢIJİĠĨĮĦĤ ĬIJĽĽ ĞĮĤ ĞĢĻĨŁĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨŁĞĻĨIJĮ ł AĢĻĨŁĞĻĨIJĮçSIJÌĻİĞŇçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

ƒ CĹĦĞĻĦ IJĴĻĨÌĨŅĦĹ
ƒIJĴĻĨÌĨŅĦĹ ł OĴĻĨÌĨŅĦĹçSGDâĤĦĢĞÑłíĦhíd ÌIJİĦĮĻłİłflgîã
```

```
IJĴĻĨĬĨŅĦĹ ł OĴĻĨĬĨŅĦĹçAĤĞĬĹĞĤâĤĦĢĞÑłĝĦhǵã
fi TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢĴ ĨĮ ĹĞĮĬĦâĝflflflĝãf

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮĬ ĤĞĻĞ ĻĨĹIJŁĬĨ ĻĨĨĽ ĬĞÑĦĹ
    ĤĦĮĽĦĝgÌIJĹŃĞĹĤâXã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĨĹIJŁĬĨ ĞĢĻĨL·ĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    fi ĻĞĨĦĹ ĻĨĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ĤĦĮĽĦ ĬĞÑĦĹ ĨĦĹĦ
    ĞĢĻĨL·ĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĽĦĝgIJŁĻĴŁĻã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĨĹIJŁĬĨ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ
    fi ĻĞĨĦĹ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨL·ĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
    ĤĦĮĽĦğgÌIJĹŃĞĹĤâĞĢĻĨL·ĞĻĨIJĮĝgIJŁĻĴŁĻã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĨĹIJŁĬĨ ĻĨĦ ĞĢĻĨL·ĞĻĨIJĮwĬIJĽĽ ÌŁĮĢĻĨIJĮ
    fi ĻĞĨĦĹ ĻĨĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮĤ ĤĦĮĽĦ ĬĞÑĦĹ ĨĦĹĦ ĞĮĤ ĹĦĻŁĹĮĽ ĬIJĽĽ
    ĬIJĽĽ ł ĬIJĽĽçĞĢĻĨL·ĞĻĨIJĮgÌIJĹŃĞĹĤâĤĦĮĽĦğgIJŁĻĴŁĻd Ñã

    fi CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨL·ĞĻĨIJĮğ ĞĮĤ ĻĞĹĬĦĻĽ
    fi ĢĞĬĢŁĬĞĻĦ L·ĞĬŁĦĹ ĞĬIJĮĬ ÌĨĹĽĻ ĞÑĨĽ
    ĴĹĦĤĨĞĻĨIJĮĽ ł ĮĴgĞĹĬİĞÑâĬIJĽĽçĞĢĻĨL·ĞĻĨIJĮgIJŁĻĴŁĻd ĞÑĨĽłĝã
    ĨÌ ĬĦĮâÑgĽĬĞĴĦã łł ğf
        Ñ ł ĮĴgĞĹĬİĞÑâÑd ĞÑĨĽłĝã
    ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦĤĨĞĻĨIJĮĽłłÑã

    ĨÌ ĮIJĻ ĦĴIJĢĴ ķ ĝflflf
        ĴĹĨĮĻâÌdĦĴIJĢĴf ăĦĴIJĢĴåd ď κ
              ÌdĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌdĬIJĽĽf ăĬIJĽĽfgġÌåd ď κ
              ÌdĬĹf ăIJĴĻĨĬĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦåďã

    fi BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĬIJĽĽçĞĢĻĨL·ĞĻĨIJĮgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨL·ĞĻĨIJĮgIJŁĻĴŁĻd Ñã
    ĤĦĮĽĦğgĠĞĢĪŃĞĹĤâĬIJĽĽçĞĢĻĨL·ĞĻĨIJĮgĤĨĮĴŁĻĽã
    ĞĢĻĨL·ĞĻĨIJĮĝgĠĞĢĪŃĞĹĤâĤĦĮĽĦğgĤĨĮĴŁĻĽã
    ĤĦĮĽĦĝgĠĞĢĪŃĞĹĤâĞĢĻĨL·ĞĻĨIJĮĝgĤĨĮĴŁĻĽã

    fi UĴĤĞĻĦ ŃĦĨĬĬĻĽ ĞĮĤ ĠĨĞĽĦĽ
    IJĴĻĨĬĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĽâã
    IJĴĻĨĬĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞĬĽâĤĦĮĽĦĝã
    IJĴĻĨĬĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞĬĽâĤĦĮĽĦğã
    IJĴĻĨĬĨŅĦĹgĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞĬĽâã


ńńń
ĦĴIJĢĴf fld ĞĢĢf flgġħfld ĬIJĽĽf ĝgflîid ĬĹf ĝgfl
ĦĴIJĢĴf ĝflfld ĞĢĢf flgǵĥìd ĬIJĽĽf ĝgflĝğd ĬĹf flgîîflĝîìflǵîğĝğìîġġ
ĦĴIJĢĴf ğflfld ĞĢĢf flgĥğìd ĬIJĽĽf flgîġħd ĬĹf flgîíflǵííğíġĝħĥflĝħĝ
```

```
ĦĴIJĢÎf ġflfld ĞĢĢf flgħflfld ĬIJĽĽf flgíìġd ĬĹf flgîìflîħíflĥĥĝĥflîíĥĥ
ggg
ĦĴIJĢÎf ĝğflfld ĞĢĢf flgìflfld ĬIJĽĽf flgħġfld ĬĹf flgíîğîġħíħîġħġġġħ
ggg
ĦĴIJĢÎf ĝìflfld ĞĢĢf flgìĥfld ĬIJĽĽf flgĥìîd ĬĹf flgíĥġììġîĝğġflflĝîħħ
ggg
ĦĴIJĢÎf ġìflfld ĞĢĢf flgíflfld ĬIJĽĽf flgġħġd ĬĹf flgħíflġĝíġíîflflħflĥġí
ggg
ĦĴIJĢÎf ĥĝflfld ĞĢĢf flgíĝfld ĬIJĽĽf flgġĥġd ĬĹf flgħħğĝîĥĥĝħğĥîġĥġî
ggg
ĦĴIJĢÎf ħìflfld ĞĢĢf flgíğfld ĬIJĽĽf flgġğħd ĬĹf flgĥîííġíğĥġìííħĥĝî
ggg
ĦĴIJĢÎf ìĥflfld ĞĢĢf flgíġfld ĬIJĽĽf flgġĝğd ĬĹf flgĥìĝġħĝğğħġĥĥìîĝí
ggg
ĦĴIJĢÎf îîflfld ĞĢĢf flgíġìd ĬIJĽĽf flgġíĝd ĬĹf flgĥflğĥġìíĝĥîìflħĥĝí
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgíġìd ĬIJĽĽf flgġìîd ĬĹf flgĥflflflğĥflflĝğĥflflħğħ
```

Fig 10.20: Model training with AdaGrad optimizer.

Epilepsy Warning (quick flashing colors)

Anim 10.20: https://nnfs.io/bop

AdaGrad worked quite well here, but not as good as SGD with momentum, and we can see that loss consistently fell throughout the entire training process. It is interesting to note that AdaGrad initially took a few more epochs to reach similar results to Stochastic Gradient Descent with momentum.

RMSProp

Continuing with Stochastic Gradient Descent adaptations, we reach **RMSProp**, short for **Root Mean Square Propagation**. Similar to AdaGrad, RMSProp calculates an adaptive learning rate per parameter; it's just calculated in a different way than AdaGrad.

Where AdaGrad calculates the cache as:

```
ĢĞĢÎĦ ĸł ÍĹĞĤĨĦĮĻ ăă ğ
```

RMSProp calculates the cache as:

```
ĢĞĢÎĦ ł ĹÎIJ ă ĢĞĢÎĦ ĸ âĝ h ĹÎIJã ă ÍĹĞĤĨĦĮĻ ăă ğ
```

Note that this is similar to both momentum with the SGD optimizer and cache with the AdaGrad. RMSProp adds a mechanism similar to momentum but also adds a per-parameter adaptive learning rate, so the learning rate changes are smoother. This helps to retain the global direction of changes and slows changes in direction. Instead of continually adding squared gradients to a cache (like in Adagrad), it uses a moving average of the cache. Each update to the cache retains a part of the cache and updates it with a fraction of the new, squared, gradients. In this way, cache contents "move" with data in time, and learning does not stall. In the case of this optimizer, the per-parameter learning rate can either fall or rise, depending on the last updates and current gradient. RMSProp applies the cache in the same way as AdaGrad does.

The new hyperparameter here is *rho*. *Rho* is the cache memory decay rate. Because this optimizer, with default values, carries over so much momentum of gradient and the adaptive learning rate updates, even small gradient updates are enough to keep it going; therefore, a default learning rate of *1* is far too large and causes instant model instability. A learning rate that becomes stable again and gives fast enough updates is around *0.001* (that's also the default value for this optimizer used in well-known machine learning frameworks). That's what we'll use as default from now on too. The following is the full code for RMSProp optimizer class:

```
# RMSPropOptimizer
class OptimizerRMSProp:

    # Initialize optimizer - set attributes
    def __init__(self, learning_rate=1.0,
                 decay=0.0, epsilon=1e-7,
                 rho=0.9):
        self.learning_rate = self.initial_learning_rate
        self.decay = decay
        self.iterations = 0
        self.epsilon = epsilon
        self.rho = rho

    # Call once before any parameter updates
    def pre_update_params(self):
        if self.decay:
            self.current_learning_rate = self.learning_rate
                * (1. / (1. + self.decay * self.iterations))

    # Update parameters
    def update_params(self, layer):

        # If layer does not contain cache arrays,
        # create them filled with zeros
        if hasattr(layer, 'weight_cache'):
            layer.weight_cache = numpy.zeros_like(layer.weights)
            layer.bias_cache = numpy.zeros_like(layer.biases)

        # Update cache with squared current gradients
        layer.weight_cache = self.rho * layer.weight_cache +
            (1 - self.rho) * layer.dweights**2
        layer.bias_cache = self.rho * layer.bias_cache +
            (1 - self.rho) * layer.dbiases**2

        # Vanilla SGD parameter update + normalization
        # with square rooted cache
        layer.weights += -self.learning_rate *
            layer.dweights / (numpy.sqrt(layer.weight_cache) +
            self.epsilon)
        layer.biases += -self.learning_rate *
            layer.dbiases / (numpy.sqrt(layer.bias_cache) +
            self.epsilon)
```

```
        ƒi VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹİĞĬĨŅĞĻĨIJĮ
        ƒi ŃĨĻÎ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢÎĦ
        ĬĞÑĦĹgŃĦĨÍÎĻĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                         ĬĞÑĦĹgĤŃĦĨÍÎĻĽ w ċ
                         âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                        ĬĞÑĦĹgĤĠĨĞĽĦĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã

    ƒi CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãƒ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ
```

Changing the optimizer used in our main neural network testing code:

```
IJĴĻĨĬĨŅĦĹ ł OĴĻĨĬĨŅĦĹçRMSĴĹIJĴâĤĦĢĞÑłĝĦhǵã
```

And running this code gives us:

```
ńńń
ĦĴIJĢÎƒ ƒld ĞĢĢƒ ƒlgǵĥƒld ĬIJĽĹƒ ĝgƒlîîd ĬĹƒ ƒlgƒlƒĝ
ĦĴIJĢÎƒ ĝƒlƒd ĞĢĢƒ ƒlgǵĝìd ĬIJĽĹƒ ĝgƒlììd ĬĹƒ ƒlgƒlƒlƒîîƒlĝîìƒlǵîĝĝĝìîǵĝ
ĦĴIJĢÎƒ ĝƒlƒd ĞĢĢƒ ƒlgǵĤìd ĬIJĽĹƒ ĝgƒlìĝd ĬĹƒ ƒlgƒlƒlƒîíƒlǵííĝíǵĝĝĥĤƒlĝĥĝ
ĦĴIJĢÎƒ ǵƒlƒd ĞĢĢƒ ƒlgǵíƒld ĬIJĽĹƒ ĝgƒlĥĝd ĬĹƒ ƒlgƒlƒlƒîìƒlîĥíƒlĤĥĝĤƒlîíĤĥ
ggg
ĦĴIJĢÎƒ ĝƒlƒlƒd ĞĢĢƒ ƒlgĤîìd ĬIJĽĹƒ ƒlgîĥĝd ĬĹƒ ƒlgƒlƒlƒîƒlîĝìǵĤĥĝĝĝĝíǵîǵ
ggg
ĦĴIJĢÎƒ ǵíƒlƒd ĞĢĢƒ ƒlgìƒlǵd ĬIJĽĹƒ ƒlgìĥìd ĬĹƒ ƒlgƒlƒlƒĥìĤìĝĝǵĝĝĤĝĝĝĥìì
ggg
ĦĴIJĢÎƒ Ĥíƒlƒd ĞĢĢƒ ƒlgìĝǵd ĬIJĽĹƒ ƒlgìǵǵd ĬĹƒ ƒlgƒlƒlƒĥǵĝîĤĝǵĤĝĥĝĝĤíǵí
ggg
ĦĴIJĢÎƒ ìĝƒlƒd ĞĢĢƒ ƒlgìĝƒld ĬIJĽĹƒ ƒlgìĝíd ĬĹƒ ƒlgƒlƒlƒĤíǵíĝîĤĝĝĝîǵĝíƒlǵ
ggg
ĦĴIJĢÎƒ ĝƒlƒlƒlƒd ĞĢĢƒ ƒlgìǵƒld ĬIJĽĹƒ ƒlgĥĥíd ĬĹƒ ƒlgƒlƒlƒĤƒlƒlƒlĝĤƒlƒlĝĝĤƒlƒlƒĥĝĤ
```

Fig 10.21: Model training with RMSProp optimizer.

Epilepsy Warning (quick flashing colors)

Anim 10.21: https://nnfs.io/pun

The results are not the greatest, but we can slightly tweak the hyperparameters:

```
IJĴĻĨĬĨŅĦĹ ł OĴĻĨĬĨŅĦĹçRMSĴĹIJĴâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflğd ĤĦĢĞÑłĝĦhĥd
                                ĹĬIJłflgîîîã


ńńń
ĦĴIJĢĴf fld ĞĢĢf flgġħfld ĬIJĽĽf ĝgflîîd ĬĹf flgflğ
ĦĴIJĢĴf ĝflfld ĞĢĢf flgǵħìd ĬIJĽĽf ĝgflĝǵd ĬĹf flgflĝîîíflğĝîĥíğħĝġğĝ
ĦĴIJĢĴf ğflfld ĞĢĢf flgĥġfld ĬIJĽĽf flgîĥîd ĬĹf flgflĝîîħflğìîflǵġìflĝflǵħ
ggg
ĦĴIJĢĴf ħflfld ĞĢĢf flgħğġd ĬIJĽĽf flgìħğd ĬĹf flgflĝîííflîĝġġğîĝĥíġǵġ
ggg
ĦĴIJĢĴf ĝflflfld ĞĢĢf flgìĝfld ĬIJĽĽf flgħġǵd ĬĹf flgflĝîíflğĝìħğĥîĝìflííǵ
ggg
ĦĴIJĢĴf ĝíflfld ĞĢĢf flgíĝfld ĬIJĽĽf flgǵìĥd ĬĹf flgflĝîħǵħĥĥíǵĝǵĝğîíĝ
ggg
ĦĴIJĢĴf ġíflfld ĞĢĢf flgíĥfld ĬIJĽĽf flgġĥĝd ĬĹf flgflĝîğħíflflíġħğġĝĥħġ
ggg
ĦĴIJĢĴf ħğflfld ĞĢĢf flgíìfld ĬIJĽĽf flgğíħd ĬĹf flgflĝííġğĥħîflǵǵîflħğħġ
ggg
ĦĴIJĢĴf ħħflfld ĞĢĢf flgîflġd ĬIJĽĽf flgğħğd ĬĹf flgflĝíìħĝîflğflíĝħġġflġǵ
ggg
ĦĴIJĢĴf ìĝflfld ĞĢĢf flgîflfld ĬIJĽĽf flgğìǵd ĬĹf flgflĝíħìǵĝĝflħíǵĥflħíĥì
ggg
ĦĴIJĢĴf îĥflfld ĞĢĢf flgíífld ĬIJĽĽf flgğǵǵd ĬĹf flgflĝíğħĥflflħîíħġħĥĝìǵ
ĦĴIJĢĴf îħflfld ĞĢĢf flgííġd ĬIJĽĽf flgğǵĝd ĬĹf flgflĝíğǵíǵǵĝħíĝîǵîħĥǵ
ĦĴIJĢĴf îìflfld ĞĢĢf flgìǵġd ĬIJĽĽf flgìîǵd ĬĹf flgflĝíğġĝìflħìħĝğğíǵĥħ
ĦĴIJĢĴf îíflfld ĞĢĢf flgîĝìd ĬIJĽĽf flgğĝġd ĬĹf flgflĝíğĝĥĝflğĝǵĝĝíĥğĥĥ
ĦĴIJĢĴf îîflfld ĞĢĢf flgîflìd ĬIJĽĽf flgğğĥd ĬĹf flgflĝíĝîíĥğììġîĝflĥîflì
ĦĴIJĢĴf ĝflflflfld ĞĢĢf flgîĝfld ĬIJĽĽf flgğğĝd ĬĹf flgflĝíĝíĝîíġǵìğĥììflğĥ
```

Fig 10.22: Model training with RMSProp optimizer (tuned).

Epilepsy Warning (quick flashing colors)

Anim 10.22: https://nnfs.io/not

Pretty good result, close to SGD with momentum but not as good. We still have one final adaptation to stochastic gradient descent to cover.

Adam

Adam, short for **Adaptive Momentum**, is currently the most widely-used optimizer and is built atop RMSProp, with the momentum concept from SGD added back in. This means that, instead of applying current gradients, we're going to apply momentums like in the SGD optimizer with momentum, then apply a per-weight adaptive learning rate with the cache as done in RMSProp.

The Adam optimizer additionally adds a bias correction mechanism. Do not confuse this with the layer's bias. The bias correction mechanism is applied to the cache and momentum, compensating for the initial zeroed values before they warm up with initial steps. To achieve this correction, both momentum and caches are divided by $1\text{-}beta^{step}$. As step raises, $beta^{step}$ approaches *0* (a fraction to the power of a rising value decreases), solving this whole expression to a fraction during the first steps and approaching *1* as training progresses. For example, *beta 1*, a fraction of momentum to apply, defaults to 0.9. This means that, during the first step, the correction value equals:

$$1 - 0.9^1 = 1 - 0.9 = 0.1$$

With training progression, as step count rises:

$$1 - \lim_{step \to \infty} 0.9^{step} = 1 - 0 = 1$$

The same applies to the cache and the *beta 2* — in this case, the starting value is 0.001 and also approaches *1*. These values divide the momentums and the cache, respectively. Division by a fraction causes them to be multiple times bigger, significantly speeding up training in the initial stages before both tables warm up during multiple initial steps. We also previously mentioned that both of these bias-correcting coefficients go towards a value of *1* as training progresses and return parameter updates to their typical values for the later training steps. To get parameter updates, we divide the scaled momentum by the scaled square-rooted cache.

The code for the Adam Optimizer is based on the RMSProp optimizer. It adds the cache seen from the SGD along with the *beta 1* hyper-parameter. Next, it introduces the bias correction mechanism for both the momentum and the cache. We've also modified the way the parameter updates are calculated — using corrected momentums and corrected caches, instead of gradients and caches. The full list of changes made from RMSProp are posted after the following code:

```
class Adam:
```

```
ĢĬĞĹĹ OĴĻĨĬĨŅĦĹçAĤĞİf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨĬĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮĹĹ
    ÂĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮĹçĹĞĻĦłflgflflĝd ÂĦĢĞÑłflgd ĦĴĹĨĬIJĮłĝĦhìd
                ĠĦĻĞçĝłflgîd ĠĦĻĞçğłflgîîîãf
        ĹĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĹĦĬÌgÂĦĢĞÑ ł ÂĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgĦĴĹĨĬIJĮ ł ĦĴĹĨĬIJĮ
        ĹĦĬÌgĠĦĻĞçĝ ł ĠĦĻĞçĝ
        ĹĦĬÌgĠĦĻĞçğ ł ĠĦĻĞçğ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ łĴÂĞĻĦĹ
    ÂĦÌ ĴĹĦçłĴÂĞĻĦçĴĞĹĞİĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgÂĦĢĞÑf
            ĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĹĦĬÌgÂĦĢĞÑ ắ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    # UĴÂĞĻĦ ĴĞĹĞİĦĻĦĹĹ
    ÂĦÌ łĴÂĞĻĦçĴĞĹĞİĹâĹĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ÂIJĦĹ IJIJ ĢIJĮĻĞĨĮ ĢĞĢĨĦ ĞĹĹĞÑĹd
        # ĢĹĦĞĻĦ ĻĨĦİ ÌĨĬĬĦÂ ÑĨĻĨ ŅĦĹIJĹ
        ĨÌ IJIJ ĨĞĹĞĻĻĹâĬĞÑĦĹd ďÑĦĨĬĨĻçĢĞĢĨĦďãf
            ĬĞÑĦĹgÑĦĨĬĨĻçĬIJĬĦĮĻłĬĹ ł IĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgÑĦĨĬĨĻĹã
            ĬĞÑĦĹgÑĦĨĬĨĻçĢĞĢĨĦ ł IĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgÑĦĨĬĨĻĹã
            ĬĞÑĦĹgĠĨĞĹçĬIJĬĦĮĻłĬĹ ł IĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã
            ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ ł IĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã

        # UĴÂĞĻĦ ĬIJĬĦĮĻłĬ  ÑĨĻĨ ĢłĹĹĦĮĻ ĬĹĞÂĨĦĮĻĹ
        ĬĞÑĦĹgÑĦĨĬĨĻçĬIJĬĦĮĻłĬĹ ł ĹĦĬÌgĠĦĻĞçĝ ắ ċ
                                  ĬĞÑĦĹgÑĦĨĬĨĻçĬIJĬĦĮĻłĬĹ κ ċ
                                  âĝ h ĹĦĬÌgĠĦĻĞçĝã ắ ĬĞÑĦĹgÂÑĦĨĬĨĻĹ
        ĬĞÑĦĹgĠĨĞĹçĬIJĬĦĮĻłĬĹ ł ĹĦĬÌgĠĦĻĞçĝ ắ ċ
                                ĬĞÑĦĹgĠĨĞĹçĬIJĬĦĮĻłĬĹ κ ċ
                                âĝ h ĹĦĬÌgĠĦĻĞçĝã ắ ĬĞÑĦĹgÂĠĨĞĹĦĹ
        # GĦĻ ĢIJĹĹĦĢĻĦÂ ĬIJĬĦĮĻłĬ
        # ĹĦĬÌgĨĻĦĹĞĻĨIJĮ ĨĹ fl ĞĻ ÌĨĹĹĻ ĴĞĹĹ
        # ĞĮÂ ÑĦ ĮĦĦÂ ĻIJ ĹĻĞĹĻ ÑĨĻĨ ĝ ĨĦĹĦ
        ÑĦĨĬĨĻçĬIJĬĦĮĻłĬĹçĢIJĹĹĦĢĻĦÂ ł ĬĞÑĦĹgÑĦĨĬĨĻçĬIJĬĦĮĻłĬĹ w ċ
            âĝ h ĹĦĬÌgĠĦĻĞçĝ ắắ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
        ĠĨĞĹçĬIJĬĦĮĻłĬĹçĢIJĹĹĦĢĻĦÂ ł ĬĞÑĦĹgĠĨĞĹçĬIJĬĦĮĻłĬĹ w ċ
            âĝ h ĹĦĬÌgĠĦĻĞçĝ ắắ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
        # UĴÂĞĻĦ ĢĞĢĨĦ ÑĨĻĨ ĹĶłĞĹĦÂ ĢłĹĹĦĮĻ ĬĹĞÂĨĦĮĻĹ
        ĬĞÑĦĹgÑĦĨĬĨĻçĢĞĢĨĦ ł ĹĦĬÌgĠĦĻĞçğ ắ ĬĞÑĦĹgÑĦĨĬĨĻçĢĞĢĨĦ κ ċ
            âĝ h ĹĦĬÌgĠĦĻĞçğã ắ ĬĞÑĦĹgÂÑĦĨĬĨĻĹắắğ
        ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ ł ĹĦĬÌgĠĦĻĞçğ ắ ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ κ ċ
            âĝ h ĹĦĬÌgĠĦĻĞçğã ắ ĬĞÑĦĹgÂĠĨĞĹĦĹắắğ
```

```
        # GĦĻ ĢIJĹĹĦĢĻĦĤ ĢĞĢĨĦ
        ŃĦĨÍĨĻçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦ w ċ
            âĝ h ĹĦĬĬgĠĦĻĞçğ ăă âĹĦĬĬgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
        ĠĨĞĹçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ w ċ
            âĝ h ĹĦĬĬgĠĦĻĞçğ ăă âĹĦĬĬgĨĻĦĹĞĻĨIJĮĹ κ ĝãã

        # VĞĮĨĬĬĞ SGD ĴĞĹĞĬĦĻĦĹ łĴĤĞĻĦ κ ĮIJĹĬĞĬĨŅĞĻĨIJĮ
        # ŃĨĻĨ ĹĶłĞĹĦ ĹIJIJĻĦĤ ĢĞĢĨĦ
        ĬĞÑĦĹgŃĦĨÍĨĻĹ κł hĹĦĬĬgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ă ċ
                        ŃĦĨÍĨĻçĬIJĬĦĮĻłĬĹçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĹĶĹĻâŃĦĨÍĨĻçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤã κ
                            ĹĦĬĬgĦĴĹĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĹĦĹ κł hĹĦĬĬgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ă ċ
                        ĠĨĞĹçĬIJĬĦĮĻłĬĹçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĹĶĹĻâĠĨĞĹçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤã κ
                            ĹĦĬĬgĦĴĹĨĬIJĮã

    # CĞĬĬ IJĮĢĦ ĞĬĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴĤĞĻĦĹ
    ĤĦĬ ĴIJĹĻçłĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬĬãf
        ĹĦĬĬgĨĻĦĹĞĻĨIJĮĹ κł ĝ
```

The following changes were made from copying the RMSProp class code:

1. renamed class from `OĴĻĨĬĨŅĦĹçRMSĴĹIJĴ` to `OĴĻĨĬĨŅĦĹçAĤĞĬ`
2. renamed the *`ĹĨIJ`* hyperparameter and property to *`ĠĦĻĞçğ`* in `ççĨĮĨĻçç`
3. added *`ĠĦĻĞçĝ`* hyperparameter and property in `ççĨĮĨĻçç`
4. added *momentum* array creation in *update_params()*
5. added *momentum* calculation
6. renamed `ĹĦĬĬgĹĨIJ` to `ĹĦĬĬgĠĦĻĞçğ` with cache calculation code in `łĴĤĞĻĦçĴĞĹĞĬĹ`
7. added `ăçĢIJĹĹĦĢĻĦĤ` variables as corrected momentums and weights
8. replaced `ĬĞÑĦĹgĤŃĦĨÍĨĻĹ`, `ĬĞÑĦĹgĤĠĨĞĹĦĹ`, `ĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦ`, and `ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ` with corrected arrays of values in parameter updates with momentum arrays

Back to our main neural network code. We can now set our optimizer to Adam, run the code, and see what impact these changes had:

```
IJĴĻĨĬĨŅĦĹ ł OĴĻĨĬĨŅĦĹçAĤĞĬâĬĦĞĹĮĨĮÍçĹĞĻĦłflgflğd ĤĦĢĞÑłĝĦhĥã
```

With our default settings, we end with:

```
ńńń
ĦĴIJĢĴf fld ĞĢĢf flgġħfld ĬIJĹĹf ĝgflîîd ĬĹf flgflğ
ĦĴIJĢĴf ĝflfld ĞĢĢf flgħíġd ĬIJĹĹf flgìiğd ĬĹf flgflĝîîíflğĝîĥíğħĝġğĝ
ĦĴIJĢĴf ğflfld ĞĢĢf flgìîġd ĬIJĹĹf flgĥħfld ĬĹf flgflĝîîħflğìîflġġìflĝflġħ
ĦĴIJĢĴf ġflfld ĞĢĢf flgíĥfld ĬIJĹĹf flgġĥíd ĬĹf flgflĝîîġflġìíğħíîìĥìħġ
ĦĴIJĢĴf ġflfld ĞĢĢf flgíìġd ĬIJĹĹf flgġìġd ĬĹf flgflĝîîğflĥĝìĝġħħğġíì
```

```
ĦĴIJĢĴf ĥflfld ĞĢĢf flgíîìd ĬIJĽĽf flgġğĝd ĬĹf flgflĝîîflflĥîĥĥğîġflíìĥ
ĦĴIJĢĴf ħflfld ĞĢĢf flgíîġd ĬIJĽĽf flgğíħd ĬĹf flgflĝîííflîĝġġğîĝĥíġġġ
ĦĴIJĢĴf ìflfld ĞĢĢf flgîflfld ĬIJĽĽf flgğħfld ĬĹf flgflĝîíħĝĝìflġĝíììğììí
ggg
ĦĴIJĢĴf ĝìflfld ĞĢĢf flgîġfld ĬIJĽĽf flgĝħġd ĬĹf flgflĝîħħĥíìħìĥġîĥflġíġ
ggg
ĦĴIJĢĴf ğħflfld ĞĢĢf flgîĥfld ĬIJĽĽf flgĝġğd ĬĹf flgflĝîġîġğħìġíĝìġíġħġ
ggg
ĦĴIJĢĴf îîflfld ĞĢĢf flgîħìd ĬIJĽĽf flgflìíd ĬĹf flgflĝíĝîíĥğììġîĝflĥîflì
ĦĴIJĢĴf ĝflflflfld ĞĢĢf flgîħġd ĬIJĽĽf flgflìîd ĬĹf flgflĝíĝíĝîíġĝìğĥììflğĥ
```

Fig 10.23: Model training with Adam optimizer.

Epilepsy Warning (quick flashing colors)

Anim 10.23: https://nnfs.io/you

This is the best result so far, but let's adjust the learning rate to be a bit higher, to *0.05* and change decay to *5e-7*:

```
IJĴĻĨĬĨŅĦĹ ł OĴĻĨĬĨŅĦĹçAĤĞİâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflĥd ĤĦĢĞÑłĥĦhìã
```

In this case, loss and accuracy slightly improved, ending on:

```
ńńń
ĦĴIJĢĴf fld ĞĢĢf flgġħfld ĬIJĹĹf ĝgflîîd ĬĹf flgflĥ
ĦĴIJĢĴf ĝflfld ĞĢĢf flgìĝġd ĬIJĹĹf flgħíġd ĬĹf flgflġîîîìĥĝĥĝĝĝĥflħġġ
ĦĴIJĢĴf ğflfld ĞĢĢf flgíğìd ĬIJĹĹf flgĥĝĝd ĬĹf flgflġîîîĥflğĥġîġîħġĝħ
ggg
ĦĴIJĢĴf ìflfld ĞĢĢf flgîflìd ĬIJĹĹf flgğħġd ĬĹf flgflġîîíğĥġĝĝflĥġìíħìĥ
ĦĴIJĢĴf íflfld ĞĢĢf flgíîìd ĬIJĹĹf flgğìíd ĬĹf flgflġîîíflflġğîìħíğĥìĥ
ĦĴIJĢĴf îflfld ĞĢĢf flgîğġd ĬIJĹĹf flgğġfld ĬĹf flgflġîîììĥġĥflîìîìġġħħ
ggg
ĦĴIJĢĴf ğflflfld ĞĢĢf flgîġfld ĬIJĹĹf flgĝìfld ĬĹf flgflġîîĥflflìġîflflĝġìġĝ
ggg
ĦĴIJĢĴf ġġflfld ĞĢĢf flgîĥfld ĬIJĹĹf flgĝġħd ĬĹf flgflġîîĝìħħflíĝíġìîîğ
ggg
ĦĴIJĢĴf ìíflfld ĞĢĢf flgîìġd ĬIJĹĹf flgflííd ĬĹf flgflġîíflĥìíğġĥĝìĝîġí
ĦĴIJĢĴf ìîflfld ĞĢĢf flgîìfld ĬIJĹĹf flgflííd ĬĹf flgflġîíflġġflĝíĥîġflħħì
ĦĴIJĢĴf íflflfld ĞĢĢf flgîífld ĬIJĹĹf flgflííd ĬĹf flgflġîíflflíğĝħĝġîĥġîî
ggg
ĦĴIJĢĴf îîflfld ĞĢĢf flgîíġd ĬIJĹĹf flgflìġd ĬĹf flgflġîìĥġìġġíġġíġîîħĥ
ĦĴIJĢĴf ĝflflflfld ĞĢĢf flgîíġd ĬIJĹĹf flgflìġd ĬĹf flgflġîìĥĝğħíĥġĝîħîġğ
```

Fig 10.24: Model training with Adam optimizer (tuned).
Epilepsy Warning (quick flashing colors)

Anim 10.24 : https://nnfs.io/car

It doesn't get much better, both for accuracy and loss. While Adam has performed the best here and is usually the best optimizer of those shown, that's not always the case. It's usually a good idea to try the Adam optimizer first but to also try the others, especially if you're not getting the results you hoped for. Sometimes simple SGD or SGD + momentum performs better than Adam. Reasons why will vary, but keep this in mind.

We will cover choosing various hyperparameters (such as the learning rate) when training, but a general starting learning rate for SGD is 1.0, with a decay down to 0.1. For Adam, a good starting LR is 0.001 (1e-3), decaying down to 0.0001 (1e-4). Different problems may require different values here, but these are decent to start.

We achieved 98.3% accuracy on the generated dataset in this section, and a loss approaching perfection (0). Rather than being excited, you will soon learn to fear results this good, or at least approach them cautiously. There are cases where you can truly achieve valid results as good as these, but, in this case, we've been ignoring a major concept in machine learning: out-of-sample testing data (which can shed light on over-fitting), which is the subject of the next section.

Full code up to this point:

```
ÕİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ÕİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ÕİĴIJĹĻ ĽĴÕĹĞĬçĤĞĻĞ

ĮĮÌĽgÕĮÕĮĻâã
```

```
ſi DĦIĻĽĦ ĬĞÑĦĹ
ĢĬĞĹĹ LĞÑĦĹçDĦIĻĽĦf

    ſi LĞÑĦĹ ĨIĮĨĻĨĞĬĨÑĢĞĻĨIJĮ
    ÂĦÌ ççĨIĮĨĻççâĹĦĬÌd IçĨIĮĴŁĻĹd IçIĮĦŁĹIJĮĹãf
        ſi IĮĨĻĨĞĬĨÑĦ ŃĦĨÍÎĻĹ ĞĮÂ ĠĨĞĹĦĹ
        ĹĦĬÌgŃĦĨÍÎĻĹ ł flgflĝ ǻ ĮĴgĹĞĮÂIJİgĹĞĮÂĮâĮçĨĮĴŁĻĹd IçIĮĦŁĹIJĮĹã
        ĹĦĬÌgĠĨĞĹĦĹ ł ĮĴgŅĦĹIJĹââĝd IçIĮĦŁĹIJĮĹãã

    ſi FIJĹŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ÌIJĹŃĞĹÂâĹĦĬÌd ĨĮĴŁĻĹãf
        ſi RĦİĦİĠĦĹ ĨIĮĴŁĻ ĹĞĬŁĦĹ
        ĹĦĬÌgĨIĮĴŁĻĹ ł ĨIĮĴŁĻĹ
        ſi CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ĹĞĬŁĦĹ ÌĹIJİ ĨIĮĴŁĻĹd ŃĦĨÍÎĻĹ ĞĮÂ ĠĨĞĹĦĹ
        ĹĦĬÌgIJŁĻĴŁĻ ł ĮĴgÂIJĻâĨIĮĴŁĻĹd ĹĦĬÌgŃĦĨÍÎĻĹã κ ĹĦĬÌgĠĨĞĹĦĹ

    ſi BĞĢĪŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ĠĞĢĪŃĞĹÂâĹĦĬÌd ÂĹĞĬŁĦĹãf
        ſi GĹĞÂĨĦIĻĹ IJI ĴĞĹĞİĦĻĦĹĹ
        ĹĦĬÌgÂŃĦĨÍÎĻĹ ł ĮĴgÂIJĻâĹĦĬÌgĨIĮĴŁĻĹgTd ÂĹĞĬŁĦĹã
        ĹĦĬÌgÂĠĨĞĹĦĹ ł ĮĴgĹŁİâÂĹĞĬŁĦĹd ĞŃĨĹŁfld ĪĦĦĴÂĨİĹŁTĹŁĦã
        ſi GĹĞÂĨĦIĻ IJI ĹĞĬŁĦĹ
        ĹĦĬÌgÂĨIĮĴŁĻĹ ł ĮĴgÂIJĻâÂĹĞĬŁĦĹd ĹĦĬÌgŃĦĨÍÎĻĹgTã


ſi RĦLU ĞĢĻĨĹĞĻĨIJĮ
ĢĬĞĹĹ AĢĻĨĹĞĻĨIJĮçRĦLUf

    ſi FIJĹŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ÌIJĹŃĞĹÂâĹĦĬÌd ĨĮĴŁĻĹãf
        ſi RĦİĦİĠĦĹ ĨIĮĴŁĻ ĹĞĬŁĦĹ
        ĹĦĬÌgĨIĮĴŁĻĹ ł ĨIĮĴŁĻĹ
        ſi CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ĹĞĬŁĦĹ ÌĹIJİ ĨIĮĴŁĻĹ
        ĹĦĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĨİŁİâfld ĨIĮĴŁĻĹã

    ſi BĞĢĪŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ĠĞĢĪŃĞĹÂâĹĦĬÌd ÂĹĞĬŁĦĹãf
        ſi SĨIĢĦ ŃĦ IĦĦÂ ĻIJ İIJÂĨÌÑ IJĹĨÍĨIĢĬ ĹĞĹĨĞĠĬĦd
        ſi ĬĦĻďĹ İĞĪĦ Ğ ĢIJĴÑ IJÌ ĹĞĬŁĦĹ ÌĨĹĹĻ
        ĹĦĬÌgÂĨIĮĴŁĻĹ ł ÂĹĞĬŁĦĹgĢIJĴÑâã

        ſi ZĦĹIJ ÍĹĞÂĨĦIĻ ŃÎĦĹĦ ĨIĮĴŁĻ ĹĞĬŁĦĹ ŃĦĹĦ IĮĦÍĞĻĨĹĦ
        ĹĦĬÌgÂĨIĮĴŁĻĹäĹĦĬÌgĨIĮĴŁĻĹ l·ł flã ł fl
```

```
# SIJÌĻİĞŇ ĞĢĻĨĹĞĻĨIJĮ
ĢĬĞĹĹ AĢĻĨĹĞĻĨIJĮçSIJÌĻİĞŇf

    # FIJĹŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ĨĮĴŁĻĹãf
        # RĦİĦİĠĦĹ ĨĮĴŁĻ ĹĞĬŁĦĹ
        ĹĦĬÌgĨĮĴŁĻĹ ł ĨĮĴŁĻĹ

        # GĦĻ ŁĮIJĹİĞĬĨŅĦĤ ĴĹIJĠĞĠĨĬĨĻĨĦĹ
        ĦŇĴçĹĞĬŁĦĹ ł ĮĴgĦŇĴâĨĮĴŁĻĹ h ĮĴgİĞŇâĨĮĴŁĻĹd ĞŇĨĹłĝd
                                                ĪĦĦĴĤĨİĹłTĹŁĦãã
        # NIJĹİĞĬĨŅĦ ĻĤĦİ ÌIJĹ ĦĞĢĤ ĹĞİĴĬĦ
        ĴĹIJĠĞĠĨĬĨĻĨĦĹ ł ĦŇĴçĹĞĬŁĦĹ w ĮĴgĹŁİâĦŇĴçĹĞĬŁĦĹd ĞŇĨĹłĝd
                                                ĪĦĦĴĤĨİĹłTĹŁĦã

        ĹĦĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĨĬĨĻĨĦĹ

    # BĞĢĨŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĹĦĬÌd ĤĹĞĬŁĦĹã f

        # CĹĦĞĻĦ ŁĮĨĮĨĻĨĞĬĨŅĦĤ ĞĹĹĞŇ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĮĴgĦİĴĻŇçĬĨĨĦâĤĹĞĬŁĦĹã

        # EĮŁİĦĹĞĻĦ IJŁĻĴŁĻĹ ĞĮĤ ÍĹĞĤĨĦĮĻĹ
        ÌIJĹ ĨĮĤĦŇd âĹĨĮÍĬĦçIJŁĻĴŁĻd ĹĨĮÍĬĦçĤĹĞĬŁĦĹã ĨĮ ċ
                ĦĮŁİĦĹĞĻĦâŅĨĴâĹĦĬÌgIJŁĻĴŁĻd ĤĹĞĬŁĦĹãã f
            # FĬĞĻĻĦĮ IJŁĻĴŁĻ ĞĹĹĞŇ
            ĹĨĮÍĬĦçIJŁĻĴŁĻ ł ĹĨĮÍĬĦçIJŁĻĴŁĻgĹĦĹĨĞĴĦâhĝd ĝã
            # CĞĬĢŁĬĞĻĦ JĞĢIJĠĨĞĮ İĞĻĹĨŇ IJÌ ĻĤĦ IJŁĻĴŁĻ ĞĮĤ
            ÏĞĢIJĠĨĞĮçİĞĻĹĨŇ ł ĮĴgĤĨĞÍÌĬĞĻâĹĨĮÍĬĦçIJŁĻĴŁĻã h ċ
                                ĮĴgĤIJĻâĹĨĮÍĬĦçIJŁĻĴŁĻd ĹĨĮÍĬĦçIJŁĻĴŁĻgTã
            # CĞĬĢŁĬĞĻĦ ĹĞİĴĬĦhŃĨĹĦ ÍĹĞĤĨĦĮĻ
            # ĞĮĤ ĞĤĤ ĨĻ ĻIJ ĻĤĦ ĞĹĹĞŇ IJÌ ĹĞİĴĬĦ ÍĹĞĤĨĦĮĻĹ
            ĹĦĬÌgĤĨĮĴŁĻĹäĨĮĤĦŇã ł ĮĴgĤIJĻâÏĞĢIJĠĨĞĮçİĞĻĹĨŇd
                                        ĹĨĮÍĬĦçĤĹĞĬŁĦĹã


# SGD IJĴĻĨİĨŅĦĹ
ĢĬĞĹĹ OĴĻĨİĨŅĦĹçSGDf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮÍĹd
    # ĬĦĞĹĮĨĮÍ ĹĞĻĦ IJÌ ĝg ĨĹ ĤĦÌĞŁĬĻ ÌIJĹ ĻĤĨĹ IJĴĻĨİĨŅĦĹ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ĤĦĢĞŇłflgd İIJİĦĮĻŁİłflgãf
        ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĢŁĹĹĦĮçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĤĦĢĞŇ ł ĤĦĢĞŇ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgİIJİĦĮĻŁİ ł İIJİĦĮĻŁİ
```

```
ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĹ
ÂĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌãƒ
    ĨÌ ĹĦĬÌgĤĦĢĞÑƒ
        ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
            âĝg w âĝg κ ĹĦĬÌgĤĦĢĞÑ ắ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

ƒ UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĹ
ÂĦÌ ŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌd ĬĞÑĦĹãƒ

    ƒ IÌ ŃĦ ŁĹĦ İIJİĦĮĻŁİ
    ĨÌ ĹĦĬÌgİIJİĦĮĻŁİƒ

        ƒ IÌ ĬĞÑĦĹ ÂIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ İIJİĦĮĻŁİ ĞĹĹĞÑĹd ĢĹĦĞĻĦ ĻĨĦİ
        ƒ ÌĨĬĬĦÂ ŃĨĻĨ ŅĦĹIJĹ
        ĨÌ ĮIJĻ ĨĞĹĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍĨĻçİIJİĦĮĻŁİĹďãƒ
            ĬĞÑĦĹgŃĦĨÍĨĻçİIJİĦĮĻŁİĹ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgŃĦĨÍĨĻĹã
            ƒ IÌ ĻĨĦĹĦ ĨĹ ĮIJ İIJİĦĮĻŁİ ĞĹĹĞÑ ÌIJĹ ŃĦĨÍĨĻĹ
            ƒ TĨĦ ĞĹĹĞÑ ÂIJĦĹĮďĻ ĦŃĨĹĻ ÌIJĹ ĠĨĞĹĦĹ ÑĦĻ ĦĨĻĨĦĹg
            ĬĞÑĦĹgĠĨĞĹçİIJİĦĮĻŁİĹ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã

        ƒ BŁĨĬĨ ŃĦĨÍĨĻ ŁĴĤĞĻĦĹ ŃĨĻĨ İIJİĦĮĻŁİ h ĻĞĨĦ ĴĹĦŁĨIJŁĹ
        ƒ ŁĴĤĞĻĦĹ İŁĬĻĨĴĬĨĦÂ ĠÑ ĹĦĻĞĨĮ ÌĞĢĻIJĹ ĞĮÂ ŁĴĤĞĻĦ ŃĨĻĨ
        ƒ ĢŁĹĹĦĮĻ ÍĹĞÂĨĦĮĻĹ
        ŃĦĨÍĨĻçŁĴĤĞĻĦĹ ł ċ
            ĹĦĬÌgİIJİĦĮĻŁİ ắ ĬĞÑĦĹgŃĦĨÍĨĻçİIJİĦĮĻŁİĹ h ċ
            ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ĬĞÑĦĹgĤŃĦĨÍĨĻĹ
        ĬĞÑĦĹgŃĦĨÍĨĻçİIJİĦĮĻŁİĹ ł ŃĦĨÍĨĻçŁĴĤĞĻĦĹ

        ƒ BŁĨĬĨ ĠĨĞĹ ŁĴĤĞĻĦĹ
        ĠĨĞĹçŁĴĤĞĻĦĹ ł ċ
            ĹĦĬÌgİIJİĦĮĻŁİ ắ ĬĞÑĦĹgĠĨĞĹçİIJİĦĮĻŁİĹ h ċ
            ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ĬĞÑĦĹgĤĠĨĞĹĦĹ
        ĬĞÑĦĹgĠĨĞĹçİIJİĦĮĻŁİĹ ł ĠĨĞĹçŁĴĤĞĻĦĹ

    ƒ VĞĮĨĬĬĞ SGD ŁĴĤĞĻĦĹ âĞĹ ĠĦÌIJĹĦ İIJİĦĮĻŁİ ŁĴĤĞĻĦã
    ĦĬĹĦƒ
        ŃĦĨÍĨĻçŁĴĤĞĻĦĹ ł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                            ĬĞÑĦĹgĤŃĦĨÍĨĻĹ
        ĠĨĞĹçŁĴĤĞĻĦĹ ł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                          ĬĞÑĦĹgĤĠĨĞĹĦĹ

    ƒ UĴĤĞĻĦ ŃĦĨÍĨĻĹ ĞĮĤ ĠĨĞĹĦĹ ŁĹĨĮÍ ĦĨĻĨĦĹ
    ƒ ŁĞĮĨĬĬĞ IJĹ İIJİĦĮĻŁİ ŁĴĤĞĻĦĹ
    ĬĞÑĦĹgŃĦĨÍĨĻĹ κł ŃĦĨÍĨĻçŁĴĤĞĻĦĹ
    ĬĞÑĦĹgĠĨĞĹĦĹ κł ĠĨĞĹçŁĴĤĞĻĦĹ
```

```
    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ÂĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ


# AĤĞÍĹĞĤ IJĴĻĨÌĬĨĢĦĹ
ĢĬĞĽĽ OĴĻĨÌĬĨĢĦĹçAĤĞÍĹĞĤf

    # IĮĨĻĨĞĬĨĢĦ IJĴĻĨÌĬĨĢĦĹ h ĽĦĻ ĽĦĻĻĨĮÍĽ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ÂĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìãf
        ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgÂĦĢĞÑ ł ÂĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ÂĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgÂĦĢĞÑf
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĽĦĬÌgÂĦĢĞÑ ắ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    # UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
    ÂĦÌ ŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ÂIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĨĦ ĞĹĹĞÑĽd
        # ĢĹĦĞĻĦ ĻĨĦİ ÌĨĬĬĦÂ ŃĨĻĨ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ĨĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍĨĻçĢĞĢĨĦďãf
            ĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgŃĦĨÍĨĻĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgĠĨĞĽĦĽã

        # UĴĤĞĻĦ ĢĞĢĨĦ ŃĨĻĨ ĽĶŁĞĹĦÂ ĢŁĹĹĦĮĻ ÍĹĞÂĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦ κł ĬĞÑĦĹgÂŃĦĨÍĨĻĽááğ
        ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ κł ĬĞÑĦĹgÂĠĨĞĽĦĽááğ

        # VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ IJĹİĞĬĨŅĞĻĨIJĮ
        # ŃĨĻĨ ĽĶŁĞĹĦ ĹIJIJĻĦÂ ĢĞĢĨĦ
        ĬĞÑĦĹgŃĦĨÍĨĻĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                        ĬĞÑĦĹgÂŃĦĨÍĨĻĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                        ĬĞÑĦĹgÂĠĨĞĽĦĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã

    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ÂĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ
```

```
# RMSJĹIJĴ IJĴĻĨĬĨŅĦĹ
ĢĬĞĽĹ OĴĻĨĬĨŅĦĹçRMSJĹIJĴf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨĬĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĺĽ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮĺçĹĞĻĦłflgflflĝd ÂĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìd
                ĹÎIJłflgîãf
        ĽĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĽĦĬÌgÂĦĢĞÑ ł ÂĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬÌgĹÎIJ ł ĹÎIJ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĠĦĻĦĹ łĴÂĞĻĦĽ
    ÂĦÌ ĴĹĦçłĴÂĞĻĦçĴĞĹĞĠĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgÂĦĢĞÑf
            ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ấ ċ
                âĝg w âĝg κ ĽĦĬÌgÂĦĢĞÑ ấ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    # UĴÂĞĻĦ ĴĞĹĞĠĦĻĦĹĽ
    ÂĦÌ łĴÂĞĻĦçĴĞĹĞĠĽâĽĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ÂIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĬĦ ĞĹĹĞÑĽd
        # ĢĹĦĞĻĦ ĻĬĦĠ ÌĬĬĬĦÂ ŃĨĻĬ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ĬĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĺĬĻçĢĞĢĬĦďãf
            ĬĞÑĦĹgŃĦĨĺĬĻçĢĞĢĬĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨĺĬĻĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢĬĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

        # UĴÂĞĻĦ ĢĞĢĬĦ ŃĨĻĬ ĽĶŁĞĹĦÂ ĢŁĹĹĦĮĻ ĺĹĞÂĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨĺĬĻçĢĞĢĬĦ ł ĽĦĬÌgĹÎIJ ấ ĬĞÑĦĹgŃĦĨĺĬĻçĢĞĢĬĦ κ ċ
            âĝ h ĽĦĬÌgĹÎIJã ấ ĬĞÑĦĹgÂŃĦĨĺĬĻĽấấğ
        ĬĞÑĦĹgĠĨĞĽçĢĞĢĬĦ ł ĽĦĬÌgĹÎIJ ấ ĬĞÑĦĹgĠĨĞĽçĢĞĢĬĦ κ ċ
            âĝ h ĽĦĬÌgĹÎIJã ấ ĬĞÑĦĹgÂĠĨĞĽĦĽấấğ

        # VĞĮĨĬĬĞ SGD ĴĞĹĞĠĦĻĦĹ łĴÂĞĻĦ κ IJĹĠĬĞĬĨŅĞĻĨIJĮ
        # ŃĨĻĬ ĽĶŁĞĹĦ ĹIJIJĻĦÂ ĢĞĢĬĦ
        ĬĞÑĦĹgŃĦĨĺĬĻĽ κł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ấ ċ
                        ĬĞÑĦĹgÂŃĦĨĺĬĻĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨĺĬĻçĢĞĢĬĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ấ ċ
                       ĬĞÑĦĹgÂĠĨĞĽĦĽ w ċ
                       âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢĬĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã

    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĠĦĻĦĹ łĴÂĞĻĦĽ
    ÂĦÌ ĴIJĽĻçłĴÂĞĻĦçĴĞĹĞĠĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ
```

```
ƒ AĤĞİ IJĴĻĨİĨĨŅĦĹ
ĢĬĞĽĹ OĴĻĨİĨĨŅĦĹçAĤĞİf

    ƒ IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĬĹ
    ÎĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮĬçĹĞĻĦłflgflflĝd ÎĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìd
                ĠĦĻĞçĝłflgîd ĠĦĻĞçğłflgîîîãf
        ĽĦĬÌgĬĦĞĹĮĨĮĬçĹĞĻĦ ł ĬĦĞĹĮĨĮĬçĹĞĻĦ
        ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ł ĬĦĞĹĮĨĮĬçĹĞĻĦ
        ĽĦĬÌgÎĦĢĞÑ ł ÎĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬÌgĠĦĻĞçĝ ł ĠĦĻĞçĝ
        ĽĦĬÌgĠĦĻĞçğ ł ĠĦĻĞçğ

    ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ łĴÎĞĻĦĽ
    ÎĦÌ ĴĹĦçłĴÎĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgÎĦĢĞÑf
            ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĬçĹĞĻĦ ấ ċ
                âĝg w âĝg κ ĽĦĬÌgÎĦĢĞÑ ấ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    ƒ UĴÎĞĻĦ ĴĞĹĞİĦĻĦĹĽ
    ÎĦÌ łĴÎĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

        ƒ IÌ ĬĞÑĦĹ ÎIJĦĽ ĮIJĮ ĢIJĮĻĞĨĮ ĢĞĢÎĦ ĞĹĹĞÑĽd
        ƒ ĢĹĦĞĻĦ ĻÎĦİ ÌĨĬĬĦÎ ŃĨĻÎ ŅĦĹIJĽ
        ĨÌ ĮIJĮ ÎĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍÎĻçĢĞĢÎĦďãf
            ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻłİĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨÍÎĻĽã
            ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨÍÎĻĽã
            ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

        ƒ UĴÎĞĻĦ İIJİĦĮĻłİ  ŃĨĻÎ ĢłĹĹĦĮĻ ÍĹĞÎĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻłİĽ ł ĽĦĬÌgĠĦĻĞçĝ ấ ċ
                                  ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻłİĽ κ ċ
                                  âĝ h ĽĦĬÌgĠĦĻĞçĝã ấ ĬĞÑĦĹgÎŃĦĨÍÎĻĽ
        ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ ł ĽĦĬÌgĠĦĻĞçĝ ấ ċ
                                ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ κ ċ
                                âĝ h ĽĦĬÌgĠĦĻĞçĝã ấ ĬĞÑĦĹgÎĠĨĞĽĦĽ
        ƒ GĦĻ ĢIJĹĹĦĢĻĦÎ İIJİĦĮĻłİ
        ƒ ĽĦĬÌgĨĻĦĹĞĻĨIJĮ ĨĽ fl ĞĻ ÌĨĹĹĽĻ ĴĞĽĽ
        ƒ ĞĮÎ ŃĦ ĮĦĦÎ ĻIJ ĽĻĞĹĹĻ ŃĨĻÎ ĝ ÎĦĹĦ
        ŃĦĨÍÎĻçİIJİĦĮĻłİĽçĢIJĹĹĦĢĻĦÎ ł ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻłİĽ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçĝ ấấ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã
        ĠĨĞĽçİIJİĦĮĻłİĽçĢIJĹĹĦĢĻĦÎ ł ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçĝ ấấ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã
        ƒ UĴÎĞĻĦ ĢĞĢÎĦ ŃĨĻÎ ĽĶłĞĹĦÎ ĢłĹĹĦĮĻ ÍĹĞÎĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĽĦĬÌgĠĦĻĞçğ ấ ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ κ ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğã ấ ĬĞÑĦĹgÎŃĦĨÍÎĻĽấấğ
```

```
        ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĽĦĬÌgĠĦĻĞçğ ő ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ κ ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğã ő ĬĞÑĦĹgĤĠĨĞĽĦĽőőğ
        ńĦĻ ĢIJĹĹĦĢĻĦĤ ĢĞĢÎĦ
        ŃĦĨÍÎĮçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨÍÎĮçĢĞĢÎĦ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğ őő âĽĦĬÌgĨĮĦĹĞĮĨIJĮĽ κ ĝãã
        ĠĨĞĽçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğ őő âĽĦĬÌgĨĮĦĹĞĮĨIJĮĽ κ ĝãã

        ńĦ VĞĮĨĬĬĞ SGD ĴĞĹĞĠĦĻĦĹ łĴĤĞĮĦ κ ĮIJĹĠĞĬĨŅĞĮĨIJĮ
        ńĦ ŃĨĮÎ ĽĶłĞĹĦ ĹIJIJĻĦĤ ĢĞĢÎĦ
        ĬĞÑĦĹgŃĦĨÍÎĮĽ κł hĽĦĬÌgĢłĹĹĦĮĮçĬĦĞĹĮĨĮĬçĹĞĮĦ ő ċ
                        ŃĦĨÍÎĮçĠIJĠĦĮĮłĠĽçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĽĶĹĮâŃĦĨÍÎĮçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤã κ
                            ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢłĹĹĦĮĮçĬĦĞĹĮĨĮĬçĹĞĮĦ ő ċ
                        ĠĨĞĽçĠIJĠĦĮĮłĠĽçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĽĶĹĮâĠĨĞĽçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤã κ
                            ĽĦĬÌgĦĴĽĨĬIJĮã

    ńĦ CĞĬĬ IJĮĢĦ ĞÌĮĦĹ ĞĮÑ ĴĞĹĞĠĦĻĦĹ łĴĤĞĮĦĽ
    ĤĦÌ ĴIJĽĮçłĴĤĞĮĦçĴĞĹĞĠĽâĽĦĬÌãf
        ĽĦĬÌgĨĮĦĹĞĮĨIJĮĽ κł ĝ


ńĦ CIJĠĠIJĮ ĬIJĽĽ ĢĬĞĽĽ
ĢĬĞĽĽ LIJĽĽf

    ńĦ CĞĬĢłĬĞĮĦĽ ĮĨĦ ĤĞĮĞ ĞĮĤ ĹĦÍłĬĞĹĨŅĞĮĨIJĮ ĬIJĽĽĦĽ
    ńĦ ÍĨĮĦĮ ĠIJĤĦĬ IJłĮĴłĮ ĞĮĤ ÍĹIJłĮĤ ĮĹłĮÎ ĮĞĬłĦĽ
    ĤĦÌ ĢĞĬĢłĬĞĮĦâĽĦĬÌd IJłĮĴłĮd Ñãf

        ńĦ CĞĬĢłĬĞĮĦ ĽĞĠĴĬĦ ĬIJĽĽĦĽ
        ĽĞĠĴĬĦçĬIJĽĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹĤâIJłĮĴłĮd Ñã

        ńĦ CĞĬĢłĬĞĮĦ ĠĦĞĮ ĬIJĽĽ
        ĤĞĮĞçĬIJĽĽ ł ĮĴgĠĦĞĮâĽĞĠĴĬĦçĬIJĽĽĦĽã

        ńĦ RĦĮłĹĮ ĬIJĽĽ
        ĹĦĮłĹĮ ĤĞĮĞçĬIJĽĽ


ńĦ CĹIJĽĽhĦĮĮĹIJĴÑ ĬIJĽĽ
ĢĬĞĽĽ LIJĽĽçCĞĮĦĹIJĹĨĢĞĬCĹIJĽĽĦĮĮĹIJĴÑâĽIJĽĽãf

    ńĦ FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ÑçĴĹĦĤd ÑçĮĹłĦãf

        ńĦ NłĠĦĹ IJÌ ĽĞĠĴĬĦĽ ĨĮ Ğ ĠĞĮĢĤ
        ĽĞĠĴĬĦĽ ł ĬĦĮâÑçĴĹĦĤã
```

```
    ƒ CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ĤĨŁĨĹĨIJĮ ĠÑ ƒl
    ƒ CĬĨĴ ĠIJĻĨ ĹĨĤĦĹ ĻIJ ĮIJĻ ĤĹĞÍ ĬĦĞĮ ĻIJŃĞĹĤĹ ĞĮÑ ŁĞĬŁĦ
    ÑçĴĹĦĤçĢĬĨĴĴĦĤ ł ĮĴgĢĬĨĴâÑçĴĹĦĤd ĝĦhìd ĝ h ĝĦhìã

    ƒ PĹIJĠĞĠĨĬĨĻĨĦĹ ÌIJĹ ĻĞĹÍĦĻ ŁĞĬŁĦĹ h
    ƒ IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĹ
    ĨÌ ĬĦĮâÑçĻĹŁĦgĹĨĞĴĦã łł ĝf
        ĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĹ ł ÑçĴĹĦĤçĢĬĨĴĴĦĤä
            ĹĞĮÍĦâĹĞĬĴĬĦĹãd
            ÑçĻĹŁĦ
        ã

    ƒ MĞĹĨ ŁĞĬŁĦĹ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJĤĦĤ ĬĞĠĦĬĹ
    ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĹĨĞĴĦã łł ğf
        ĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĹ ł ĮĴgĹŁİâ
            ÑçĴĹĦĤçĢĬĨĴĴĦĤ ǻ ÑçĻĹŁĦd
            ĞÑĨĹłĝ
        ã

    ƒ LIJĹĹĦĹ
    ĮĦÍĞĻĨŁĦçĬIJÍçĬĨĨĦĬĨĨIJIJĤĹ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĹã
    ĹĦĻŁĹĮ ĮĦÍĞĻĨŁĦçĬIJÍçĬĨĨĦĬĨĨIJIJĤĹ

ƒ BĞĢĪŃĞĹĤ ĴĞĹĹ
ĤĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ĤŁĞĬŁĦĹd ÑçĻĹŁĦã f

    ƒ NŁİĠĦĹ IJÌ ĹĞİĴĬĦĹ
    ĹĞİĴĬĦĹ ł ĬĦĮâĤŁĞĬŁĦĹã
    ƒ NŁİĠĦĹ IJÌ ĬĞĠĦĬĹ ĨĮ ĦŁĦĹÑ ĹĞİĴĬĦ
    ƒ WĦďĬĬ ŁĹĦ ĻĤĦ ÌĨĹĹĻ ĹĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦİ
    ĬĞĠĦĬĹ ł ĬĦĮâĤŁĞĬŁĦĹäflãã

    ƒ IÌ ĬĞĠĦĬĹ ĞĹĦ ĹĴĞĹĹĦd ĻŁĹĮ ĻĤĦİ ĨĮĻIJ IJĮĦhÎIJĻ ŁĦĢĻIJĹ
    ĨÌ ĬĦĮâÑçĻĹŁĦgĹĨĞĴĦã łł ĝf
        ÑçĻĹŁĦ ł ĮĴgĦÑĦâĬĞĠĦĬĹääÑçĻĹŁĦã

    ƒ CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
    ĹĦĬÌgĤĨĮĴŁĻĹ ł hÑçĻĹŁĦ w ĤŁĞĬŁĦĹ
    ƒ NIJĹĬĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
    ĹĦĬÌgĤĨĮĴŁĻĹ ł ĹĦĬÌgĤĨĮĴŁĻĹ w ĹĞİĴĬĦĹ
```

```
# SIJÌĻİĞŇ ĢĬĞĽĽĨÌÌĨĦĹ h ĢIJİĠĨĮĦĤ SIJÌĻİĞŇ ĞĢĻĨŁĠĻĨIJĮ
# ĞĮĤ ĢĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ ÌIJĹ ÌĞĽĻĦĹ ĠĞĢĪŃĞĹĤ ĹĻĦĴ
ĢĬĞĽĽ AĢĻĨŁĠĻĨIJĮçSIJÌĻİĞŇçLIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâãf

    # CĹĦĞĻĦĹ ĞĢĻĨŁĠĻĨIJĮ ĞĮĤ ĬIJĹĽ ÌŁĮĢĻĨIJĮ IJĠÏĦĢĻĽ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ĽĦĬÌgĞĢĻĨŁĠĻĨIJĮ ł AĢĻĨŁĠĻĨIJĮçSIJÌĻİĞŇâã
        ĽĦĬÌgĬIJĹĽ ł LIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâã

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ÑçĻĹŁĦãf
        # OŁĻĴŁĻ ĬĞÑĦĹďĽ ĞĢĻĨŁĠĻĨIJĮ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĞĢĻĨŁĠĻĨIJĮgÌIJĹŃĞĹĤâĨĮĴŁĻĽã
        # SĦĻ ĻĤĦ IJŁĻĴŁĻ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĽĦĬÌgĞĢĻĨŁĠĻĨIJĮgIJŁĻĴŁĻ
        # CĞĬĢŁĬĞĻĦ ĞĮĤ ĹĦĻŁĹĮ ĬIJĹĽ ŁĠĬŁĦ
        ĹĦĻŁĹĮ ĽĦĬÌgĬIJĹĽgĢĞĬĢŁĬĞĻĦâĽĦĬÌgIJŁĻĴŁĻd ÑçĻĹŁĦã

    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤŁĠĬŁĦĽd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâĤŁĠĬŁĦĽã

        # IÌ ĬĞĠĦĬĽ ĞĹĦ IJĮĦhÎIJĻ ĦĮĢIJĤĦĤd
        # ĻŁĹĮ ĻĤĦİ ĨĮĻIJ ĤĨĽĢĹĦĻĦ ŁĠĬŁĦĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ÑçĻĹŁĦ ł ĮĴgĞĹÍİĞŇâÑçĻĹŁĦd ĞŇĨĽłĝã

        # CIJĴÑ ĽIJ ŃĦ ĢĞĮ ĽĞÌĦĬÑ İIJĤĨÌÑ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĤŁĠĬŁĦĽgĢIJĴÑâã
        # CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽäĹĞĮÍĦâĽĞİĴĬĦĽãd ÑçĻĹŁĦã hł ĝ
        # NIJĹİĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽĞİĴĬĦĽ


# CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

# CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮĤ ħǵ IJŁĻĴŁĻ ŁĠĬŁĦĽ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħǵã

# CĹĦĞĻĦ RĦLU ĞĢĻĨŁĠĻĨIJĮ âĻIJ ĠĦ ŁĽĦĤ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨŁĠĻĨIJĮĝ ł AĢĻĨŁĠĻĨIJĮçRĦLUâã

# CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJŁĻĴŁĻ
# IJÌ ĴĹĦŁĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ġ IJŁĻĴŁĻ ŁĠĬŁĦĽ âIJŁĻĴŁĻ ŁĠĬŁĦĽã
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâħǵd ġã
```

```
# CĹĦĞĻĦ SIJÌĻİĞŇ ĢĬĞĽĹĨÌĨĦĹďĽ ĢIJİĠĨĮĦĤ ĬIJĹĽ ĞĮĤ ĞĢĻĨĿĞĻĨIJĮ
ĬIJĹĽçĞĢĻĨĿĞĻĨIJĮ ł AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇçLIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâã

# CĹĦĞĻĦ IJĴĻĨİĨĨŅĦĹ
IJĴĻĨİĨĨŅĦĹ ł OĴĻĨİĨŅĦĹçAĤĞİâĬĦĞĹĮĨĮÍçĹĞĻĦłflgflĥd ĤĦĢĞŇłĥĦhìã

# TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢĤ ĨĮ ĹĞĮÍĦâĝflflflĝãf

    # PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ĤĞĻĞ ĻĤĹIJŁÍĤ ĻĤĨĽ ĬĞÑĦĹ
    ĤĦĮĽĦĝgÌIJĹŃĞĹĤâXã

    # PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĤĹIJŁÍĤ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    # ĻĞĨĦĹ ĻĤĦ IJŁĻĴŁĻ IJÌ ÌĨĹĹĽĻ ĤĦĮĽĦ ĬĞÑĦĹ ĤĦĹĦ
    ĞĢĻĨĿĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĽĦĝgIJŁĻĴŁĻã

    # PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĤĹIJŁÍĤ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ
    # ĻĞĨĦĹ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
    ĤĦĮĽĦğgÌIJĹŃĞĹĤâĞĢĻĨĿĞĻĨIJĮĝgIJŁĻĴŁĻã

    # PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻĤĹIJŁÍĤ ĻĤĦ ĞĢĻĨĿĞĻĨIJĮwĬIJĹĽ ÌŁĮĢĻĨIJĮ
    # ĻĞĨĦĹ ĻĤĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮĤ ĤĦĮĽĦ ĬĞÑĦĹ ĤĦĹĦ ĞĮĤ ĹĦĻŁĹĮĽ ĬIJĹĽ
    ĬIJĹĽ ł ĬIJĹĽçĞĢĻĨĿĞĻĨIJĮgÌIJĹŃĞĹĤâĤĦĮĽĦğgIJŁĻĴŁĻd Ñã

    # CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĿĞĻĨIJĮğ ĞĮĤ ĻĞĹĹĦĻĽ
    # ĢĞĬĢŁĬĞĻĦ ĿĞĬŁĦĽ ĞĬIJĮĹ ÌĨĹĹĽĻ ĞŇĨĽ
    ĴĹĦĤĬĢĻĨIJĮĽ ł ĮĴgĞĹĹİĞŇâĬIJĹĽçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻd ĞŇĨĽłĝã
    ĨÌ ĬĦĮâÑgĽĤĞĴĦã łł ğf
        Ñ ł ĮĴgĞĹĹİĞŇâÑd ĞŇĨĽłĝã
    ĞĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦĤĬĢĻĨIJĮĽłłÑã

    ĨÌ ĮIJĻ ĦĴIJĢĤ ķ ĝflflf
        ĴĹĨĮĻâÌďĦĴIJĢĤf ăĦĴIJĢĤåd ď κ
              ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĹĽf ăĬIJĹĽfgġÌåd ď κ
              ÌďĬĹf ăIJĴĻĨİĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

    # BĞĢĬŃĞĹĤ ĴĞĽĽ
    ĬIJĹĽçĞĢĻĨĿĞĻĨIJĮgĠĞĢĬŃĞĹĤâĬIJĹĽçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻd Ñã
    ĤĦĮĽĦğgĠĞĢĬŃĞĹĤâĬIJĹĽçĞĢĻĨĿĞĻĨIJĮgĤĨĮĴŁĻĽã
    ĞĢĻĨĿĞĻĨIJĮĝgĠĞĢĬŃĞĹĤâĤĦĮĽĦğgĤĨĮĴŁĻĽã
    ĤĦĮĽĦĝgĠĞĢĬŃĞĹĤâĞĢĻĨĿĞĻĨIJĮĝgĤĨĮĴŁĻĽã

    # UĴĤĞĻĦ ŃĦĬÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
    IJĴĻĨİĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâã
    IJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĤĦĮĽĦĝã
    IJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĤĦĮĽĦğã
    IJĴĻĨİĨŅĦĹgĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâã
```

Supplementary Material: https://nnfs.io/ch10 Chapter code, further resources, and errata for this chapter.

Chapter 11

Testing with Out-of-Sample Data

Up to this point, we've created a model that is seemingly 98% accurate at predicting the testing dataset that we've generated. These generated data are created based on a very clear set of rules outlined in the *spiral_data* function. The expectation is that a well-trained neural network can learn a representation of these rules and use this representation to predict classes of additional generated data.

Imagine that you've trained a neural network model to read license plates on vehicles. The expectation for a well-trained model, in this case, would be that it could see future examples of license plates and still accurately predict them (a prediction, in this case, would be correctly identifying the characters on the license plate).

The complexity of neural networks is their biggest issue and strength. By having a massive amount of tunable parameters, they are exceptional at "fitting" to data. This is a gift, a curse,

and something that we must constantly try to balance. With enough neurons, a model can easily memorize a dataset; however, it can not generalize the data with too few. This is one reason why we do not simply solve problems with neural networks by using the most neurons or biggest models possible.

At the moment, we're uncertain whether our latest neural network's 98% accuracy is due to learning to meaningfully represent the underlying data-generation function or instead **overfitting** the data. So far, we have only tuned hyper-parameters to achieve the highest possible accuracy on the training data, and have never tried to challenge the model with the previously unseen data.

Overfitting is effectively just memorizing the data without any understanding of it. An overfit model will do very well predicting the data that it has already seen, but often significantly worse on unseen data.

Fig 11.01: Good generalization (left) and overfitting (right) on the data

The left image shows an example of generalization. In this example, the model learned to separate red and blue data points, even if some of them will be predicted incorrectly. One reason for this might be the data that contains some "confusing" samples. When you look at the image, you can see that, for example, some of these blue dots might not be there, which would raise the data quality and make it easier to fit. A good dataset is one of the biggest challenges with neural networks. The image on the right shows the model that memorized the data, fitting them perfectly and ruining generalization.

Without knowing if a model overfits the training data, we cannot trust the model's results. For this reason, it's essential to have both **training** and **testing data** as separate sets for different purposes.

Training data should only be used to train a model. The **testing,** or **out-of-sample** data, should only be used to validate a model's performance after training (we are using the testing data during training later in this chapter for demonstration purposes only). The idea is that some data are reserved and withheld from the training data for testing the model's performance.

In many cases, one can take a random sampling of available data to train with and make the remaining data the testing dataset. You still need to be very careful about information leaking through. One common area where this can be problematic is in time-series data. Consider a scenario where you have data from sensors collected every second. You might have millions of observations collected, and randomly selecting your data for the **testing** data might result in samples in your **testing** dataset that are only a second in time apart from your **training** data, thus are very similar. This means overfitting can spill into your testing data, and the model can achieve good results on both the training and the testing data, which won't mean it generalized well. Randomly allocating time-series data as testing data may be very similar to training data. Both datasets must differ enough to prove the model's ability to generalize. In time-series data, a better approach is to take multiple slices of your data, entire blocks of time, and reserve those for testing.

Other biases like these can sneak into your testing dataset, and this is something you must be vigilant about, carefully considering if data leakage has occurred and how to truly isolate **out-of-sample** data.

In our case, we can use our data-generating function to create new data that will serve as out-of-sample/testing data:

```
ﬁ CĹĦĞĻĦ ĻĦĹĻ ĤĞĻĞĹĦĻ
XçĻĦĹĻd ÑçĻĦĹĻ ł ĹĴĨĹĞĬçĤĞĻĞâĹĞĬĴĬĦĹłĝﬂﬂd ĢĬĞĹĹĦĹłġã
```

Given what was just said about overfitting, it may look wrong to only generate more data, as the testing data could look similar to the training data. Intuition and experience are both important to spot potential issues with out-of-sample data. By looking at the image representation of the data, we can see that another set of data generated by the same function will be adequate. This is just about as safe as it gets for out-of-sample data as the classes are partially mixing at the edges (also, we're quite literally using the "underlying function" to make more data).

With these data, we evaluate the model's performance by doing a forward pass and calculating loss and accuracy the same as before:

```
# VĞĬĨĤĞĻĦ ĻÎĦ İIJĤĦĬ

# CĹĦĞĻĦ ĻĦĹĻ ĤĞĻĞĹĦĻ
XçĻĦĹĻd ÑçĻĦĹĻ ł ĹĴĨĹĞĬçĤĞĻĞâĹĞİĴĬĦĹłĝﬁﬁd ĢĬĞĹĹĦĹłġã

# PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ IJÌ IJŁĹ ĻĦĹĻĨĮÍ ĤĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĹ ĬĞÑĦĹ
ĤĦĮĹĦĝgÌIJĹŃĞĹĤâXçĻĦĹĻã

# PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ ĻÎĹIJŁÍÎ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
# ĻĞĪĦĹ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĹĻ ĤĦĮĹĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨĿĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĹĦĝgIJŁĻĴŁĻã

# PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ ĻÎĹIJŁÍÎ ĹĦĢIJĮĤ DĦĮĹĦ ĬĞÑĦĹ
# ĻĞĪĦĹ IJŁĻĴŁĻĹ IJÌ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĹĻ ĬĞÑĦĹ ĞĹ ĨĮĴŁĻĹ
ĤĦĮĹĦğgÌIJĹŃĞĹĤâĞĢĻĨĿĞĻĨIJĮĝgIJŁĻĴŁĻã

# PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ ĻÎĹIJŁÍÎ ĻÎĦ ĞĢĻĨĿĞĻĨIJĮwĬIJĹĹ ÌŁĮĢĻĨIJĮ
# ĻĞĪĦĹ ĻÎĦ IJŁĻĴŁĻ IJÌ ĹĦĢIJĮĤ ĤĦĮĹĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮĤ ĹĦĻŁĹĮĹ ĬIJĹĹ
ĬIJĹĹ ł ĬIJĹĹçĞĢĻĨĿĞĻĨIJĮgÌIJĹŃĞĹĤâĤĦĮĹĦğgIJŁĻĴŁĻd ÑçĻĦĹĻã

# CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĿĞĻĨIJĮğ ĞĮĤ ĻĞĹĹĦĻĹ
# ĢĞĬĢŁĬĞĻĦ ĿĞĬŁĦĹ ĞĬIJĮÍ ÌĨĹĹĻ ĞÑĨĹ
ĴĹĦĤĨĢĻĨIJĮĹ ł ĮĴgĞĹĹİĞÑâĬIJĹĹçĞĢĻĨĿĞĻĨIJĮgIJŁĻĴŁĻd ĞÑĨĹłĝã
ĨÌ ĬĦĮâÑçĻĦĹĻgĹĴĞĴĦã łł ğf
    ÑçĻĦĹĻ ł ĮĴgĞĹĹİĞÑâÑçĻĦĹĻd ĞÑĨĹłĝã
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦĤĨĢĻĨIJĮĹłłÑçĻĦĹĻã

ĴĹĨĮĻâÌďĿĞĬĨĤĞĻĨIJĮd ĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ĬIJĹĹf ăĬIJĹĹfgġÌåďã


ńńń
ggg
ĦĴIJĢĴf îíﬁﬁd ĞĢĢf ﬁgîíġd ĬIJĹĹf ﬁgﬁìĥd ĬĹf ﬁgﬁġîìĥĥĝĝîġﬁġﬁġġíġ
ĦĴIJĢĴf îîﬁﬁd ĞĢĢf ﬁgîíġd ĬIJĹĹf ﬁgﬁìġd ĬĹf ﬁgﬁġîìĥġìġġíġġíġîîĥĥ
ĦĴIJĢĴf ĝﬁﬁﬁﬁd ĞĢĢf ﬁgîíġd ĬIJĹĹf ﬁgﬁìġd ĬĹf ﬁgﬁġîìĥĝĝĥíĥġĝîĥîġĝ
ĿĞĬĨĤĞĻĨIJĮd ĞĢĢf ﬁgíﬁġd ĬIJĹĹf ﬁgíĥí
```

While 80.3% accuracy and a loss of *0.858* is not terrible, this contrasts with our training data that achieved 98% accuracy and a loss of 0.074. This is evidence of over-fitting. In the following image, the training data is dimmed, and validation data points are shown on top of it at the same positions for both the well-generalized (on the left) and overfitted (on the right) models.

Fig 11.02: Left - prediction with well-generalized model; right - prediction mistakes with an overfit model.

We can recognize overfitting when testing data results begin to diverge in trend from training data. It will usually be the case that performance against your training data is better, but having training loss differ from test performance by over 10% approximately is a common sign of serious overfitting from our anecdotal experience. Ideally, both datasets would have identical performance. Even a small difference means that the model did not correctly predict some testing samples, implying slight overfitting of training data. In most cases, modest overfitting is not a serious problem, but something we hope to minimize.

Let's see the training process of this model once again, but with the training data, training accuracy, and loss plots dimmed. We add the test data and its loss and accuracy plotted on top of the training counterparts to show this model overfitting:

Fig 11.03: Prediction issues on the testing data — overfitted model.

Anim 11.03: https://nnfs.io/zog

This is a classic example of overfitting — the validation loss falls down, then starts rising once the model starts overfitting. The dots representing classes in the validation data can be spotted over areas of effect of other classes. Previously, we weren't aware that this was happening; we were just seeing very good training results. That's why we usually should use the testing data to test the model after training. The model is currently tuned to achieve the best possible score on the training data, and most likely the learning rate is too high, there are too many training epochs, or the model is too big. There are other possible causes and ways to fix this, but this is the topic of the following chapters. In general, the goal is to have the testing loss identical to the training loss, even if that means higher loss and lower accuracy on the training data. Similar performance on both datasets means that model generalized instead of overfitting on the training data.

As mentioned, one option to prevent overfitting is to change the model's size. If a model is not learning at all, one solution might be to try a larger model. If your model is learning, but there's a divergence between the training and testing data, it could mean that you should try a smaller model. One general rule to follow when selecting initial model hyperparameters is to find the smallest model possible that still learns. Other possible ways to avoid overfitting are regularization techniques we'll discuss in chapter 14, and the *Dropout* layer explained in chapter 15. Often the divergence of the training and testing data can take a long time to occur. The process of trying different model settings is called hyperparameter searching. Initially, you can very quickly (usually within minutes) try different settings (e.g., layer sizes) to see if the models are learning *something*. If they are, train the models fully — or at least significantly longer — and compare results to pick the best set of hyperparameters. Another possibility is to create a list of different hyperparameter sets and train the model in a loop using each of those sets at a time to pick the best set at the end. The reasoning here is that the fewer neurons you have, the less chance you have that the model is memorizing the data. Fewer neurons can mean it's easier for a neural network to generalize (actually learn the meaning of the data) compared to memorizing the data. With enough neurons, it's easier for a neural network to memorize the data. Remember that the neural network wants to decrease training loss and follows the path of least resistance to meet that objective. Our job as the programmer is to make the path to generalization the easiest path. This can often mean our job is actually to make the path to lowering loss for the model more challenging!

Supplementary Material: https://nnfs.io/ch11
Chapter code, further resources, and errata for this chapter.

Chapter 12

Validation Data

In the chapter on optimization, we used hyperparameter tuning to select hyperparameters that lead to better results, but one more thing requires clarification. We *should not* check different hyperparameters using the test dataset; if we do that, we're going to be manually optimizing the model to the test dataset, biasing it towards overfitting these data, and these data are supposed to be used only to perform the last check if the model trains and generalizes well. In other words, if we're tuning our network's parameters to fit the testing data, then we're essentially optimizing our network on the testing data, which is another way for overfitting on these data.

Thus, hyperparameter tuning using the test dataset is a mistake. The test dataset should only be used as unseen data, not informing the model in any way, which hyperparameter tuning is, other than to test performance.

Hyperparameter tuning can be performed using yet another dataset called **validation data**. The test dataset needs to contain real out-of-sample data, but with a validation dataset, we have more freedom with choosing data. If we have a lot of training data and can afford to use some for validation purposes, we can take it as an out-of-sample dataset, similar to a test dataset. We can now search for parameters that work best using this new validation dataset and test our model

at the end using the test dataset to see if we really tuned the model or just overfitted it to the validation data.

There are situations when we'll be short on data and cannot afford to create yet another dataset from the training data. In those situations, we have two options:

The first is to temporarily split the training data into a smaller training dataset and validation dataset for hyperparameter tuning. Afterward, with the final hyperparameter set, train the model on all the training data. We allow ourselves to do that as we tune the model to the part of training data that we put aside as validation data. Keep in mind that we still have a test dataset to check the model's performance after training.

The second possibility in situations where we are short on data is a process called **cross-validation**. Cross-validation is primarily used when we have a small training dataset and cannot afford any data for validation purposes. How it works is we split the training dataset into a given number of parts, let's say 5. We now train the model on the first 4 chunks and validate it on the last. So far, this is similar to the case described previously — we are also only using the training dataset and can validate on data that was not used for training. What makes cross-validation different is that we then swap samples. For example, if we have 5 chunks, we can call them chunks A, B, C, D, and E. We may first train on A, B, C, and D, then validate on E. We'll then train on A, B, C, E, and validate on D, doing this until we've validated on each of the 5 sample groups. This way, we do not lose any training data. We validate using the data that was not used for training during any given iteration and validate on more data than if we just temporarily split the training dataset and train on all of the samples. This validation method is often called k-fold cross-validation; here, our k is 5. Here's an example of 2 steps of cross-validation:

Fig 12.01: Cross-validation, first step.

Fig 12.02: Cross-validation, third step.

Anim 12.01-12.02: https://nnfs.io/lho

When using a validation dataset and cross-validation, it is common to loop over different hyperparameter sets, leaving the code to run training multiple times, applying different settings each run, and reviewing the results to choose the best set of hyperparameters. In general, we should not loop over *all* possible setting combinations that we would like to check unless training is exceptionally fast. It's usually better to check some settings that we suspect will work well, pick the best combination of those settings, tweak them to create the next list of setting sets, and train the model on new sets. We can repeat this process as many times as we'd like.

Supplementary Material: https://nnfs.io/ch12 Chapter code, further resources, and errata for this chapter.

Chapter 13

Training Dataset

Since we are talking about datasets and testing, it's worth mentioning a few things about the training dataset and operations that we can perform on it; this technique is referred to as **preprocessing**. However, it's important to remember that any preprocessing we do to our training data also needs to be done to our validation and testing data and later done to the prediction data.

Neural networks usually perform best on data consisting of numbers in a range of 0 to 1 or -1 to 1, with the latter being preferable. Centering data on the value of 0 can help with model training as it attenuates weight biasing in some direction. Models can work fine with data in the range of 0 to 1 in most cases, but sometimes we're going to need to rescale them to a range of -1 to 1 to get training to behave or achieve better results.

Speaking of the data range, the values do not have to strictly be in the range of -1 and 1 — the model will perform well with data slightly outside of this range or with just some values being many times bigger. The case here is that when we multiply data by a weight and sum the results with a bias, we're usually passing the resulting output to an activation function. Many activation functions behave properly within this described range. For example, *softmax* outputs a vector of probabilities containing numbers in the range of 0 to 1; *sigmoid* also has an output range of 0 to 1,

but *tanh* outputs a range from -1 to 1.

Another reason why this scaling is ideal is a neural network's reliance on many multiplication operations. If we multiply by numbers above 1 or below -1, the resulting value is larger in scale than the original one. Within the -1 to 1 range, the result becomes a fraction, a smaller value. Multiplying big numbers from our training data with weights might cause floating-point overflow or instability — weights growing too fast. It's easier to control the training process with smaller numbers.

There are many terms related to data **preprocessing**: standardization, scaling, variance scaling, mean removal (as mentioned above), non-linear transformations, scaling to outliers, etc., but they are out of the scope of this book. We're only going to scale data to a range by simply dividing all of the numbers by the maximum of their absolute values. For the example of an image that consists of numbers in the range between *0* and *255*, we divide the whole dataset by *255* and return data in the range from *0* to *1*. We can also subtract *127.5* (to get a range from *-127.5* to *127.5*) and divide by 127.5, returning data in the range from -1 to 1.

We need to ensure identical scaling for all the datasets (same scale parameters). For example, we can find the maximum for training data and divide training, validation and testing data by this number. In general, we should prepare a scaler of our choice and use its instance on every dataset. It is important to remember that once we train our model and want to predict using new samples, we need to scale those new samples by using the same scaler instance we used on the training, validation, and testing data. In most cases, when we are working with data (e.g., sensor data), we will need to save the scaler object along with the model and use it during prediction as well; otherwise, results are likely to vary as the model might not effectively recognize these data without being scaled. It is usually fine to scale datasets that consist of larger numbers than the training data using a scaler prepared on the training data. If the resulting numbers are slightly outside of the *-1* to *1* range, it does not affect validation or testing negatively, since we do not train on these data. Additionally, for linear scaling, we can use different datasets to find the maximum as well, but be aware that non-linear scaling can leak the information from other datasets to the training dataset and, in this case, the scaler should be prepared on the training data only.

In cases where we do not have many training samples, we could use **data augmentation**. One easy way to understand augmentation is in the case of images. Let's imagine that our model's goal is to detect rotten fruits — apples, for example. We will take a photo of an apple from different angles and predict whether it's rotten. We should get more pictures in this case, but let's assume that we cannot. What we could do is to take photos that we have, rotate, crop, and save those as worthy data too. This way, we have added more samples to the dataset, which can help with model generalization. In general, if we use augmentation, then it's only useful if the augmentations that we make are similar to variations that we could see in reality. For example, we may refrain from using a rotation when creating a model to detect road signs as they are not being rotated in real-life scenarios (in most cases, anyway). The case of a rotated road sign, however, is one you better

consider if you're making a self-driving car. Just because a bolt came loose on a stop sign, flipping it over, doesn't mean you no longer need to stop there!

How many samples do we need to train the model? There is no single answer to this question — one model might require just a few per class, and another may require a few million or billion. Usually, a few thousand per class will be necessary, and a few tens of thousands should be preferable to start. The difference depends on the data complexity and model size. If the model has to predict sensor data with 2 simple classes, for example, if an image contains a dark area or does not, hundreds of samples per class might be enough. To train on data with many features and several classes, tens of thousands of samples are what you should start with. If you're attempting to train a chatbot the intricacies of written language, then you're going to likely want at least millions of samples.

Supplementary Material: https://nnfs.io/ch13
Chapter code, further resources, and errata for this chapter.

Chapter 14

L1 and L2 Regularization

Regularization methods are those which reduce generalization error. The first forms of regularization that we'll address are **L1** and **L2 regularization**. L1 and L2 regularization are used to calculate a number (called a **penalty**) added to the loss value to penalize the model for large weights and biases. Large weights might indicate that a neuron is attempting to memorize a data element; generally, it is believed that it would be better to have many neurons contributing to a model's output, rather than a select few.

Forward Pass

L1 regularization's penalty is the sum of all the absolute values for the weights and biases. This is a linear penalty as regularization loss returned by this function is directly proportional to parameter values. L2 regularization's penalty is the sum of the squared weights and biases. This non-linear approach penalizes larger weights and biases more than smaller ones because of the square function used to calculate the result. In other words, L2 regularization is commonly used as it does not affect small parameter values substantially and does not allow the model to grow weights too large by heavily penalizing relatively big values. L1 regularization, because of its linear nature, penalizes small weights more than L2 regularization, causing the model to start being invariant to small inputs and variant only to the bigger ones. That's why L1 regularization is rarely used alone and usually combined with L2 regularization if it's even used at all. Regularization functions of this type drive the sum of weights and the sum of parameters towards *0*, which can also help in cases of exploding gradients (model instability, which might cause weights to become very large values). Beyond this, we also want to dictate how much of an impact we want this regularization penalty to carry. We use a value referred to as **lambda** in this equation — where a higher value means a more significant penalty.

L1 weight regularization:

$$L_{1w} = \lambda \sum_{m} |w_m|$$

L1 bias regularization:

$$L_{1b} = \lambda \sum_{n} |b_n|$$

L2 weight regularization:

$$L_{2w} = \lambda \sum_{m} w_m^2$$

L2 bias regularization:

$$L_{2b} = \lambda \sum_n b_n^2$$

Overall loss:

$$Loss = DataLoss + L_{1w} + L_{1b} + L_{2w} + L_{2b}$$

Using code notation:

```
ĬĝŃ ł ĬĞİĠĤĞçĬĝŃ ő ĹłİâĞĠĹâŃĦĬÍÎĻĹãã
ĬĝĠ ł ĬĞİĠĤĞçĬĝĠ ő ĹłİâĞĠĹâĠĬĞĹĦĹãã
ĬğŃ ł ĬĞİĠĤĞçĬğŃ ő ĹłİâŃĦĬÍÎĻĹőőğã
ĬğĠ ł ĬĞİĠĤĞçĬğĠ ő ĹłİâĠĬĞĹĦĹőőğã
ĬIJĹĹ ł ĤĞĻĞçĬIJĹĹ κ ĬĝŃ κ ĬĝĠ κ ĬğŃ κ ĬğĠ
```

Regularization losses are calculated separately, then summed with the data loss, to form the overall loss. Parameter *m* is an arbitrary iterator over all of the weights in a model, parameter *n* is the bias equivalent of this iterator, w_m is the given weight, and b_n is the given bias.

To implement regularization in our neural network code, we'll start with the ççĬĮĬĻçç method of the DĦĮĹĦ layer's class, which will house the **lambda** regularization strength hyperparameters, since these can be set separately for every layer:

```
    ń LĞÑĦĹ ĬĮĬĻĬĞĬĬŃĞĻĬIJĮ
    ÎĦÌ ççĬĮĬĻççâĹĦĬÌd ĮçĬĮĴłĻĹd ĮçĮĦłĹIJĮĹd
                ŃĦĬÍÎĻçĹĦÍłĬĞĹĬŃĦĹçĬĝłfld ŃĦĬÍÎĻçĹĦÍłĬĞĹĬŃĦĹçĬğłfld
                ĠĬĞĹçĹĦÍłĬĞĹĬŃĦĹçĬĝłfld ĠĬĞĹçĹĦÍłĬĞĹĬŃĦĹçĬğłflãf
        ń IĮĬĻĬĞĬĬŃĦ ŃĦĬÍÎĻĹ ĞĮĤ ĠĬĞĹĦĹ
        ĹĦĬÌgŃĦĬÍÎĻĹ ł flgflĝ ő ĮĴgĹĞĮĤIJĠgĹĞĮĤĮâĬĮĴłĻĹd ĮĦłĹIJĮĹã
        ĹĦĬÌgĠĬĞĹĦĹ ł ĮĴgŃĦĹIJĹââĝd ĮĦłĹIJĮĹãã
        ń SĦĻ ĹĦÍłĬĞĹĬŃĞĻĬIJĮ ĹĻĹĦĮÍĻÎ
        ĹĦĬÌgŃĦĬÍÎĻçĹĦÍłĬĞĹĬŃĦĹçĬĝ ł ŃĦĬÍÎĻçĹĦÍłĬĞĹĬŃĦĹçĬĝ
        ĹĦĬÌgŃĦĬÍÎĻçĹĦÍłĬĞĹĬŃĦĹçĬğ ł ŃĦĬÍÎĻçĹĦÍłĬĞĹĬŃĦĹçĬğ
        ĹĦĬÌgĠĬĞĹçĹĦÍłĬĞĹĬŃĦĹçĬĝ ł ĠĬĞĹçĹĦÍłĬĞĹĬŃĦĹçĬĝ
        ĹĦĬÌgĠĬĞĹçĹĦÍłĬĞĹĬŃĦĹçĬğ ł ĠĬĞĹçĹĦÍłĬĞĹĬŃĦĹçĬğ
```

This method sets the lambda hyperparameters. Now we update our loss class to include the additional penalty if we choose to set the lambda hyperparameter for any of the regularizers in the layer's initialization. We will implement this code into the *Loss* class as it is common for the hidden layers. What's more, the regularization calculation is the same, regardless of

the type of loss used. It's only a penalty that is summed with the data loss value resulting in a final, overall loss value. For this reason, we're going to add a new method to a general loss class, which is inherited by all of our specific loss functions (such as our existing LIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑ). For the code of this method, we'll create the layer's regularization loss variable. We'll add to it each of the atomic regularization losses if its corresponding lambda value is greater than *0*. To perform these calculations, we read the lambda hyperparameters, weights, and biases from the passed-in layer object. For our general loss class:

```
ſi RĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽ ĢĞĬĢŁĬĞĻĨIJĮ
ÎĦÌ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽâĹĦĬÌd ĬĞÑĦĹãf

    ſi ſl ĠÑ ÎĦÌĞŁĬĻ
    ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ł ſl

    ſi Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ŃĦĨÍÎĻĹ
    ſi ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ ſl
    ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń ſlf
        ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ấ ċ
                                ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgŃĦĨÍÎĻĽãã

    ſi Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ŃĦĨÍÎĻĹ
    ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ſlf
        ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ấ ċ
                                ĮĴgĽŁİâĬĞÑĦĹgŃĦĨÍÎĻĽ ấ ċ
                                        ĬĞÑĦĹgŃĦĨÍÎĻĽã

    ſi Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĹ
    ſi ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ ſl
    ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń ſlf
        ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ κł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ấ ċ
                                ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgĠĨĞĽĦĹãã

    ſi Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĹ
    ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ſlf
        ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ κł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ấ ċ
                                ĮĴgĽŁİâĬĞÑĦĹgĠĨĞĽĦĹ ấ ċ
                                        ĬĞÑĦĹgĠĨĞĽĦĹã

    ĹĦĻŁĹĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ
```

Then we'll calculate the regularization loss and add it to our calculated loss in the training loop:

```
ƒ CĞĬĢŁĬĞĻĦ ĬIJĹĽ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĿĞĻĨIJĮğ ĹIJ ĹIJÌĻİĞŃ ĞĢĻĨĿĞĻĨIJĮ
ĤĞĻĞçĬIJĹĽ ł ĬIJĹĽçÌŁĮĢĻĨIJĮgÌIJĹŃĞĹĤâĞĢĻĨĿĞĻĨIJĮğgIJŁĻĴŁĻd Ñã

ƒ CĞĬĢŁĬĞĻĦ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĴĦĮĞĬĻÑ
ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ł ĬIJĹĽçÌŁĮĢĻĨIJĮgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽâĤĦĮĹĦĝã κ ċ
                        ĬIJĹĽçÌŁĮĢĻĨIJĮgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽâĤĦĮĹĦğã

ƒ CĞĬĢŁĬĞĻĦ IJĿĦĹĞĬĬ ĬIJĹĽ
ĬIJĹĽ ł ĤĞĻĞçĬIJĹĽ κ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ
```

We created a new `ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ` variable and added all layer's regularization losses to it. This completes the forward pass for regularization, but this also means our overall loss has changed since part of the calculation can include regularization, which must be accounted for in the backpropagation of the gradients. Thus, we will now cover the partial derivatives for both L1 and L2 regularization.

Backward pass

The derivative of L2 regularization is relatively simple:

$$L_{2w} = \lambda \sum_m w_m^2 \quad \rightarrow \quad \frac{\partial L_{2w}}{\partial w_m} = \frac{\partial}{\partial w_m}[\lambda \sum_m w_m^2] =$$

$$= \lambda \frac{\partial}{\partial w_m} w_m^2 = \lambda \cdot 2w_m^{2-1} = 2\lambda w_m$$

This might look complicated, but is one of the simpler derivative calculations that we have to derive in this book. Lambda is a constant, so we can move it outside of the derivative term. We can remove the sum operator since we calculate the partial derivative with respect to the given parameter only, and the sum of one element equals this element. So, we only need to calculate the derivative of w^2, which we know is $2w$. From the coding perspective, we will multiply all of the weights by 2λ. We'll implement this with NumPy directly as it's just a simple multiplication operation.

L1 regularization's derivative, on the other hand, requires more explanation. In the case of L1 regularization, we must calculate the derivative of the absolute value piecewise function, which effectively multiplies a value by -1 if it is less than 0; otherwise, it's multiplied by 1. This is because the absolute value function is linear for positive values, and we know that a linear function's derivative is:

$$f(x) = x \quad \rightarrow \quad f'(x) = 1$$

For negative values, it negates the sign of the value to make it positive. In other words, it multiplies values by -1:

$$f(x) = -x \quad \rightarrow \quad f'(x) = -1$$

When we combine that:

$$abs(x) = \begin{cases} x & x > 0 \\ -x & x < 0 \end{cases} \quad \rightarrow \quad abs'(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \end{cases}$$

And the complete partial derivative of L1 regularization with respect to given weight:

$$L_{1w} = \lambda \sum_m |w_m| \quad \rightarrow \quad L'_{1w} = \frac{\partial}{\partial w_m} \lambda \sum_m |w_m| = \lambda \frac{\partial}{\partial w_m} |w_m| = \lambda \begin{cases} 1 & w_m > 0 \\ -1 & w_m < 0 \end{cases}$$

Like L2 regularization, lambda is a constant, and we calculate the partial derivative of this regularization with respect to the specific input. The partial derivative, in this case, equals 1 or -1 depending on the w_m (weight) value.
We are calculating this derivative with respect to weights, and the resulting gradient, which has the same shape as the weights, is what we'll use to update the weights. To put this into pure Python code:

```
ŃĦĨÍÎĻĽ ł äflgğd flgíd hflgĥā  fi ŃĦĨÍÎĻĽ IJÌ IJĮĦ ĮĦŁĹIJĮ
ĤLĝ ł äā  fi ĞĹĹĞÑ IJÌ ĴĞĹĻĨĞĬ ĤĦĹĨĿĞĻĨĿĦĽ IJÌ Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ
ÌIJĹ ŃĦĨÍÎĻ ĨĮ ŃĦĨÍÎĻĽf
    ĨÌ ŃĦĨÍÎĻ ńł flf
        ĤLĝgĞĴĴĦĮĤâĝã
    ĦĬĽĦf
        ĤLĝgĞĴĴĦĮĤâhĝã
ĴĹĨĮĻâĤLĝã


ńńń
äĝd ĝd hĝā
```

You may have noticed that we're using `ńł fl` in the code where the equation above clearly depicts > *0*. If we picture the `ĮĴgĞĠĽ` function, it's a line going down and "bouncing" at the value *0*, like a saw tooth. At the pointed end (i.e., the value of *0*), the derivative of the `ĮĴgĞĠĽ` function is undefined, but we cannot code it this way, so we need to handle this situation and break this rule a bit.

Now let's try to modify this L1 derivative to work with multiple neurons in a layer:

```
ŃĦĨÍÎĻĽ ł ääflgğd flgíd hflgĥd ĝād  fi ĮIJŃ ŃĦ ÎĞŁĦ ġ ĽĦĻĽ IJÌ ŃĦĨÍÎĻĽ
           äflgĥd hflgîĝd flgğħd hflgĥād
           ähflgğħd hflgğìd flgĝìd flgíìāā
ĤLĝ ł āā  fi ĞĹĹĞÑ IJÌ ĴĞĹĻĨĞĬ ĤĦĹĨŁĞĻĨŁĦĽ IJÌ Lĝ ĹĦÍłĬĞĹĨŅĞĻĨIJĮ
ÌIJĹ ĮĦłĹIJĮ ĨĮ ŃĦĨÍÎĻĽf
    ĮĦłĹIJĮçĤLĝ ł āā  fi ĤĦĹĨŁĞĻĨŁĦĽ ĹĦĬĞĻĦĤ ĻIJ IJĮĦ ĮĦłĹIJĮ
    ÌIJĹ ŃĦĨÍÎĻ ĨĮ ĮĦłĹIJĮf
        ĨÌ ŃĦĨÍÎĻ ńł flf
            ĮĦłĹIJĮçĤLĝgĞĴĴĦĮĤâĝã
        ĦĬĽĦf
            ĮĦłĹIJĮçĤLĝgĞĴĴĦĮĤâhĝã
    ĤLĝgĞĴĴĦĮĤâĮĦłĹIJĮçĤLĝã
ĴĹĨĮĻâĤLĝã


ńńń
ääĝd ĝd hĝd ĝād äĝd hĝd ĝd hĝād ähĝd hĝd ĝd ĝāā
```

That's the vanilla Python version, now for the NumPy version. With NumPy, we're going to use conditions and binary masks. We'll create the gradient as an array filled with values of *1* and shaped like weights, using `ĮĴgIJĮĦĽçĬĨĨĦâŃĦĨÍÎĻĽã`. Next, the condition `ŃĦĨÍÎĻĽ l· fl` returns an array of the same shape as `ĤLĝ`, containing `fl` where the condition is false and `ĝ` where it's true. We're using this as a binary mask to `ĤLĝ` to set values to *-1* only where the condition is true (where weight values are less than 0):

```
ĨĬĴIJĹĻ ĮłĬĴÑ ĞĽ ĮĴ

ŃĦĨÍÎĻĽ ł ĮĴgĞĹĹĞÑâääflgğd flgíd hflgĥd ĝād
                     äflgĥd hflgîĝd flgğħd hflgĥād
                     ähflgğħd hflgğìd flgĝìd flgíìāāā

ĤLĝ ł ĮĴgIJĮĦĽçĬĨĨĦâŃĦĨÍÎĻĽã

ĤLĝäŃĦĨÍÎĻĽ l· flā ł hĝ

ĴĹĨĮĻâĤLĝã


ńńń
ĞĹĹĞÑâää ĝgd  ĝgd hĝgd  ĝgād
       ä ĝgd hĝgd  ĝgd hĝgād
       ähĝgd hĝgd  ĝgd  ĝgāāã
```

This returned an array of the same shape containing values of 1 and -1 — the partial gradient of the `ĮĴgĞĠĽ` function (we still have to multiply it by the lambda hyperparameter). We can now

take these and update the backward pass method for the dense layer object. For L1 regularization, we'll take the code above and multiply it by λ for weights and perform the same operation for biases. For L2 regularization, as discussed at the beginning of this chapter, all we need to do is take the weights/biases, multiply them by 2λ, and add that product to the gradients:

```
ſı DĦĮĽĦ LĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf
    ggg
    ſı BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ĠĞĢĪŃĞĹÎâĽĦĬÌd ÎĹĞĬŁĦĽãf
        ſı GĹĞÎĨĦĮĻĽ IJĮ ĴĞĹĞİĦĻĦĹĽ
        ĽĦĬÌgÎŃĦĨÍÎĻĽ ł ĮĴgÎIJĻâĽĦĬÌgĨĮĴŁĻĽgTd ÎĹĞĬŁĦĽã
        ĽĦĬÌgÎĠĬĞĽĦĽ ł ĮĴgĽŁİâÎĹĞĬŁĦĽd ĞŃĨĽłfld ĪĦĦĴÎĨĬĽłTĹŁĦã

        ſı GĹĞÎĨĦĮĻĽ IJĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ
        ſı Lĝ IJĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ÎĹĝ ł ĮĴgIJĮĦĽçĬĨĨĦâĽĦĬÌgŃĦĨÍÎĻĽã
            ÎĹĝäĽĦĬÌgŃĦĨÍÎĻĽ l· flā ł hĝ
            ĽĦĬÌgÎŃĦĨÍÎĻĽ κł ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ő ÎĹĝ
        ſı Lğ IJĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĽĦĬÌgÎŃĦĨÍÎĻĽ κł ğ ő ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                ĽĦĬÌgŃĦĨÍÎĻĽ
        ſı Lĝ IJĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ÎĹĝ ł ĮĴgIJĮĦĽçĬĨĨĦâĽĦĬÌgĠĨĞĽĦĽã
            ÎĹĝäĽĦĬÌgĠĨĞĽĦĽ l· flā ł hĝ
            ĽĦĬÌgÎĠĨĞĽĦĽ κł ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ő ÎĹĝ
        ſı Lğ IJĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĽĦĬÌgÎĠĨĞĽĦĽ κł ğ ő ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                ĽĦĬÌgĠĨĞĽĦĽ

        ſı GĹĞÎĨĦĮĻ IJĮ ĹĞĬŁĦĽ
        ĽĦĬÌgÎĨĮĴŁĻĽ ł ĮĴgÎIJĻâÎĹĞĬŁĦĽd ĽĦĬÌgŃĦĨÍÎĻĽgTã


With this, we can update our print to include new information — regularization loss and overall
loss:

ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
      ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
      ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď κ
      ÌďÎĞĻĞçĬIJĽĽf ăÎĞĻĞçĬIJĽĽfgġÌåd ď κ
      ÌďĹĦÍçĬIJĽĽf ăĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽfgġÌåãd ď κ
      ÌďĬĹf ăIJĴĻĬĬĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĬĮÍçĹĞĻĦåďã
```

Then we can add weight and bias regularizer parameters when defining a layer:

```
# CLHĞĻH DHĮĽH ĬĞÑHĹ ŃĨĻĨ ğ ĨĮĴŁĻ ÌHĞĻŁĹHĽ ĞĮĤ ġ IJŁĻĴŁĻ ĿĞĬŁHĽ
ĤHĮĽHĝ ł LĞÑHĹçDHĮĽHâğd ħǵd ŃHĨÍÎĻçĹHÍŁĬĞĹĨŃHĹçĬğłĥHHǵd
                            ĠĨĞĽçĹHÍŁĬĞĹĨŃHĹçĬğłĥHHǵã
```

We usually add regularization terms to the hidden layers only. Even if we are calling the regularization method on the output layer as well, it won't modify gradients if we do not set the lambda hyperparameters to values other than *0*.

Full code up to this point:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽHĻĽ ĨİĴIJĹĻ ĽĴĨĹĞĬçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


# DHĮĽH ĬĞÑHĹ
ĢĬĞĽĽ LĞÑHĹçDHĮĽHf

    # LĞÑHĹ ĨĮĨĻĨĞĬĨŃĞĻĨIJĮ
    ÂHÌ ççĨĮĨĻççâĽHĬÌd ĮçĨĮĴŁĻĽd ĮçĮHŁĹIJĮĽd
                 ŃHĨÍÎĻçĹHÍŁĬĞĹĨŃHĹçĬĝłfld ŃHĨÍÎĻçĹHÍŁĬĞĹĨŃHĹçĬğłfld
                 ĠĨĞĽçĹHÍŁĬĞĹĨŃHĹçĬĝłfld ĠĨĞĽçĹHÍŁĬĞĹĨŃHĹçĬğłflãf
        # IĮĨĻĨĞĬĨŃH ŃHĨÍÎĻĽ ĞĮĤ ĠĨĞĽHĽ
        ĽHĬÌgŃHĨÍÎĻĽ ł flgflĝ ǻ ĮĴgĹĞĮĤIJĮgĹĞĮĤĮâĮçĨĮĴŁĻĽd ĮçĮHŁĹIJĮĽã
        ĽHĬÌgĠĨĞĽHĽ ł ĮĴgŃHĹIJĽââĝd ĮçĮHŁĹIJĮĽãã
        # SHĻ ĹHÍŁĬĞĹĨŃĞĻĨIJĮ ĽĻĹHĮÍĻÎ
        ĽHĬÌgŃHĨÍÎĻçĹHÍŁĬĞĹĨŃHĹçĬĝ ł ŃHĨÍÎĻçĹHÍŁĬĞĹĨŃHĹçĬĝ
        ĽHĬÌgŃHĨÍÎĻçĹHÍŁĬĞĹĨŃHĹçĬğ ł ŃHĨÍÎĻçĹHÍŁĬĞĹĨŃHĹçĬğ
        ĽHĬÌgĠĨĞĽçĹHÍŁĬĞĹĨŃHĹçĬĝ ł ĠĨĞĽçĹHÍŁĬĞĹĨŃHĹçĬĝ
        ĽHĬÌgĠĨĞĽçĹHÍŁĬĞĹĨŃHĹçĬğ ł ĠĨĞĽçĹHÍŁĬĞĹĨŃHĹçĬğ

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽãf
        # RHİHİĠHĹ ĨĮĴŁĻ ĿĞĬŁHĽ
        ĽHĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        # CĞĬĢŁĬĞĻH IJŁĻĴŁĻ ĿĞĬŁHĽ ÌĹIJİ ĨĮĴŁĻĽd ŃHĨÍÎĻĽ ĞĮĤ ĠĨĞĽHĽ
        ĽHĬÌgIJŁĻĴŁĻ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĽHĬÌgŃHĨÍÎĻĽã к ĽHĬÌgĠĨĞĽHĽ
```

```
    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ÂĽĞĬŁHĽãf
        # GĹĞÂĨHĮĻĽ IJĮ ĴĞĹĞİHĻHĹĽ
        ĽHĬÌgĤŃHĨÍÎĻĽ ł ĮĴgĤIJĻâĽHĬÌgĨĮĴŁĻĽgTd ÂĽĞĬŁHĽã
        ĽHĬÌgĤĠĨĞĽHĽ ł ĮĴgĽŁİâÂĽĞĬŁHĽd ĞŃĨĽłfld ĪHHĴÂĨİĽłTĹŁHã

        # GĹĞÂĨHĮĻĽ IJĮ ĹHÍŁĬĞĹĨŅĞĻĨIJĮ
        # Lĝ IJĮ ŃHĨÍÎĻĽ
        ĨÌ ĽHĬÌgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬĝ ń flf
            ÂLĝ ł ĮĴgIJĮHĽçĬĨĨHâĽHĬÌgŃHĨÍÎĻĽã
            ÂLĝäĽHĬÌgŃHĨÍÎĻĽ l· flā ł hĝ
            ĽHĬÌgĤŃHĨÍÎĻĽ κł ĽHĬÌgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬĝ ấ ÂLĝ
        # Lğ IJĮ ŃHĨÍÎĻĽ
        ĨÌ ĽHĬÌgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬğ ń flf
            ĽHĬÌgĤŃHĨÍÎĻĽ κł ğ ấ ĽHĬÌgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬğ ấ ċ
                                ĽHĬÌgŃHĨÍÎĻĽ
        # Lĝ IJĮ ĠĨĞĽHĽ
        ĨÌ ĽHĬÌgĠĨĞĽçĹHÍŁĬĞĹĨŅHĹçĬĝ ń flf
            ÂLĝ ł ĮĴgIJĮHĽçĬĨĨHâĽHĬÌgĠĨĞĽHĽã
            ÂLĝäĽHĬÌgĠĨĞĽHĽ l· flā ł hĝ
            ĽHĬÌgĤĠĨĞĽHĽ κł ĽHĬÌgĠĨĞĽçĹHÍŁĬĞĹĨŅHĹçĬĝ ấ ÂLĝ
        # Lğ IJĮ ĠĨĞĽHĽ
        ĨÌ ĽHĬÌgĠĨĞĽçĹHÍŁĬĞĹĨŅHĹçĬğ ń flf
            ĽHĬÌgĤĠĨĞĽHĽ κł ğ ấ ĽHĬÌgĠĨĞĽçĹHÍŁĬĞĹĨŅHĹçĬğ ấ ċ
                                ĽHĬÌgĠĨĞĽHĽ

        # GĹĞÂĨHĮĻ IJĮ ĽĞĬŁHĽ
        ĽHĬÌgĤĨĮĴŁĻĽ ł ĮĴgĤIJĻâÂĽĞĬŁHĽd ĽHĬÌgŃHĨÍÎĻĽgTã


# RHLU ĞĢĻĨĽĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĽĞĻĨIJĮçRHLUf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽãf
        # RHİHİĠHĹ ĨĮĴŁĻ ĽĞĬŁHĽ
        ĽHĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        # CĞĬĢŁĬĞĻH IJŁĻĴŁĻ ĽĞĬŁHĽ ÌĹIJİ ĨĮĴŁĻĽ
        ĽHĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĨİŁİâfld ĨĮĴŁĻĽã

    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ÂĽĞĬŁHĽãf
        # SĨĮĢH ŃH ĮHHÂ ĻIJ İIJÂĨÌÑ IJĹĨÍĨĮĞĬ ĽĞĹĨĞĠĬHd
        # ĬHĻďĽ İĞĪH Ğ ĢIJĴÑ IJÌ ĽĞĬŁHĽ ÌĨĹĽĻ
        ĽHĬÌgÂĨĮĴŁĻĽ ł ÂĽĞĬŁHĽgĢIJĴÑâã

        # ZHĹIJ ÍĹĞÂĨHĮĻ ŃÎHĹH ĨĮĴŁĻ ĽĞĬŁHĽ ŃHĹH ĮHÍĞĻĨĽH
        ĽHĬÌgÂĨĮĴŁĻĽäĽHĬÌgĨĮĴŁĻĽ l·ł flā ł fl
```

```
# SIJÌĻİĞŇ ĞĢĻĨĹĞĻĨIJĮ
ĢĬĞĹĹ AĢĻĨĹĞĻĨIJĮçSIJÌĻİĞŇf

    # FIJĹŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ÌIJĹŃĞĹÂâĹĦĬÌd ĨĮĴŁĻĹãf
        # RĦİĦİĠĦĹ ĨĮĴŁĻ ĹĞĬŁĦĹ
        ĹĦĬÌgĨĮĴŁĻĹ ł ĨĮĴŁĻĹ

        # GĦĻ ŁĮIJĹİĞĬĨŅĦÂ ĴĹIJĠĞĠĨĬĨĻĨĦĹ
        ĦŇĴçĹĞĬŁĦĹ ł ĮĴgĦŇĴâĨĮĴŁĻĹ h ĮĴgİĞŇâĨĮĴŁĻĹd ĞŇĨĹłĝd
                                                ĪĦĦĴÂĨİĹłTĹŁĦãã
        # NIJĹİĞĬĨŅĦ ĻÎĦİ ÌIJĹ ĦĞĢÎ ĹĞİĴĬĦ
        ĴĹIJĠĞĠĨĬĨĻĨĦĹ ł ĦŇĴçĹĞĬŁĦĹ w ĮĴgĹŁİâĦŇĴçĹĞĬŁĦĹd ĞŇĨĹłĝd
                                                ĪĦĦĴÂĨİĹłTĹŁĦã

        ĹĦĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĨĬĨĻĨĦĹ

    # BĞĢĨŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ĠĞĢĨŃĞĹÂâĹĦĬÌd ÂĹĞĬŁĦĹãf

        # CĹĦĞĻĦ ŁĮĨĮĨĻĨĞĬĨŅĦÂ ĞĹĹĞÑ
        ĹĦĬÌgÂĨĮĴŁĻĹ ł ĮĴgĦİĴĻÑçĬĨĨĦâÂĹĞĬŁĦĹã

        # EĮŁİĦĹĞĻĦ IJŁĻĴŁĻĹ ĞĮÂ ÍĹĞÂĨĦĮĻĹ
        ÌIJĹ ĨĮÂĦŇd âĹĨĮÍĬĦçIJŁĻĴŁĻd ĹĨĮÍĬĦçÂĹĞĬŁĦĹã ĨĮ ċ
                ĦĮŁİĦĹĞĻĦâŅĨĴâĹĦĬÌgIJŁĻĴŁĻd ÂĹĞĬŁĦĹãã f
            # FĬĞĻĻĦĮ IJŁĻĴŁĻ ĞĹĹĞÑ
            ĹĨĮÍĬĦçIJŁĻĴŁĻ ł ĹĨĮÍĬĦçIJŁĻĴŁĻgĹĦĹÎĞĴĦâhĝd ĝã
            # CĞĬĢŁĬĞĻĦ JĞĢIJĠĨĞĮ İĞĻĹĨŇ IJÌ ĻÎĦ IJŁĻĴŁĻ ĞĮÂ
            ĬĞĢIJĠĨĞĮçİĞĻĹĨŇ ł ĮĴgÂĨĞÍÌĬĞĻâĹĨĮÍĬĦçIJŁĻĴŁĻã h ċ
                                ĮĴgÂIJĻâĹĨĮÍĬĦçIJŁĻĴŁĻd ĹĨĮÍĬĦçIJŁĻĴŁĻgTã
            # CĞĬĢŁĬĞĻĦ ĹĞİĴĬĦhŃĨĹĦ ÍĹĞÂĨĦĮĻ
            # ĞĮÂ ĞÂÂ ĨĻ ĻIJ ĻÎĦ ĞĹĹĞÑ IJÌ ĹĞİĴĬĦ ÍĹĞÂĨĦĮĻĹ
            ĹĦĬÌgÂĨĮĴŁĻĹäĨĮÂĦŇã ł ĮĴgÂIJĻâĬĞĢIJĠĨĞĮçİĞĻĹĨŇd
                                        ĹĨĮÍĬĦçÂĹĞĬŁĦĹã


# SGD IJĴĻĨİĨŅĦĹ
ĢĬĞĹĹ OĴĻĨİĨŅĦĹçSGDf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮÍĹd
    # ĬĦĞĹĮĨĮÍ ĹĞĻĦ IJÌ ĝg ĨĹ ÂĦÌĞŁĬĻ ÌIJĹ ĻÎĨĹ IJĴĻĨİĨŅĦĹ
    ÂĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ÂĦĢĞŇłflgd İIJİĦĮĻŁİłflgãf
        ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĢŁĹĹĦĮçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgÂĦĢĞŇ ł ÂĦĢĞŇ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgİIJİĦĮĻŁİ ł İIJİĦĮĻŁİ
```

```
# CĞĬĬ IJĮĢH ĠHÌIJĹH ĞĮÑ ĴĞĹĞİHĻHĹ ŁĴĤĞĻHĹ
ĤHÌ ĴĹHçŁĴĤĞĻHçĴĞĹĞİĹâĹHĬÌãf
    ĨÌ ĹHĬÌgĤHĢĞÑf
        ĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ł ĹHĬÌgĬHĞĹĮĨĮÍçĹĞĻH ấ ċ
            âĝg w âĝg κ ĹHĬÌgĤHĢĞÑ ấ ĹHĬÌgĨĻHĹĞĻĨIJĮĹãã

# UĴĤĞĻH ĴĞĹĞİHĻHĹĹ
ĤHÌ ŁĴĤĞĻHçĴĞĹĞİĹâĹHĬÌd ĬĞÑHĹãf

    # IÌ ŃH ŁĹH İIJİHĮĻŁİ
    ĨÌ ĹHĬÌgİIJİHĮĻŁİf

        # IÌ ĬĞÑHĹ ĤIJHĹ ĮIJĻ ĢIJĮĻĞĨĮ İIJİHĮĻŁİ ĞĹĹĞÑĹd ĢĹHĞĻH ĻĤHİ
        # ÌĨĬĬHĤ ŃĨĻĤ ŅHĹIJĹ
        ĨÌ ĮIJĻ ĤĞĹĞĻĻĹâĬĞÑHĹd ďŃHĨÍĤĻçİIJİHĮĻŁİĹďãf
            ĬĞÑHĹgŃHĨÍĤĻçİIJİHĮĻŁİĹ ł ĮĴgŅHĹIJĹçĬĨĬHâĬĞÑHĹgŃHĨÍĤĻĹã
            # IÌ ĻĤHĹH ĨĹ ĮIJ İIJİHĮĻŁİ ĞĹĹĞÑ ÌIJĹ ŃHĨÍĤĻĹ
            # TĤH ĞĹĹĞÑ ĤIJHĹĮďĻ HŃĨĹĻ ÌIJĹ ĠĨĞĹHĹ ÑHĻ HĨĻĤHĹg
            ĬĞÑHĹgĠĨĞĹçİIJİHĮĻŁİĹ ł ĮĴgŅHĹIJĹçĬĨĬHâĬĞÑHĹgĠĨĞĹHĹã

        # BŁĨĬĤ ŃHĨÍĤĻ ŁĴĤĞĻHĹ ŃĨĻĤ İIJİHĮĻŁİ h ĻĞĨH ĴĹHŁĨIJŁĹ
        # ŁĴĤĞĻHĹ İŁĬĻĨĴĬĨHĤ ĠÑ ĹHĻĞĨĮ ÌĞĢĻIJĹ ĞĮĤ ŁĴĤĞĻH ŃĨĻĤ
        # ĢŁĹĹHĮĻ ÍĹĞĤĨHĮĻĹ
        ŃHĨÍĤĻçŁĴĤĞĻHĹ ł ċ
            ĹHĬÌgİIJİHĮĻŁİ ấ ĬĞÑHĹgŃHĨÍĤĻçİIJİHĮĻŁİĹ h ċ
            ĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ấ ĬĞÑHĹgĤŃHĨÍĤĻĹ
        ĬĞÑHĹgŃHĨÍĤĻçİIJİHĮĻŁİĹ ł ŃHĨÍĤĻçŁĴĤĞĻHĹ

        # BŁĨĬĤ ĠĨĞĹ ŁĴĤĞĻHĹ
        ĠĨĞĹçŁĴĤĞĻHĹ ł ċ
            ĹHĬÌgİIJİHĮĻŁİ ấ ĬĞÑHĹgĠĨĞĹçİIJİHĮĻŁİĹ h ċ
            ĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ấ ĬĞÑHĹgĤĠĨĞĹHĹ
        ĬĞÑHĹgĠĨĞĹçİIJİHĮĻŁİĹ ł ĠĨĞĹçŁĴĤĞĻHĹ

    # VĞĮĨĬĬĞ SGD ŁĴĤĞĻHĹ âĞĹ ĠHÌIJĹH İIJİHĮĻŁİ ŁĴĤĞĻHã
    HĬĹHf
        ŃHĨÍĤĻçŁĴĤĞĻHĹ ł hĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ấ ċ
                            ĬĞÑHĹgĤŃHĨÍĤĻĹ
        ĠĨĞĹçŁĴĤĞĻHĹ ł hĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ấ ċ
                           ĬĞÑHĹgĤĠĨĞĹHĹ

    # UĴĤĞĻH ŃHĨÍĤĻĹ ĞĮĤ ĠĨĞĹHĹ ŁĹĨĮÍ HĨĻĤHĹ
    # ŁĞĮĨĬĬĞ IJĹ İIJİHĮĻŁİ ŁĴĤĞĻHĹ
    ĬĞÑHĹgŃHĨÍĤĻĹ κł ŃHĨÍĤĻçŁĴĤĞĻHĹ
    ĬĞÑHĹgĠĨĞĹHĹ κł ĠĨĞĹçŁĴĤĞĻHĹ

# CĞĬĬ IJĮĢH ĞÌĻHĹ ĞĮÑ ĴĞĹĞİHĻHĹ ŁĴĤĞĻHĹ
ĤHÌ ĴIJĹĻçŁĴĤĞĻHçĴĞĹĞİĹâĹHĬÌãf
    ĹHĬÌgĨĻHĹĞĻĨIJĮĹ κł ĝ
```

```
# AĤĞÍĹĞĤ IJĴĻĨĬĬĨŅĦĹ
ĢĬĞĹĹ OĴĻĨĬĬĨŅĦĹçAĤĞÍĹĞĤf

    # IĮĨĻĨĞĬĬĨŅĦ IJĴĻĨĬĬĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮÍĹ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ĤĦĢĞÑłflgd ĦĴĹĨĬIJĮłĝĦhìãf
        ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgĦĴĹĨĬIJĮ ł ĦĴĹĨĬIJĮ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦĹ
    ĤĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgĤĦĢĞÑf
            ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĹĦĬÌgĤĦĢĞÑ ắ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    # UĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĹ
    ĤĦÌ ŁĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ĤIJĦĹ ĮIJĮ ĢIJĮĻĞĨĮ ĢĞĢĨĦ ĞĹĹĞÑĹd
        # ĢĹĦĞĻĦ ĻĨĦĬ ÌĨĬĬĦĤ ÑĨĻĨ ĢĦĹIJĹ
        ĨÌ ĮIJĮ ĨĞĹĞĻĮĹâĬĞÑĦĹd ďÑĦĨÍĨĻçĢĞĢĨĦďãf
            ĬĞÑĦĹgÑĦĨÍĨĻçĢĞĢĨĦ ł ĮĴgĢĦĹIJĹçĬĨĨĦâĬĞÑĦĹgÑĦĨÍĨĻĹã
            ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ ł ĮĴgĢĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã

        # UĴĤĞĻĦ ĢĞĢĨĦ ÑĨĻĨ ĹĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ÍĹĞĤĨĦĮĻĹ
        ĬĞÑĦĹgÑĦĨÍĨĻçĢĞĢĨĦ κł ĬĞÑĦĹgĤÑĦĨÍĨĻĹắắğ
        ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ κł ĬĞÑĦĹgĤĠĨĞĹĦĹắắğ

        # VĞĮĨĬĬĞ SGD ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦ κ IJĹĬĞĬĨŅĞĻĨIJĮ
        # ÑĨĻĨ ĹĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĨĦ
        ĬĞÑĦĹgÑĦĨÍĨĻĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                        ĬĞÑĦĹgĤÑĦĨÍĨĻĹ w ċ
                        âĮĴgĹĶĹĻâĬĞÑĦĹgÑĦĨÍĨĻçĢĞĢĨĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĹĦĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                       ĬĞÑĦĹgĤĠĨĞĹĦĹ w ċ
                       âĮĴgĹĶĹĻâĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã

    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦĹ
    ĤĦÌ ĴIJĹĻçŁĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌãf
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κł ĝ
```

```
# RMSPropOptimizer
class OptimizerRMSProp:

    # Initialize optimizer - set attributes
    def __init__(self, learning_rate=1.0, decay=0.0,
                 epsilon=1e-7, rho=0.9):
        self.learning_rate = learning_rate
        self.current_learning_rate = learning_rate
        self.decay = decay
        self.iterations = 0
        self.epsilon = epsilon
        self.rho = rho

    # Call once before any parameter updates
    def pre_update_params(self):
        if self.decay:
            self.current_learning_rate = self.learning_rate
                * (1. / (1. + self.decay * self.iterations))

    # Update parameters
    def update_params(self, layer):

        # If layer does not contain cache arrays,
        # create them filled with zeros
        if hasattr(layer, 'weight_cache'):
            layer.weight_cache = np.zeros_like(layer.weights)
            layer.bias_cache = np.zeros_like(layer.biases)

        # Update cache with squared current gradients
        layer.weight_cache = self.rho*layer.weight_cache
            + (1 - self.rho) * layer.dweights**2
        layer.bias_cache = self.rho*layer.bias_cache
            + (1 - self.rho) * layer.dbiases**2

        # Vanilla SGD parameter update + normalization
        # with square rooted cache
        layer.weights += -self.current_learning_rate
            * layer.dweights
            / (np.sqrt(layer.weight_cache) + self.epsilon)
        layer.biases += -self.current_learning_rate
            * layer.dbiases
            / (np.sqrt(layer.bias_cache) + self.epsilon)

# Create instance of optimizer
optimizer = OptimizerRMSProp(learning_rate=0.02, decay=1e-5,
                             rho=0.9)
# Loop through optimization steps
for epoch in range(10000):
    # Forward pass through layers
    dense1.forward(X)
    activation1.forward(dense1.output)
    dense2.forward(activation1.output)
    activation2.forward(dense2.output)
    # Calculate loss
    loss = loss_function.calculate(activation2.output, y)
    # Print epoch, loss, accuracy
    if not epoch:
        print('epoch: {:d}, loss: {:f}'.format(epoch, loss))
    # Backpropagation
    loss_function.backward(activation2.output, y)
    activation2.backward(loss_function.dinputs)
    dense2.backward(activation2.dinputs)
    activation1.backward(dense2.dinputs)
    dense1.backward(activation1.dinputs)
    # Update weights and biases
    optimizer.pre_update_params()
    optimizer.update_params(dense1)
    optimizer.update_params(dense2)
    optimizer.post_update_params()
```

```
# AĤĞİ IJĴĻĨİĨĨŅĦĹ
ĢĬĞĹĹ OĴĻĨİĨĨŅĦĹçAĤĞİf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮĬĹ
    ÎĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮĬçĹĞĻĦłflgflflĝd ÎĦĢĞÑłflgd ĦĴĹĨĬIJĮłĝĦhìd
                ĠĦĻĞçĝłflgîd ĠĦĻĞçğłflgîîîãf
        ĹĦĬÌgĬĦĞĹĮĨĮĬçĹĞĻĦ ł ĬĦĞĹĮĨĮĬçĹĞĻĦ
        ĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ł ĬĦĞĹĮĨĮĬçĹĞĻĦ
        ĹĦĬÌgÎĦĢĞÑ ł ÎĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgĦĴĹĨĬIJĮ ł ĦĴĹĨĬIJĮ
        ĹĦĬÌgĠĦĻĞçĝ ł ĠĦĻĞçĝ
        ĹĦĬÌgĠĦĻĞçğ ł ĠĦĻĞçğ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ łĴÎĞĻĦĹ
    ÎĦÌ ĴĹĦçłĴÎĞĻĦçĴĞĹĞİĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgÎĦĢĞÑf
            ĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮĬçĹĞĻĦ ấ ċ
                âĝg w âĝg κ ĹĦĬÌgÎĦĢĞÑ ấ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    # UĴÎĞĻĦ ĴĞĹĞİĦĻĦĹĹ
    ÎĦÌ łĴÎĞĻĦçĴĞĹĞİĹâĹĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ÎIJĦĹ IJIJĻ ĢIJĮĻĞĨĮ ĢĞĢÎĦ ĞĹĹĞÑĹd
        # ĢĹĦĞĻĦ ĻÎĦİ ÌĨĬĬĦÎ ŃĨĻÎ ŅĦĹIJĹ
        ĨÌ IJIJĻ ÎĞĹĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍÎĻçĢĞĢÎĦďãf
            ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦIJĮłİĹ ł IJĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgŃĦĨÍÎĻĹã
            ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł IJĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgŃĦĨÍÎĻĹã
            ĬĞÑĦĹgĠĨĞĹçİIJİĦIJĮłİĹ ł IJĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã
            ĬĞÑĦĹgĠĨĞĹçĢĞĢÎĦ ł IJĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã

        # UĴÎĞĻĦ İIJİĦIJĮłİ  ŃĨĻÎ ĢłĹĹĦIJĮ ÍĹĞÎĨĦIJĮĹ
        ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦIJĮłİĹ ł ĹĦĬÌgĠĦĻĞçĝ ấ ċ
                                    ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦIJĮłİĹ κ ċ
                                    âĝ h ĹĦĬÌgĠĦĻĞçĝã ấ ĬĞÑĦĹgÎŃĦĨÍÎĻĹ
        ĬĞÑĦĹgĠĨĞĹçİIJİĦIJĮłİĹ ł ĹĦĬÌgĠĦĻĞçĝ ấ ċ
                                ĬĞÑĦĹgĠĨĞĹçİIJİĦIJĮłİĹ κ ċ
                                âĝ h ĹĦĬÌgĠĦĻĞçĝã ấ ĬĞÑĦĹgÎĠĨĞĹĦĹ
        # GĦĻ ĢIJĹĹĦĢĻĦÎ İIJİĦIJĮłİ
        # ĹĦĬÌgĨĻĦĹĞĻĨIJĮ ĨĹ fl ĞĻ ÌĨĹĹĻ ĴĞĹĹ
        # ĞĮÎ ŃĦ IJĦĦÎ ĻIJ ĹĻĞĹĻ ŃĨĻÎ ĝ ÎĦĹĦ
        ŃĦĨÍÎĻçİIJİĦIJĮłİĹçĢIJĹĹĦĢĻĦÎ ł ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦIJĮłİĹ w ċ
            âĝ h ĹĦĬÌgĠĦĻĞçĝ ấấ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
        ĠĨĞĹçİIJİĦIJĮłİĹçĢIJĹĹĦĢĻĦÎ ł ĬĞÑĦĹgĠĨĞĹçİIJİĦIJĮłİĹ w ċ
            âĝ h ĹĦĬÌgĠĦĻĞçĝ ấấ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
        # UĴÎĞĻĦ ĢĞĢÎĦ ŃĨĻÎ ĹĶłĞĹĦÎ ĢłĹĹĦIJĮ ÍĹĞÎĨĦIJĮĹ
        ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĹĦĬÌgĠĦĻĞçğ ấ ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ κ ċ
            âĝ h ĹĦĬÌgĠĦĻĞçğã ấ ĬĞÑĦĹgÎŃĦĨÍÎĻĹấğ
```

```
        ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĽĦĬÌgĠĦĻĞçğ ǻ ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ κ ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğã ǻ ĬĞÑĦĹgĤĠĨĞĽĦĽǻǻğ
        ſ GĦĻ ĢIJĹĹĦĢĻĦĤ ĢĞĢÎĦ
        ŃĦĨÍÎĻçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğ ǻǻ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã
        ĠĨĞĽçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğ ǻǻ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã

        ſ VĞĮĨĬĬĞ SGD ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹĬĞĬĨŅĞĻĨIJĮ
        ſ ŃĨĻÎ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢÎĦ
        ĬĞÑĦĹgŃĦĨÍÎĻĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ǻ ċ
                        ŃĦĨÍÎĻçĬIJĬĦĮĻŁĬĽçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĽĶĹĻâŃĦĨÍÎĻçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤã κ
                            ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ǻ ċ
                        ĠĨĞĽçĬIJĬĦĮĻŁĬĽçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĽĶĹĻâĠĨĞĽçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤã κ
                            ĽĦĬÌgĦĴĽĨĬIJĮã

    ſ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞĬĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ


ſ CIJĬĬIJĮ ĬIJĽĽ ĢĬĞĽĽ
ĢĬĞĽĽ LIJĽĽf

    ſ RĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽ ĢĞĬĢŁĬĞĻĨIJĮ
    ĤĦÌ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽâĽĦĬÌd ĬĞÑĦĹãf

        ſ fl ĠÑ ĤĦÌĞŁĬĻ
        ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ ł fl

        ſ Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ŃĦĨÍÎĻĽ
        ſ ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ fl
        ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ǻ ċ
                                    ĮĴgĽŁĬâĮĴgĞĠĽâĬĞÑĦĹgŃĦĨÍÎĻĽãã

        ſ Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ŃĦĨÍÎĻĽ
        ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ǻ ċ
                                    ĮĴgĽŁĬâĬĞÑĦĹgŃĦĨÍÎĻĽ ǻ ċ
                                            ĬĞÑĦĹgŃĦĨÍÎĻĽã
```

```
        # Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
        # ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ fl
        ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ĸł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ǻ ċ
                                    ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgĠĨĞĽĦĽãã

        # Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
        ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ĸł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ǻ ċ
                                    ĮĴgĽŁİâĬĞÑĦĹgĠĨĞĽĦĽ ǻ ċ
                                           ĬĞÑĦĹgĠĨĞĽĦĽã

        ĹĦĻŁĹĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ

    # CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽĦĽ
    # ÍĨĹĦĮ ĬIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ĹĞĬŁĦĽ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñãf

        # CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦ ĬIJĹĽĦĽ
        ĽĞİĴĬĦçĬIJĹĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        # CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĹĽ
        ÂĞĻĞçĬIJĹĽ ł ĮĴgĬĦĞĮâĽĞİĴĬĦçĬIJĹĽĦĽã

        # RĦĻŁĹĮ ĬIJĹĽ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĽ


# CĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ
ĢĬĞĽĽ LIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâLIJĹĽãf

    # FIJŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJŃĞĹÂâĽĦĬÌd ÑçĴĹĦÂd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ ĨĮ Ğ ĠĞĻĢÎ
        ĽĞİĴĬĦĽ ł ĬĦĮâÑçĴĹĦÂã

        # CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ÂĨŁĨĹĨIJĮ ĠÑ fl
        # CĬĨĴ ĠIJĻÎ ĹĨÂĦĽ ĻIJ ĮIJĻ ÂĹĞÍ İĦĞĮ ĻIJŃĞĹÂĽ ĞĮÑ ŁĞĬŁĦ
        ÑçĴĹĦÂçĢĬĨĴĴĦÂ ł ĮĴgĢĬĨĴâÑçĴĹĦÂd ĝĦhìd ĝ h ĝĦhìã

        # PĹIJĠĞĠĨĬĨĻĨĦĽ ÌIJĹ ĻĞĹÍĦĻ ŁĞĬŁĦĽ h
        # IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĽ ł ÑçĴĹĦÂçĢĬĨĴĴĦÂä
                ĹĞĮÍĦâĽĞİĴĬĦĽãd
                ÑçĻĹŁĦ
            ã
```

```
        # MĞĽĪ ĽĞĬŁĦĽ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJĤĦĤ ĬĞĠĦĬĽ
        ĦĬĪÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ĢIJĹĹĦĢĻçĢIJĮÌĪĤĦĮĢĦĽ ł ĮĴgĽŁİâ
                ÑçĴĹĦĤçĢĬĪĴĴĦĤ ǻ ÑçĻĹŁĦd
                ĞŇĪĽłĝ
            ã

        # LIJĹĽĦĽ
        ĮĦÍĞĻĪĽĦçĬIJÍçĬĪĪĦĬĪÎIJIJĤĽ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĪĤĦĮĢĦĽã
        ĹĦĻŁĹĮ ĮĦÍĞĻĪĽĦçĬIJÍçĬĪĪĦĬĪÎIJIJĤĽ

    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĽĞĬŁĦĽd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâĤĽĞĬŁĦĽã
        # NŁİĠĦĹ IJÌ ĬĞĠĦĬĽ ĪĮ ĦĽĦĹÑ ĽĞİĴĬĦ
        # WĦďĬĬ ŁĽĦ ĻĤĦ ÌĪĹĽĻ ĽĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦİ
        ĬĞĠĦĬĽ ł ĬĦĮâĤĽĞĬŁĦĽäflãã

        # IÌ ĬĞĠĦĬĽ ĞĹĦ ĽĴĞĹĽĦd ĻŁĹĮ ĻĤĦİ ĪĮĻIJ IJĮĦhÎIJĻ ĽĦĢĻIJĹ
        ĪÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ÑçĻĹŁĦ ł ĮĴgĦÑĦâĬĞĠĦĬĽãäÑçĻĹŁĦã

        # CĞĬĢŁĬĞĻĦ ÍĹĞĤĪĦĮĻ
        ĽĦĬÌgĤĪĮĴŁĻĽ ł hÑçĻĹŁĦ w ĤĽĞĬŁĦĽ
        # NIJĹĬĞĬĪŅĦ ÍĹĞĤĪĦĮĻ
        ĽĦĬÌgĤĪĮĴŁĻĽ ł ĽĦĬÌgĤĪĮĴŁĻĽ w ĽĞİĴĬĦĽ


# SIJÌĻĬĞŇ ĢĬĞĽĽĪÌÌĪĦĹ h ĢIJĬĠĪĮĦĤ SIJÌĻĬĞŇ ĞĢĻĪĽĞĻĪIJĮ
# ĞĮĤ ĢĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ ÌIJĹ ÌĞĽĻĦĹ ĠĞĢĪŃĞĹĤ ĽĻĦĴ
ĢĬĞĽĽ AĢĻĪĽĞĻĪIJĮçSIJÌĻĬĞŇçLIJĽĽçCĞĻĦÍIJĹĪĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâãf

    # CĹĦĞĻĦĽ ĞĢĻĪĽĞĻĪIJĮ ĞĮĤ ĬIJĽĽ ÌŁĮĢĻĪIJĮ IJĠÏĦĢĻĽ
    ĤĦÌ ççĪĮĪĻççâĽĦĬÌãf
        ĽĦĬÌgĞĢĻĪĽĞĻĪIJĮ ł AĢĻĪĽĞĻĪIJĮçSIJÌĻĬĞŇâã
        ĽĦĬÌgĬIJĽĽ ł LIJĽĽçCĞĻĦÍIJĹĪĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĪĮĴŁĻĽd ÑçĻĹŁĦãf
        # OŁĻĴŁĻ ĬĞÑĦĹďĽ ĞĢĻĪĽĞĻĪIJĮ ÌŁĮĢĻĪIJĮ
        ĽĦĬÌgĞĢĻĪĽĞĻĪIJĮgÌIJĹŃĞĹĤâĪĮĴŁĻĽã
        # SĦĻ ĻĤĦ IJŁĻĴŁĻ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĽĦĬÌgĞĢĻĪĽĞĻĪIJĮgIJŁĻĴŁĻ
        # CĞĬĢŁĬĞĻĦ ĞĮĤ ĹĦĻŁĹĮ ĬIJĽĽ ĽĞĬŁĦ
        ĹĦĻŁĹĮ ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâĽĦĬÌgIJŁĻĴŁĻd ÑçĻĹŁĦã
```

```
    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĽĞĬŁĦĽd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâĤĽĞĬŁĦĽã

        # IÌ ĬĞĠĦĬĽ ĞĹĦ IJĮĦhÎIJĻ ĦĮĢIJĤĦĤd
        # ĻŁĹĮ ĻÎĦİ ĨĮĻIJ ĤĨĽĢĹĦĻĦ ĽĞĬŁĦĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ÑçĻĹŁĦ ł ĮĴgĞĹÍİĞÑâÑçĻĹŁĦd ĞÑĨĽłĝã

        # CIJĴÑ ĽIJ ŃĦ ĢĞĮ ĽĞÌĦĬÑ İIJĤĨÌÑ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĤĽĞĬŁĦĽgĢIJĴÑâã
        # CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽäĹĞĮÍĦâĽĞİĴĬĦĽãd ÑçĻĹŁĦã hł ĝ
        # NIJĹİĞĬĨÑĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽĞİĴĬĦĽ


# CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

# CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮĤ ħġ IJŁĻĴŁĻ ĽĞĬŁĦĽ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħġd ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦħġd
                              ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦħġã

# CĹĦĞĻĦ RĦLU ĞĢĻĨĽĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦĤ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĽĞĻĨIJĮĝ ł AĢĻĨĽĞĻĨIJĮçRĦLUâã

# CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħġ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJŁĻĴŁĻ
# IJÌ ĴĹĦĽĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ġ IJŁĻĴŁĻ ĽĞĬŁĦĽ âIJŁĻĴŁĻ ĽĞĬŁĦĽã
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâħġd ġã

# CĹĦĞĻĦ SIJÌĻİĞÑ ĢĬĞĽĽĨÌÌĦĹďĽ ĢIJİĠĨĮĦĤ ĬIJĽĽ ĞĮĤ ĞĢĻĨĽĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨĽĞĻĨIJĮ ł AĢĻĨĽĞĻĨIJĮçSIJÌĻİĞÑçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

# CĹĦĞĻĦ IJĴĻĨİĨÑĦĹ
IJĴĻĨİĨÑĦĹ ł OĴĻĨİĨÑĦĹçAĤĞİâĬĦĞĹĮĨĮÍçĹĞĻĦłflgflğd ĤĦĢĞÑłĥĦhìã

# TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝflflflĝãf

    # PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ĤĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑĦĹ
    ĤĦĮĽĦĝgÌIJĹŃĞĹĤâXã

    # PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    # ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ĤĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨĽĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĽĦĝgIJŁĻĴŁĻã
```

```
ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽĦĢIJĮÎ DĦĮĽĦ ĬĞÑĦĹ
ﬁ ĻĞĪĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĪĿĞĻĪIJĮ ÌŁĮĢĻĪIJĮ IJÌ ÌĪĹĽĻ ĬĞÑĦĹ ĞĽ ĪĮĴŁĻĽ
ÎĦĮĽĦğgÌIJĹŃĞĹÎâĞĢĻĪĿĞĻĪIJĮĝgIJŁĻĴŁĻã

ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĻÎĦ ĞĢĻĪĿĞĻĪIJĮwĬIJĽĽ ÌŁĮĢĻĪIJĮ
ﬁ ĻĞĪĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮÎ ÎĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮÎ ĹĦĻŁĹĮĽ ĬIJĽĽ
ÎĞĻĞçĬIJĽĽ ł ĬIJĽĽçĞĢĻĪĿĞĻĪIJĮgÌIJĹŃĞĹÎâÎĦĮĽĦğgIJŁĻĴŁĻd Ñã

ﬁ CĞĬĢŁĬĞĻĦ ĹĦÍŁĬĞĹĪŃĞĻĪIJĮ ĴĦĮĞĬĻÑ
ĹĦÍŁĬĞĹĪŃĞĻĪIJĮçĬIJĽĽ ł ċ
    ĬIJĽĽçĞĢĻĪĿĞĻĪIJĮgĬIJĽĽgĹĦÍŁĬĞĹĪŃĞĻĪIJĮçĬIJĽĽâÎĦĮĽĦĝã κ ċ
    ĬIJĽĽçĞĢĻĪĿĞĻĪIJĮgĬIJĽĽgĹĦÍŁĬĞĹĪŃĞĻĪIJĮçĬIJĽĽâÎĦĮĽĦğã

ﬁ CĞĬĢŁĬĞĻĦ IJĿĦĹĞĬĬ ĬIJĽĽ
ĬIJĽĽ ł ÎĞĻĞçĬIJĽĽ κ ĹĦÍŁĬĞĹĪŃĞĻĪIJĮçĬIJĽĽ

ﬁ CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĪĿĞĻĪIJĮğ ĞĮÎ ĻĞĹÍĦĻĽ
ﬁ ĢĞĬĢŁĬĞĻĦ ĿĞĬŁĦĽ ĞĬIJĮÍ ÌĪĹĽĻ ĞŃĪĽ
ĴĹĦÎĪĢĻĪIJĮĽ ł ĮĴgĞĹÍİĞŃâĬIJĽĽçĞĢĻĪĿĞĻĪIJĮgIJŁĻĴŁĻd ĞŃĪĽłĝã
ĪÌ ĬĦĮâÑgĽÎĞĴĦã łł ğf
    Ñ ł ĮĴgĞĹÍİĞŃâÑd ĞŃĪĽłĝã
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦÎĪĢĻĪIJĮĽłłÑã

ĪÌ ĮIJĻ ĦĴIJĢÎ ķ ĝﬂﬂf
    ĴĹĪĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
          ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
          ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď κ
          ÌďÎĞĻĞçĬIJĽĽf ăÎĞĻĞçĬIJĽĽfgġÌåd ď κ
          ÌďĹĦÍçĬIJĽĽf ăĹĦÍŁĬĞĹĪŃĞĻĪIJĮçĬIJĽĽfgġÌåãd ď κ
          ÌďĬĹf ăIJĴĻĪİĪŃĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĪĮÍçĹĞĻĦåďã

ﬁ BĞĢĪŃĞĹÎ ĴĞĽĽ
ĬIJĽĽçĞĢĻĪĿĞĻĪIJĮgĠĞĢĪŃĞĹÎâĬIJĽĽçĞĢĻĪĿĞĻĪIJĮgIJŁĻĴŁĻd Ñã
ÎĦĮĽĦğgĠĞĢĪŃĞĹÎâĬIJĽĽçĞĢĻĪĿĞĻĪIJĮgÎĪĮĴŁĻĽã
ĞĢĻĪĿĞĻĪIJĮĝgĠĞĢĪŃĞĹÎâÎĦĮĽĦğgÎĪĮĴŁĻĽã
ÎĦĮĽĦĝgĠĞĢĪŃĞĹÎâĞĢĻĪĿĞĻĪIJĮĝgÎĪĮĴŁĻĽã

ﬁ UĴÎĞĻĦ ŃĦĪÍÎĻĽ ĞĮÎ ĠĪĞĽĦĽ
IJĴĻĪİĪŃĦĹgĴĹĦçŁĴÎĞĻĦçĴĞĹĞİĽâã
IJĴĻĪİĪŃĦĹgŁĴÎĞĻĦçĴĞĹĞİĽâÎĦĮĽĦĝã
IJĴĻĪİĪŃĦĹgŁĴÎĞĻĦçĴĞĹĞİĽâÎĦĮĽĦğã
IJĴĻĪİĪŃĦĹgĴIJĽĻçŁĴÎĞĻĦçĴĞĹĞİĽâã
```

```
ﬠ VĞĬĨĤĞĻĦ ĻÎĦ İIJĤĦĬ

ﬠ CĹĦĞĻĦ ĻĦĹĻ ĤĞĻĞĹĦĻ
XçĻĦĹĻd ÑçĻĦĹĻ ł ĹĴĨĹĞĬçĤĞĻĞâĹĞİĴĬĦĹłĝﬁﬁd ĢĬĞĹĹĦĹłġã

ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ IJÌ IJłĹ ĻĦĹĻĨĮÍ ĤĞĻĞ ĻÎĹIJłÍÎ ĻÎĨĹ ĬĞÑĦĹ
ĤĦĮĹĦĝgÌIJĹŃĞĹĤâXçĻĦĹĻã

ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ ĻÎĹIJłÍÎ ĞĢĻĨĹĞĻĨIJĮ ÌłĮĢĻĨIJĮ
ﬠ ĻĞĪĦĹ ĻÎĦ IJłĻĴłĻ IJÌ ÌĨĹĹĻ ĤĦĮĹĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨĹĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĹĦĝgIJłĻĴłĻã

ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ ĻÎĹIJłÍÎ ĹĦĢIJĮĤ DĦĮĹĦ ĬĞÑĦĹ
ﬠ ĻĞĪĦĹ IJłĻĴłĻĹ IJÌ ĞĢĻĨĹĞĻĨIJĮ ÌłĮĢĻĨIJĮ IJÌ ÌĨĹĹĻ ĬĞÑĦĹ ĞĹ ĨĮĴłĻĹ
ĤĦĮĹĦĝgÌIJĹŃĞĹĤâĞĢĻĨĹĞĻĨIJĮĝgIJłĻĴłĻã

ﬠ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ ĻÎĹIJłÍÎ ĻÎĦ ĞĢĻĨĹĞĻĨIJĮwĬIJĹĹ ÌłĮĢĻĨIJĮ
ﬠ ĻĞĪĦĹ ĻÎĦ IJłĻĴłĻ IJÌ ĹĦĢIJĮĤ ĤĦĮĹĦ ĬĞÑĦĹ ÎĦĹĦ ĞĮĤ ĹĦĻłĹĮĹ ĬIJĹĹ
ĬIJĹĹ ł ĬIJĹĹçĞĢĻĨĹĞĻĨIJĮgÌIJĹŃĞĹĤâĤĦĮĹĦĝgIJłĻĴłĻd ÑçĻĦĹĻã

ﬠ CĞĬĢłĬĞĻĦ ĞĢĢłĹĞĢÑ ÌĹIJİ IJłĻĴłĻ IJÌ ĞĢĻĨĹĞĻĨIJĮğ ĞĮĤ ĻĞĹĹĦĻĹ
ﬠ ĢĞĬĢłĬĞĻĦ ĹĞĬłĦĹ ĞĬIJĮÍ ÌĨĹĹĻ ĞÑĨĹ
ĴĹĦĤĨĢĻĨIJĮĹ ł ĮĴgĞĹÍİĞÑâĬIJĹĹçĞĢĻĨĹĞĻĨIJĮgIJłĻĴłĻd ĞÑĨĹłĝã
ĨÌ ĬĦĮâÑçĻĦĹĻgĹĴĞĴĦã łł ğf
    ÑçĻĦĹĻ ł ĮĴgĞĹÍİĞÑâÑçĻĦĹĻd ĞÑĨĹłĝã
ĞĢĢłĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦĤĨĢĻĨIJĮĹłłÑçĻĦĹĻã

ĴĹĨĮĻâÌďĹĞĬĨĤĞĻĨIJĮd ĞĢĢf ăĞĢĢłĹĞĢÑfgġÌåd ĬIJĹĹf ăĬIJĹĹfgġÌåďã


ńńń
ggg
ĦĴIJĢĴf ĝﬁﬁﬁﬁd ĞĢĢf ﬁgîǵìd ĬIJĹĹf ﬁgğĝì âĤĞĻĞçĬIJĹĹf ﬁgĝĥìd ĹĦÍçĬIJĹĹf ﬁgﬁħﬁãd
ĬĹf ﬁgﬁĝîîﬁﬁĥﬁìǵĝġĝíìì ħì
ĹĞĬĨĤĞĻĨIJĮd ĞĢĢf ﬁgíġﬁd ĬIJĹĹf ﬁgǵġĥ
```

Fig 14.01: Training with regularization

Anim 14.01: https://nnfs.io/abc

This animation shows the training data in the background (dimmed dots) and the validation data in the foreground. After adding the L2 regularization term to the hidden layer, we achieved a lower validation loss (0.858 before adding regularization in, 0.435 now) and higher accuracy (0.803 before, 0.830 now). We can also take a moment to exemplify how a simple increase in data for training can make a large difference. If we grow from 100 samples to 1,000 samples:

```
ń CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflflfld ĢĬĞĽĽĦĽłġã
```

And run the code again:

```
ńńń
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgíîĥd ĬIJĽĽf flgġĥì âĤĞĻĞçĬIJĽĽf flgğîġd ĹĦÍçĬIJĽĽf flgflħġãd
ĬĹf flgflĝîîflflĥflìġĝġĝíììħì
ĻĞĬĨĤĞĻĨIJĮd ĞĢĢf flgíìġd ĬIJĽĽf flgġġğ
```

Fig 14.02: Training with regularization and more training data.

Anim 14.02: https://nnfs.io/bcd

We can see that this change alone also had a considerable impact on both validation accuracy in general, as well as the delta between the validation and training accuracies — lower accuracy and higher training loss suggest that the capacity of the model might be too low. A large delta earlier and a small one now suggests that the model was most likely overfitting previously. In theory, this regularization allows us to create much larger models without fear of overfitting (or memorization). We can test this by increasing the number of neurons per layer. Going with 128 or 256 neurons per layer helps with the training accuracy but not that much with the validation accuracy:

```
fi CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮĤ ğĤħ IJŁĻĴŁĻ ĿĞĬŁĦĽ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ğĤħd ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłĤĦhǵd
                                ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłĤĦhǵã
```

```
ﬁ CĹĦĞĻĦ RĦLU ĞĢĻĨĿĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦĤ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĿĞĻĨIJĮĝ ł AĢĻĨĿĞĻĨIJĮçRĦLUâã

ﬁ CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğĥħ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĬĦ IJŁĻĴŁĻ
ﬁ IJÌ ĴĹĦĿĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ġ IJŁĻĴŁĻ ĿĞĬŁĦĽ âIJŁĻĴŁĻ ĿĞĬŁĦĽã
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâğĥħd ġã


ńńń
ĦĴIJĢÎf ĝﬂﬂﬂﬂd ĞĢĢf ﬂgîğﬂd ĬIJĽĽf ﬂggħĝ âĤĞĻĞçĬIJĽĽf ﬂggĝġd ĹĦÍçĬIJĽĽf ﬂgﬂġìãd
ĬĹf ﬂgﬂĝîîﬂﬂĥﬂìġĝġĝíììħì
ĿĞĬĨĤĞĻĨIJĮd ĞĢĢf ﬂgíîġd ĬIJĽĽf ﬂggġğ
```

This didn't produce much of a change in results, but raising this number again to 512 did improve validation accuracy and loss as well:

```
ﬁ CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮĤ ĥĝğ IJŁĻĴŁĻ ĿĞĬŁĦĽ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ĥĝğd ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦhġd
                                ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦhġã

ﬁ CĹĦĞĻĦ RĦLU ĞĢĻĨĿĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦĤ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĿĞĻĨIJĮĝ ł AĢĻĨĿĞĻĨIJĮçRĦLUâã

ﬁ CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ĥĝğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĬĦ IJŁĻĴŁĻ
ﬁ IJÌ ĴĹĦĿĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ġ IJŁĻĴŁĻ ĿĞĬŁĦĽ âIJŁĻĴŁĻ ĿĞĬŁĦĽã
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâĥĝğd ġã


ńńń
ĦĴIJĢÎf ĝﬂﬂﬂﬂd ĞĢĢf ﬂgîĝíd ĬIJĽĽf ﬂggĥġ âĤĞĻĞçĬIJĽĽf ﬂggĝﬂd ĹĦÍçĬIJĽĽf ﬂgﬂġġãd
ĬĹf ﬂgﬂĝîîﬂﬂĥﬂìġĝġĝíììħì
ĿĞĬĨĤĞĻĨIJĮd ĞĢĢf ﬂgîğﬂd ĬIJĽĽf ﬂggĥħ
```

Fig 14.03: Training with regularization and more training data (tuned).

Anim 14.03: https://nnfs.io/cde

In this case, we see that the accuracies and losses for in-sample and out-of-sample data are almost identical. From here, we could add either more layers and neurons or both. Feel free to tinker with this to try to improve it. Next, we're going to cover another regularization method: **dropout**.

Supplementary Material: https://nnfs.io/ch14
Chapter code, further resources, and errata for this chapter.

Chapter 15

Dropout

Another option for neural network regularization is adding a **dropout layer**. This type of layer disables some neurons, while the others pass through unchanged. The idea here, similarly to regularization, is to prevent a neural network from becoming too dependent on any neuron or for any neuron to be relied upon entirely in a specific instance (which can be common if a model overfits the training data). Another problem dropout can help with is **co-adoption**, which happens when neurons depend on the output values of other neurons and do not learn the underlying function on their own. Dropout can also help with **noise** and other perturbations in the training data as more neurons working together mean that the model can learn more complex functions.

The Dropout function works by randomly disabling neurons at a given rate during every forward pass, forcing the network to learn how to make accurate predictions with only a random part of neurons remaining. Dropout forces the model to use more neurons for the same purpose, resulting in a higher chance of learning the underlying function that describes the data. For example, if we disable one half of the neurons during the current step, and the other half during the next step, we are forcing more neurons to learn the data, as only a part of them “sees” the data and gets updates in a given pass. These alternating halves of neurons are an example, and in reality, we’ll use a hyperparameter to inform the dropout layer of the number of neurons to disable randomly.

Also, since active neurons are changing, dropout helps prevent overfitting, as the model can't use specific neurons to memorize certain samples. It's also worth mentioning that the dropout layer does not truly disable neurons, but instead zeroes their outputs. In other words, dropout does not decrease the number of neurons used, nor does it make the training process twice as fast when half the neurons are disabled.

Forward Pass

In the code, we will "turn off" neurons with a filter that is an array with the same shape as the layer output but filled with numbers drawn from a Bernoulli distribution. A **Bernoulli distribution** is a binary (or discrete) probability distribution where we can get a value of *1* with a probability of p and value of *0* with a probability of q. Let's take some random value from this distribution, r_i, then:

$$P(r_i = 1) = p$$

$$P(r_i = 0) = q = 1 - p = 1 - P(r_i = 1)$$

What this means is that the probability of this value being *1* is p. The probability of it being *0* is $q = 1 - p$, therefore:

$$r_i \sim Bernoulli(p)$$

This means that the given r_i is an equivalent of a value from the Bernoulli distribution with a probability p for this value to be *1*. If r_i is a single value from this distribution, a draw from this distribution, reshaped to match the shape of the layer outputs, can be used as a mask to these outputs.

We are returned an array filled with values of *1* with a probability of p and otherwise values of *0*. We then apply this filter to the output of a layer we want to add dropout to.

Fig 15.01: Example model with no dropout applied.

Fig 15.02: Example model with 0.5 dropout.

Fig 15.03: Example model with 0.9 dropout.

Anim 15.01-15.03: https://nnfs.io/def

With the code, we have one hyperparameter for a dropout layer. This is a value for the percentage of neurons to disable in that layer. For example, if you chose 0.10 for the dropout parameter, 10% of the neurons will be disabled at random during each forward pass. Before we use NumPy, we'll demonstrate this with an example in pure Python:

```
ĨİĴIJĹĻ ĹĞĮĤIJİ

ĤĹIJĴIJŁĻçĹĞĻĦ ł flgĥ
ſi EŃĞİĴĬĦ IJŁĻĴŁĻ ĢIJĮĻĞĨĮĨĮĺ ĝſl ŁĠĬŁĦĽ
ĦŃĞİĴĬĦçIJŁĻĴŁĻ ł äflgğìd hĝgflġd flgħìd flgîîd flgflĥd
                hflgġìd hğgflĝd ĝgĝġd hflgflìd flgìġā
```

```
ƒ RĦĴĦĞĻ ĞĽ ĬIJĮÍ ĞĽ ĮĦĢĦĽĽĞĹÑ
ŃÎĨĬĦ TĹŁĦf

    ƒ RĞĮĤIJİĬÑ ĢÎIJIJĽĦ ĨĮĤĦŃ ĞĮĤ ĽĦĻ ŁĞĬŁĦ ĻIJ ƒl
    ĨĮĤĦŃ ł ĹĞĮĤIJİgĹĞĮĤĨĮĻâƒld ĬĦĮâĦŃĞİĴĬĦçIJŁĻĴŁĻã h ĝã
    ĦŃĞİĴĬĦçIJŁĻĴŁĻäĨĮĤĦŃã ł ƒl

    ƒ WĦ İĨÍÎĻ ĽĦĻ ĞĮ ĨĮĤĦŃ ĻÎĞĻ ĞĬĹĦĞĤÑ ĨĽ ĢĦĹIJĦĤ
    ƒ TÎĦĹĦ ĞĹĦ ĤĨÌÌĦĹĦĮĻ ŃĞÑĽ IJÌ IJŁĦĹĢIJİĨĮÍ ĻÎĨĽ ĴĹIJĠĬĦİd
    ƒ ÌIJĹ ĽĨİĴĬĨĢĨĻÑ ŃĦ ĢIJŁĮĻ ŁĞĬŁĦĽ ĻÎĞĻ ĞĹĦ ĦŃĞĢĻĬÑ ƒl
    ƒ ŃÎĨĬĦ ĨĻďĽ ĦŃĻĹĦİĦĬÑ ĹĞĹĦ ĨĮ ĹĦĞĬ İIJĤĦĬ ĻÎĞĻ ŃĦĨÍÎĻĽ
    ƒ ĞĹĦ ĦŃĞĢĻĬÑ ƒld ĻÎĨĽ ĨĽ ĮIJĻ ĻÎĦ ĠĦĽĻ İĦĻÎIJĤ ÌIJĹ ĽŁĹĦ
    ĤĹIJĴĴĦĤçIJŁĻ ł ƒl
    ÌIJĹ ŁĞĬŁĦ ĨĮ ĦŃĞİĴĬĦçIJŁĻĴŁĻf
        ĨÌ ŁĞĬŁĦ łł ƒlf
            ĤĹIJĴĴĦĤçIJŁĻ ĸł ĝ

    ƒ IÌ ĹĦĶŁĨĹĦĤ ĮŁİĠĦĹ IJÌ IJŁĻĴŁĻĽ ĨĽ ĢĦĹIJĦĤ h ĬĦĞŁĦ ĻÎĦ ĬIJIJĴ
    ĨÌ ĤĹIJĴĴĦĤçIJŁĻ w ĬĦĮâĦŃĞİĴĬĦçIJŁĻĴŁĻã ńł ĤĹIJĴIJŁĻçĹĞĻĦf
        ĠĹĦĞĨ

ĴĹĨĮĻâĦŃĞİĴĬĦçIJŁĻĴŁĻã


ńńń
äƒld hĝgƒlġd ƒlgħìd ƒlgîîd ƒld hƒlggìd ƒld ƒld ƒld ƒlgìġã
```

The code is relatively rudimentary, but the idea is to keep zeroing neuron outputs (setting them to 0) randomly until we've disabled whatever target % of neurons we require. If we consider a Bernoulli distribution as a special case of a Binomial distribution with *n=1* and look at a list of available methods in NumPy, it turns out that there's a much cleaner way to do this using *numpy.random.binomial*. A binomial distribution differs from Bernoulli distribution in one way, as it adds a parameter, *n,* which is the number of concurrent experiments (instead of just one) and returns the number of successes from these *n* experiments.

`ĮĴgĹĞĮĤIJİgĠĨĮIJİĨĞĬâã` works by taking the already discussed parameters *n* (number of experiments) and *p* (probability of the true value of the experiment) as well as an additional parameter *size*: `ĮĴgĹĞĮĤIJİgĠĨĮIJİĨĞĬâĮd Ĵd ĽĨĢĦã`.

The function itself can be thought of like a coin toss, where the result will be 0 or 1. The *n* is how many tosses of the coin do you want to do. The *p* is the probability for the toss result to be a 1. The overall result is a sum of all toss results. The *size* is how many of these "tests" to run, and the return is a list of overall results. For example:

```
ĮĴgĹĞĮĤIJİgĠĨĮIJİĨĞĬâğd flgĥd ĹĨŅĦłĝflã
```

This will produce an array that is of size 10, where each element will be the sum of 2 coin tosses, where the probability of 1 will be 0.5, or 50%. The resulting array:

```
ĞĹĹĞÑâäfld fld ĝd ğd fld ğd fld ĝd fld ğāã
```

We can use this to create our dropout layer. Our goal here is to create a filter where the intended dropout % is represented as 0, with everything else as 1. For example, let's say we have a dropout layer that we'll add after a layer that consists of 5 neurons, and we wish to have a 20% dropout. An example of a dropout layer might look like:

```
äĝd fld ĝd ĝd ĝā
```

As you can see, ⅕ of that list is a 0. This is an example of the filter we're going to apply to the output of the dense layer. If we multiplied a neural network's layer output by this, we'd be effectively disabling the neuron at the same index as the 0.
We can mimic that with `ĮĴgĹĞĮĤIJİgĠĨĮIJİĨĞĬâã` by doing:

```
ĤĹIJĴIJŁĻçĹĞĻĦ ł flgğfl
ĮĴgĹĞĮĤIJİgĠĨĮIJİĨĞĬâĝd ĝhĤĹIJĴIJŁĻçĹĞĻĦd ĹĨŅĦłĥã

ńńń
ĞĹĹĞÑâäfld ĝd ĝd ĝd ĝāã
```

This is based on probabilities, so there will be times when it does not look like the above array. There could be times no neurons zero out, or all neurons zero out. On average, these random draws will tend toward the probability we desire. Also, this was an example using a very small layer (5 neurons). On a realistically sized layer, you should find the probability more consistently matches your intended value.

Assume a neural network layer's output is:

```
ĦŇĞİĴĬĦçIJŁĻĴŁĻ ł ĮĴgĞĹĹĞÑâäflgğìd hĝgflġd flgħìd flgîîd flgflĥd
                                hflgġìd hğgflĝd ĝgĝġd hflgflìd flgìġāã
```

Next, let's assume our target dropout rate is 0.3, or 30%. We apply a dropout layer:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ĤĹIJĴIJŁĻçĹĞĻĦ ł flgġ
ĦŇĞİĴĬĦçIJŁĻĴŁĻ ł ĮĴgĞĹĹĞÑâäflgğìd hĝgflġd flgħìd flgîîd flgflĥd
                                hflgġìd hğgflĝd ĝgĝġd hflgflìd flgìġāã

ĦŇĞİĴĬĦçIJŁĻĴŁĻ ǻł ĮĴgĹĞĮĤIJİgĠĨĮIJİĨĞĬâĝd ĝhĤĹIJĴIJŁĻçĹĞĻĦd
                                        ĦŇĞİĴĬĦçIJŁĻĴŁĻgĽĨĞĴĦã

ĴĹĨĮĻâĦŇĞİĴĬĦçIJŁĻĴŁĻã


ńńń
ä flgğì hĝgflġ  flgflfl  flgîî  flg   hflgġì hğgflĝ  ĝgĝġ hflgflì  flg  ā
```

Note that our dropout rate is the ratio of neurons we intend to *disable (q)*. Sometimes, the implementation of dropout will include a rate parameter that instead means the fraction of neurons you intend to *keep (p)*. At the time of writing this, the dropout parameter in deep learning frameworks, TensorFlow and Keras, represents the neurons you intend to disable. On the other hand, the dropout parameter in PyTorch and the original paper on dropout (http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf) signal the ratio of neurons you intend to keep.

The way it's implemented is not important. What *is* important is that you know which method you're using!

While dropout helps a neural network generalize and is helpful for training, it's not something we want to utilize when predicting. It's not as simple as only omitting it because the magnitude of inputs to the next neurons can be dramatically different. If you have a dropout of 50%, for example, this would suggest that, on average, your inputs to the next layer neurons will be 50% smaller when summed, assuming they are fully-connected. What that means is that we used dropout during training, and, in this example, a random 50% of neurons output a value of 0 at each of the steps. Neurons in the next layer multiply inputs by weights, sum them, and receive values of 0 for half of their inputs. If we don't use dropout during prediction, all neurons will output their values, and this state won't match the state seen during training, since the sums will be statistically about twice as big. To handle this, during prediction, we might multiply all of the outputs by the dropout fraction, but that'd add another step for the forward pass, and there is a better way to achieve this. Instead, we want to scale the data back up after a dropout, during the training phase, to mimic the mean of the sum when all of the neurons output their values. *Example_output* becomes:

```
ĦŇĞİĴĬĦçIJŁĻĴŁĻ ǻł ĮĴgĹĞĮĤIJİgĠĨĮIJİĨĞĬâĝd ĝhĤĹIJĴIJŁĻçĹĞĻĦd
                                        ĦŇĞİĴĬĦçIJŁĻĴŁĻgĽĨĞĴĦã w ċ
                  âĝhĤĹIJĴIJŁĻçĹĞĻĦã
```

Notice that we added the division of the dropout's result by the dropout rate. Since this rate is a fraction, it makes the resulting values larger, accounting for the value lost because a fraction of the neuron outputs being zeroed out. This way, we don't have to worry about the prediction and can simply omit the dropout during prediction. In any specific example, you will find that scaling doesn't equal the same sum as before because we're randomly dropping neurons. That said, after enough samples, the scaling will average out overall. To prove this:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ĤĹIJĴIJŁĻçĹĞĻĦ ł flgğ
ĦŇĞİĴĬĦçIJŁĻĴŁĻ ł ĮĴgĞĹĹĞÑâäflgğìd hĝgflġd flgħìd flgîîd flgflĥd
                              hflgġìd hğgflĝd ĝgĝġd hflgflìd flgìġãã
ĴĹĨĮĻâĬďĽŁİ ĨĮĨĻĨĞĬ ăĽŁİâĦŇĞİĴĬĦçIJŁĻĴŁĻãåďã

ĽŁİĽ ł äã
ÌIJĹ Ĩ ĨĮ ĹĞĮÍĦâĝflflflflãf

    ĦŇĞİĴĬĦçIJŁĻĴŁĻğ ł ĦŇĞİĴĬĦçIJŁĻĴŁĻ ấ ċ
        ĮĴgĹĞĮĤIJİgĠĨĮIJİĨĞĬâĝd ĝhĤĹIJĴIJŁĻçĹĞĻĦd ĦŇĞİĴĬĦçIJŁĻĴŁĻgĽĨĞĴĦã w ċ
        âĝhĤĹIJĴIJŁĻçĹĞĻĦã
    ĽŁİĽgĞĴĴĦĮĤâĽŁİâĦŇĞİĴĬĦçIJŁĻĴŁĻğãã

ĴĹĨĮĻâĬďİĦĞĮ ĽŁİf ăĮĴgİĦĞĮâĽŁİĽãåďã


ńńń
ĽŁİ ĨĮĨĻĨĞĬ flgġħflflflflflflflflflflflflflĝĥ
İĦĞĮ ĽŁİf flgġħğíğflflflflflflflflflĝġ
```

It's not exact yet, but you should get the idea.

Backward Pass

The last missing piece to implement dropout as a layer is a backward pass method. As before, we need to calculate the partial derivative of the dropout operation:

When the value of element r_i equals *1*, its function and derivative becomes the neuron's output, *z*, compensated for the loss value by *1-q*, where *q* is the dropout rate, as we just described:

$$f(z,q) = \frac{z}{1-q} \quad \rightarrow \quad \frac{\partial}{\partial z}[\frac{z}{1-q}] = \frac{1}{1-q} \cdot \frac{\partial}{\partial z} z = \frac{1}{1-q} \cdot 1 = \frac{1}{1-q}$$

That's because the derivative with respect to *z* of *z* is 1, and we treat the rest as a constant. Whenr_i*=0*:

$$f(z,q) = 0 \quad \rightarrow \quad \frac{\partial}{\partial z} 0 = 0$$

And that's because we are zeroing this element of the dropout filter, and the derivative of any constant value (including *0*) is *0*. Let's combine both cases and denote *Dropout* as *Dr*:

$$Dr_i = \begin{cases} \frac{z_i}{1-q} & r_i = 1 \\ 0 & r_i = 0 \end{cases} \quad \rightarrow \quad \frac{\partial}{\partial z_i} Dr_i = \begin{cases} \frac{1}{1-q} & r_i = 1 \\ 0 & r_i = 0 \end{cases} = \frac{r_i}{1-q}$$

i denotes the index of the given input (and the layer output). When we write a derivative of the dropout function this way, we can simplify it to a value from the Bernoulli distribution divided by *1-q*, which is identical to our scaled mask, the function the dropout applies during the forward pass, as it's also either *1* divided by *1-q*, or *0*. Thus, we can save this mask during the forward pass and use it with the chain rule as the gradient of this function.

The Code

We can now implement this concept in a new layer type, the dropout layer:

```
class DropoutLayer:
    def __init__(self, rate):
        self.rate = rate

    def forward(self, inputs):
        self.inputs = inputs
        self.binary_mask = (
            np.random.binomial(1, self.rate,
                               size=inputs.shape)
            / self.rate
        )
        self.output = inputs * self.binary_mask

    def backward(self, dvalues):
        self.dinputs = dvalues * self.binary_mask

    def get_parameters(self):
        return self.rate, self.binary_mask, self.inputs, self.output

    def set_parameters(self, rate, binary_mask, inputs, output):
        self.rate = rate
        self.binary_mask = binary_mask
        self.inputs = inputs
        self.output = output
```

Let’s take this new dropout layer, and add it between our two dense layers. First defining it:

```
# Create dropout layer
dropout1 = DropoutLayer(0.1)

# Create second Dense layer with 64 input features (as we take output
# of previous layer here) and 3 output values (output values)
dense2 = DenseLayer(64, 3)

# Create activation function object
activation2 = ActivationSoftmax()

# Create loss function object
loss_function = LossCategoricalCrossEntropy()

# Create optimizer object
optimizer = OptimizerAdam(learning_rate=0.05, decay=5e-5)

# Initialize variables for loss and accuracy
loss = 0
acc = 0

# Train in loss_function.remember_trainable_layers = True
for epoch in range(10001):
    # Forward pass through layer 1
    dense1.forward(X)

    # Forward pass through activation function
    activation1.forward(dense1.output)

    # Forward pass through dropout layer
    dropout1.forward(activation1.output)

    # Forward pass through layer 2
    dense2.forward(dropout1.output)

    # Forward pass through activation function
    activation2.forward(dense2.output)

    # Calculate loss
    loss = loss_function.calculate(activation2.output, y)

    # Calculate accuracy
    predictions = np.argmax(activation2.output, axis=1)
    if len(y.shape) == 2:
        y = np.argmax(y, axis=1)
    acc = np.mean(predictions == y)

    # Backward pass
    loss_function.backward(activation2.output, y)
    activation2.backward(loss_function.dinputs)
    dense2.backward(activation2.dinputs)
    dropout1.backward(dense2.dinputs)
    activation1.backward(dropout1.dinputs)
    dense1.backward(activation1.dinputs)

    # Update weights
    optimizer.pre_update_parameters()
    optimizer.update_parameters(dense1)
    optimizer.update_parameters(dense2)
    optimizer.post_update_parameters()

    # Print output every 100 epochs
    if epoch % 100 == 0:
        print f'epoch: {epoch}, acc: {acc:.3f}, loss: {loss:.3f}'
```

```
ſı CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJŁĻĴŁĻ
ſı IJÌ ĴĹĦĹĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ġ IJŁĻĴŁĻ ĹĞĬŁĦĽ âIJŁĻĴŁĻ ĹĞĬŁĦĽã
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâħǵd ġã
```

During the forward pass, add in the dropout:

```
ſı PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻÎĹIJŁÍÎ DĹIJĴIJŁĻ ĬĞÑĦĹ
ĤĹIJĴIJŁĻĝgÌIJĹŃĞĹĤâĞĢĻĨĹĞĻĨIJĮĝgIJŁĻĴŁĻã

ſı PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ
ſı ĻĞĨĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĹĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
ĤĦĮĽĦğgÌIJĹŃĞĹĤâĤĹIJĴIJŁĻĝgIJŁĻĴŁĻã
```

And of course in the backward pass:

```
ĤĹIJĴIJŁĻĝgĠĞĢĨŃĞĹĤâĤĦĮĽĦğgĤĨĮĴŁĻĽã
ĞĢĻĨĹĞĻĨIJĮĝgĠĞĢĨŃĞĹĤâĤĹIJĴIJŁĻĝgĤĨĮĴŁĻĽã
```

Let’s also raise the learning rate a bit, from 0.02 to 0.05 and raise the learning rate decaying from 5e-7 to 5e-5 as these parameters work better with our model and dropout layer.

Full code up to now:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĨİĴIJĹĻ ĽĴĨĹĞĬçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


ſı DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ſı LĞÑĦĹ ĨĮĨĻĨĞĬĨÑĞĻĨIJĮ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽd
                ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝłfld ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłfld
                ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝłfld ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłflãf
        ſı IĮĨĻĨĞĬĨŅĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍÎĻĽ ł flgflĝ á ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã
```

```
    # SHL LHILIGLINGLIIJI LLLHIILI
    LHIIgNHIIILCLHILIGLINHLCIg = NHIIILCLHILIGLINHLCIg
    LHIIgNHIIILCLHILIGLINHLCIg = NHIIILCLHILIGLINHLCIg
    LHIIgGIGLCLHILIGLINHLCIg = GIGLCLHILIGLINHLCIg
    LHIIgGIGLCLHILIGLINHLCIg = GIGLCLHILIGLINHLCIg

# FIJLNGLA JGLL
AHI IIJLNGLAaLHIId IIJJLLLaf
    # RHIHIGHL IIJJLL LGILHL
    LHIIgIIJJLLL = IIJJLLL
    # CGIGLIGLH IJLLJLL LGILHL ILIJI IIJJLLLd NHIIILLL GIA GIGLHL
    LHIIgIJLLJLL = IJgAIJLaIIJJLLLd LHIIgNHIIILLLa + LHIIgGIGLHL

# BGGINGLA JGLL
AHI GGGINGLAaLHIId ALGILHLa f
    # GLGAIHIJLL IJJ JGLGIHLHLL
    LHIIgANHIIILLL = IJgAIJLaLHIIgIIJJLLLgTd ALGILHLa
    LHIIgAGIGLHL = IJgLLIaALGILHLd GNILLlfd IHHJAIILLlTLLHa

    # GLGAIHIJLL IJJ LHILIGLINGLIIJJ
    # Lg IJJ NHIIILL
    II LHIIgNHIIILCLHILIGLINHLCIg n f
        ALg = IJgIJJHLCIIIHaLHIIgNHIIILLLa
        ALgaLHIIgNHIIILL l· fa = hg
        LHIIgANHIIILL +l LHIIgNHIIILCLHILIGLINHLCIg f ALg
    # Lg IJJ NHIIILL
    II LHIIgNHIIILCLHILIGLINHLCIg n f
        LHIIgANHIIILL +l g f LHIIgNHIIILCLHILIGLINHLCIg f c
                        LHIIgNHIIILL
    # Lg IJJ GIGLHL
    II LHIIgGIGLCLHILIGLINHLCIg n f
        ALg = IJgIJJHLCIIIHaLHIIgGIGLHLa
        ALgaLHIIgGIGLHL l· fa = hg
        LHIIgAGIGLHL +l LHIIgGIGLCLHILIGLINHLCIg f ALg
    # Lg IJJ GIGLHL
    II LHIIgGIGLCLHILIGLINHLCIg n f
        LHIIgAGIGLHL +l g f LHIIgGIGLCLHILIGLINHLCIg f c
                        LHIIgGIGLHL

    # GLGAIHIJLL IJJ LGILHL
    LHIIgAIIJJLLL = IJgAIJLaALGILHLd LHIIgNHIIILLLgTa
```

```
ƒ DĹIJĴIJŁĻ
ĢĬĞĽĹ LĞÑĦĹçDĹIJĴIJŁĻf

    ƒ IĮĨĻ
    ÎĦÌ ççĨĮĨĻççâĽĦĬÌd ĹĞĻĦãf
        ƒ SĻIJĹĦ ĹĞĻĦd ŃĦ ĨĮŁĦĹĻ ĨĻ ĞĽ ÌIJĹ ĦŃĞİĴĬĦ ÌIJĹ ÎĹIJĴIJŁĻ
        ƒ IJÌ flgĝ ŃĦ ĮĦĦÎ ĽŁĢĢĦĽĽ ĹĞĻĦ IJÌ flgî
        ĽĦĬÌgĹĞĻĦ ł ĝ h ĹĞĻĦ

    ƒ FIJĹŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ÌIJĹŃĞĹÎâĽĦĬÌd ĨĮĴŁĻĽãf
        ƒ SĞŁĦ ĨĮĴŁĻ ŁĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ƒ GĦĮĦĹĞĻĦ ĞĮÎ ĽĞŁĦ ĽĢĞĬĦÎ İĞĽĪ
        ĽĦĬÌgĠĨĮĞĹÑçİĞĽĪ ł ĮĴgĹĞĮÎIJİgĠĨĮIJİĨĞĬâĝd ĽĦĬÌgĹĞĻĦd
                            ĽĨĢĦłĨĮĴŁĻĽgĽĨĞĴĦã w ĽĦĬÌgĹĞĻĦ
        ƒ AĴĴĬÑ İĞĽĪ ĻIJ IJŁĻĴŁĻ ŁĞĬŁĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽ ǻ ĽĦĬÌgĠĨĮĞĹÑçİĞĽĪ

    ƒ BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ĠĞĢĪŃĞĹÎâĽĦĬÌd ÎŁĞĬŁĦĽãf
        ƒ GĹĞÎĨĦĮĻ IJĮ ŁĞĬŁĦĽ
        ĽĦĬÌgÎĨĮĴŁĻĽ ł ÎŁĞĬŁĦĽ ǻ ĽĦĬÌgĠĨĮĞĹÑçİĞĽĪ


ƒ RĦLU ĞĢĻĨŁĞĻĨIJĮ
ĢĬĞĽĹ AĢĻĨŁĞĻĨIJĮçRĦLUf

    ƒ FIJĹŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ÌIJĹŃĞĹÎâĽĦĬÌd ĨĮĴŁĻĽãf
        ƒ RĦİĦİĠĦĹ ĨĮĴŁĻ ŁĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ƒ CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ŁĞĬŁĦĽ ÌĹIJİ ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgİĞÑĨİŁİâfld ĨĮĴŁĻĽã

    ƒ BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ĠĞĢĪŃĞĹÎâĽĦĬÌd ÎŁĞĬŁĦĽãf
        ƒ SĨĮĢĦ ŃĦ ĮĦĦÎ ĻIJ İIJÎĨÌÑ IJĹĨÍĨĮĞĬ ŁĞĹĨĞĠĬĦd
        ƒ ĬĦĻďĽ İĞĪĦ Ğ ĢIJĴÑ IJÌ ŁĞĬŁĦĽ ÌĨĹĽĻ
        ĽĦĬÌgÎĨĮĴŁĻĽ ł ÎŁĞĬŁĦĽgĢIJĴÑâã

        ƒ ZĦĹIJ ÍĹĞÎĨĦĮĻ ŃÎĦĹĦ ĨĮĴŁĻ ŁĞĬŁĦĽ ŃĦĹĦ ĮĦÍĞĻĨŁĦ
        ĽĦĬÌgÎĨĮĴŁĻĽäĽĦĬÌgĨĮĴŁĻĽ l·ł flã ł fl
```

```
# SIJÌĻİĞŇ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇf

    # FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ĨĮĴŁĻĽãf
        # RĦİĦİĠĦĹ ĨĮĴŁĻ ĿĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ

        # GĦĻ ŁĮĮIJĹİĞĬĨĨŅĦÂ ĴĹIJĠĞĠĨĬĨĻĨĦĽ
        ĦŇĴçĿĞĬŁĦĽ ł ĮĴgĦŇĴâĨĮĴŁĻĽ h ĮĴgİĞŇâĨĮĴŁĻĽd ĞŇĨĽłĝd
                                                ĪĦĦĴÂĨİĽłTĹŁĦãã
        # NIJĹİĞĬĨŅĦ ĻÎĦİ ÌIJĹ ĦĞĢÎ ĽĞİĴĬĦ
        ĴĹIJĠĞĠĨĬĨĻĨĦĽ ł ĦŇĴçĿĞĬŁĦĽ w ĮĴgĽŁİâĦŇĴçĿĞĬŁĦĽd ĞŇĨĽłĝd
                                                ĪĦĦĴÂĨİĽłTĹŁĦã

        ĽĦĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĨĬĨĻĨĦĽ

    # BĞĢĨŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ĠĞĢĨŃĞĹÂâĽĦĬÌd ÂĿĞĬŁĦĽãf

        # CĹĦĞĻĦ ŁĮĨĮĨĻĨĞĬĨŅĦÂ ĞĹĹĞÑ
        ĽĦĬÌgÂĨĮĴŁĻĽ ł ĮĴgĦİĴĻÑçĬĨĨĦâÂĿĞĬŁĦĽã

        # EĮŁİĦĹĞĻĦ IJŁĻĴŁĻĽ ĞĮÂ ÍĹĞÂĨĦĮĻĽ
        ÌIJĹ ĨĮÂĦĦŇd âĽĨĮÍĬĦçIJŁĻĴŁĻd ĽĨĮÍĬĦçÂĿĞĬŁĦĽã ĨĮ ċ
                ĦĮŁİĦĹĞĻĦâŅĨĴâĽĦĬÌgIJŁĻĴŁĻd ÂĿĞĬŁĦĽãã f
            # FĬĞĻĻĦĮ IJŁĻĴŁĻ ĞĹĹĞÑ
            ĽĨĮÍĬĦçIJŁĻĴŁĻ ł ĽĨĮÍĬĦçIJŁĻĴŁĻgĹĦĽÎĞĴĦâhĝd ĝã
            # CĞĬĢŁĬĞĻĦ JĞĢIJĠĨĞĮ İĞĻĹĨŇ IJÌ ĻÎĦ IJŁĻĴŁĻ ĞĮÂ
            ÏĞĢIJĠĨĞĮçİĞĻĹĨŇ ł ĮĴgÂĨĞĹÌĬĞĻâĽĨĮÍĬĦçIJŁĻĴŁĻã h ċ
                                ĮĴgÂIJĻâĽĨĮÍĬĦçIJŁĻĴŁĻd ĽĨĮÍĬĦçIJŁĻĴŁĻgTã
            # CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦhŇĨĽĦ ÍĹĞÂĨĦĮĻ
            # ĞĮÂ ĞÂÂ ĨĻ ĻIJ ĻÎĦ ĞĹĹĞÑ IJÌ ĽĞİĴĬĦ ÍĹĞÂĨĦĮĻĽ
            ĽĦĬÌgÂĨĮĴŁĻĽäĨĮÂĦŇã ł ĮĴgÂIJĻâÏĞĢIJĠĨĞĮçİĞĻĹĨŇd
                                        ĽĨĮÍĬĦçÂĿĞĬŁĦĽã


# SGD IJĴĻĨİĨŅĦĹ
ĢĬĞĽĽ OĴĻĨİĨŅĦĹçSGDf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮÍĽd
    # ĬĦĞĹĮĨĮÍ ĹĞĻĦ IJÌ ĝg ĨĽ ÂĦÌĞŁĬĻ ÌIJĹ ĻÎĨĽ IJĴĻĨİĨŅĦĹ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ÂĦĢĞÑłflgd İIJİĦĮĻŁİłflgãf
        ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgÂĦĢĞÑ ł ÂĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgİIJİĦĮĻŁİ ł İIJİĦĮĻŁİ
```

```
# CĞĬĬ IJĮĢH ĠHÌIJĹH ĞĮÑ ĴĞĹĞİHĻHĹ ŁĴĤĞĻHĹ
ĤHÌ ĴĹHçŁĴĤĞĻHçĴĞĹĞİĹâĹHĬÌãf
    ĨÌ ĹHĬÌgĤHĢĞÑf
        ĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ł ĹHĬÌgĬHĞĹĮĨĮÍçĹĞĻH ő ċ
            âĝg w âĝg κ ĹHĬÌgĤHĢĞÑ ő ĹHĬÌgĨĻHĹĞĻĨIJĮĹãã

# UĴĤĞĻH ĴĞĹĞİHĻHĹĹ
ĤHÌ ŁĴĤĞĻHçĴĞĹĞİĹâĹHĬÌd ĬĞÑHĹãf

    # IÌ ŃH ŁĹH İIJİHĮĻŁİ
    ĨÌ ĹHĬÌgİIJİHĮĻŁİf

        # IÌ ĬĞÑHĹ ĤIJHĹ ĮIJĻ ĢIJĮĻĞĨĮ İIJİHĮĻŁİ ĞĹĹĞÑĹd ĢĹHĞĻH ĻĤHİ
        # ÌĨĬĬHĤ ŃĨĻĤ ŅHĹIJĹ
        ĨÌ ĮIJĻ ĤĞĹĞĻĻĹâĬĞÑHĹd ďŃHĨÍĤĻçİIJİHĮĻŁİĹďãf
            ĬĞÑHĹgŃHĨÍĤĻçİIJİHĮĻŁİĹ ł ĮĴgŅHĹIJĹçĬĨĨHâĬĞÑHĹgŃHĨÍĤĻĹã
            # IÌ ĻĤHĹH ĨĹ ĮIJ İIJİHĮĻŁİ ĞĹĹĞÑ ÌIJĹ ŃHĨÍĤĻĹ
            # TĤH ĞĹĹĞÑ ĤIJHĹĮďĻ HŃĨĹĻ ÌIJĹ ĠĨĞĹHĹ ÑHĻ HĨĻĤHĹg
            ĬĞÑHĹgĠĨĞĹçİIJİHĮĻŁİĹ ł ĮĴgŅHĹIJĹçĬĨĨHâĬĞÑHĹgĠĨĞĹHĹã

        # BŁĨĬĤ ŃHĨÍĤĻ ŁĴĤĞĻHĹ ŃĨĻĤ İIJİHĮĻŁİ h ĻĞĨH ĴĹHŁĨIJŁĹ
        # ŁĴĤĞĻHĹ İŁĬĻĨĴĬĨHĤ ĠÑ ĹHĻĞĨĮ ÌĞĢĻIJĹ ĞĮĤ ŁĴĤĞĻH ŃĨĻĤ
        # ĢŁĹĹHĮĻ ÍĹĞĤĨHĮĻĹ
        ŃHĨÍĤĻçŁĴĤĞĻHĹ ł ċ
            ĹHĬÌgİIJİHĮĻŁİ ő ĬĞÑHĹgŃHĨÍĤĻçİIJİHĮĻŁİĹ h ċ
            ĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ő ĬĞÑHĹgĤŃHĨÍĤĻĹ
        ĬĞÑHĹgŃHĨÍĤĻçİIJİHĮĻŁİĹ ł ŃHĨÍĤĻçŁĴĤĞĻHĹ

        # BŁĨĬĤ ĠĨĞĹ ŁĴĤĞĻHĹ
        ĠĨĞĹçŁĴĤĞĻHĹ ł ċ
            ĹHĬÌgİIJİHĮĻŁİ ő ĬĞÑHĹgĠĨĞĹçİIJİHĮĻŁİĹ h ċ
            ĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ő ĬĞÑHĹgĤĠĨĞĹHĹ
        ĬĞÑHĹgĠĨĞĹçİIJİHĮĻŁİĹ ł ĠĨĞĹçŁĴĤĞĻHĹ

    # VĞĮĨĬĬĞ SGD ŁĴĤĞĻHĹ âĞĹ ĠHÌIJĹH İIJİHĮĻŁİ ŁĴĤĞĻHã
    HĬĹHf
        ŃHĨÍĤĻçŁĴĤĞĻHĹ ł hĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ő ċ
                          ĬĞÑHĹgĤŃHĨÍĤĻĹ
        ĠĨĞĹçŁĴĤĞĻHĹ ł hĹHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ő ċ
                        ĬĞÑHĹgĤĠĨĞĹHĹ

    # UĴĤĞĻH ŃHĨÍĤĻĹ ĞĮĤ ĠĨĞĹHĹ ŁĹĨĮÍ HĨĻĤHĹ
    # ŁĞĮĨĬĬĞ IJĹ İIJİHĮĻŁİ ŁĴĤĞĻHĹ
    ĬĞÑHĹgŃHĨÍĤĻĹ κł ŃHĨÍĤĻçŁĴĤĞĻHĹ
    ĬĞÑHĹgĠĨĞĹHĹ κł ĠĨĞĹçŁĴĤĞĻHĹ

# CĞĬĬ IJĮĢH ĞÌĻHĹ ĞĮÑ ĴĞĹĞİHĻHĹ ŁĴĤĞĻHĹ
ĤHÌ ĴIJĹĻçŁĴĤĞĻHçĴĞĹĞİĹâĹHĬÌãf
    ĹHĬÌgĨĻHĹĞĻĨIJĮĹ κł ĝ
```

```
fi AĤĞÍĹĞĤ IJĴĻĨIĨĨĨŅĦĹ
ĢĬĞĹĹ OĴĻĨIĨĨŅĦĹçAĤĞÍĹĞĤf

    fi IĮĨĻĨĞĬĨŅĦ IJĴĻĨIĨĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮÍĹ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ĤĦĢĞÑłflgd ĦĴĹĨĬIJĮłĝĦhìãf
        ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgĦĴĹĨĬIJĮ ł ĦĴĹĨĬIJĮ

    fi CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴĤĞĻĦĹ
    ĤĦÌ ĴĹĦçłĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgĤĦĢĞÑf
            ĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ấ ċ
                âĝg w âĝg κ ĹĦĬÌgĤĦĢĞÑ ấ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    fi UĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĹ
    ĤĦÌ łĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌd ĬĞÑĦĹãf

        fi IÌ ĬĞÑĦĹ ĤIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĨĦ ĞĹĹĞÑĹd
        fi ĢĹĦĞĻĦ ĻĨĦĬ ÌĨĨĬĦĤ ŃĨĻĨ ŅĦĹIJĹ
        ĨÌ ĮIJĻ ĨĞĹĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍĨĻçĢĞĢĨĦďãf
            ĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgŃĦĨÍĨĻĹã
            ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã

        fi UĴĤĞĻĦ ĢĞĢĨĦ ŃĨĻĨ ĹĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ÍĹĞĤĨĦĮĻĹ
        ĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦ κł ĬĞÑĦĹgĤŃĦĨÍĨĻĹááğ
        ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ κł ĬĞÑĦĹgĤĠĨĞĹĦĹááğ

        fi VĞĮĨĬĬĞ SGD ĴĞĹĞĬĦĻĦĹ łĴĤĞĻĦ κ IJĹĬĞĬĨŅĞĻĨIJĮ
        fi ŃĨĻĨ ĹĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĨĦ
        ĬĞÑĦĹgŃĦĨÍĨĻĹ κł hĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ấ ċ
                         ĬĞÑĦĹgĤŃĦĨÍĨĻĹ w ċ
                         âĮĴgĹĶĹĻâĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĹĦĹ κł hĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ấ ċ
                        ĬĞÑĦĹgĤĠĨĞĹĦĹ w ċ
                        âĮĴgĹĶĹĻâĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã

    fi CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴĤĞĻĦĹ
    ĤĦÌ ĴIJĹĻçłĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌãf
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κł ĝ
```

```
ƒ RMSĴĹIJĴ IJĴĻĨĬĨĨĨĢĤĹ
ĢĬĞĹĹ OĴĻĨĬĨĨĢĤĹçRMSĴĹIJĴf

    ƒ IĮĨĻĨĞĬĨĨĢĤ IJĴĻĨĬĨĨĢĤĹ h ĹĤĻ ĹĤĻĻĨĮĹĹ
    ÎĤÌ ççĨĮĨĻççâĹĤĬÌd ĬĤĞĹĮĨĮĹçĹĞĻĤłflgflflĝd ÎĤĢĞÑłflgd ĤĴĹĨĬIJĮłĝĤhìd
                ĹÎIJłflgîãf
        ĹĤĬÌgĬĤĞĹĮĨĮĹçĹĞĻĤ ł ĬĤĞĹĮĨĮĹçĹĞĻĤ
        ĹĤĬÌgĢłĹĹĤĮĻçĬĤĞĹĮĨĮĹçĹĞĻĤ ł ĬĤĞĹĮĨĮĹçĹĞĻĤ
        ĹĤĬÌgÎĤĢĞÑ ł ÎĤĢĞÑ
        ĹĤĬÌgĨĻĤĹĞĻĨIJĮĹ ł fl
        ĹĤĬÌgĤĴĹĨĬIJĮ ł ĤĴĹĨĬIJĮ
        ĹĤĬÌgĹÎIJ ł ĹÎIJ

    ƒ CĞĬĬ IJĮĢĤ ĠĤÌIJĹĤ ĞĮÑ ĴĞĹĞĬĤĻĤĹ łĴÎĞĻĤĹ
    ÎĤÌ ĴĹĤçłĴÎĞĻĤçĴĞĹĞĬĹâĹĤĬÌãf
        ĨÌ ĹĤĬÌgÎĤĢĞÑf
            ĹĤĬÌgĢłĹĹĤĮĻçĬĤĞĹĮĨĮĹçĹĞĻĤ ł ĹĤĬÌgĬĤĞĹĮĨĮĹçĹĞĻĤ ấ ċ
                âĝg w âĝg κ ĹĤĬÌgÎĤĢĞÑ ấ ĹĤĬÌgĨĻĤĹĞĻĨIJĮĹãã

    ƒ UĴÎĞĻĤ ĴĞĹĞĬĤĻĤĹĹ
    ÎĤÌ łĴÎĞĻĤçĴĞĹĞĬĹâĹĤĬÌd ĬĞÑĤĹãf

        ƒ IÌ ĬĞÑĤĹ ÎIJĤĹ ĮIJĮ ĢIJĮĻĞĨĮ ĢĞĢÎĤ ĞĹĹĞÑĹd
        ƒ ĢĹĤĞĻĤ ĻÎĤĬ ÌĨĬĬĤÎ ŃĨĻÎ ŅĤĹIJĹ
        ĨÌ ĮIJĮ ÎĞĹĞĻĻĹâĬĞÑĤĹd ďŃĤĨĬĬÎĻçĢĞĢÎĤďãf
            ĬĞÑĤĹgŃĤĨĬĬÎĻçĢĞĢÎĤ ł ĮĴgŅĤĹIJĹçĬĨĬĤâĬĞÑĤĹgŃĤĨĬĬÎĻĹã
            ĬĞÑĤĹgĠĨĞĹçĢĞĢÎĤ ł ĮĴgŅĤĹIJĹçĬĨĬĤâĬĞÑĤĹgĠĨĞĹĤĹã

        ƒ UĴÎĞĻĤ ĢĞĢÎĤ ŃĨĻÎ ĹĶŁĞĹĤÎ ĢŁĹĹĤĮĻ ĬĹĞÎĨĤĮĻĹ
        ĬĞÑĤĹgŃĤĨĬĬÎĻçĢĞĢÎĤ ł ĹĤĬÌgĹÎIJ ấ ĬĞÑĤĹgŃĤĨĬĬÎĻçĢĞĢÎĤ κ ċ
            âĝ h ĹĤĬÌgĹÎIJã ấ ĬĞÑĤĹgÎŃĤĨĬĬÎĻĹấấğ
        ĬĞÑĤĹgĠĨĞĹçĢĞĢÎĤ ł ĹĤĬÌgĹÎIJ ấ ĬĞÑĤĹgĠĨĞĹçĢĞĢÎĤ κ ċ
            âĝ h ĹĤĬÌgĹÎIJã ấ ĬĞÑĤĹgÎĠĨĞĹĤĹấấğ

        ƒ VĞĮĨĬĬĞ SGD ĴĞĹĞĬĤĻĤĹ łĴÎĞĻĤ κ IJĮĹĬĞĬĨŃĞĻĨIJĮ
        ƒ ŃĨĻÎ ĹĶŁĞĹĤ ĹIJIJĻĤÎ ĢĞĢÎĤ
        ĬĞÑĤĹgŃĤĨĬĬÎĻĹ κł hĹĤĬÌgĢłĹĹĤĮĻçĬĤĞĹĮĨĮĹçĹĞĻĤ ấ ċ
                        ĬĞÑĤĹgÎŃĤĨĬĬÎĻĹ w ċ
                        âĮĴgĹĶĹĻâĬĞÑĤĹgŃĤĨĬĬÎĻçĢĞĢÎĤã κ ĹĤĬÌgĤĴĹĨĬIJĮã
        ĬĞÑĤĹgĠĨĞĹĤĹ κł hĹĤĬÌgĢłĹĹĤĮĻçĬĤĞĹĮĨĮĹçĹĞĻĤ ấ ċ
                        ĬĞÑĤĹgÎĠĨĞĹĤĹ w ċ
                        âĮĴgĹĶĹĻâĬĞÑĤĹgĠĨĞĹçĢĞĢÎĤã κ ĹĤĬÌgĤĴĹĨĬIJĮã

    ƒ CĞĬĬ IJĮĢĤ ĞÌĻĤĹ ĞĮÑ ĴĞĹĞĬĤĻĤĹ łĴÎĞĻĤĹ
    ÎĤÌ ÎIJĹĻçłĴÎĞĻĤçĴĞĹĞĬĹâĹĤĬÌãf
        ĹĤĬÌgĨĻĤĹĞĻĨIJĮĹ κł ĝ
```

```
ƒ AĤĞİ IJĴĻĨİĨŅĦĹ
ĢĬĞĽĽ OĴĻĨİĨŅĦĹçAĤĞİf

    ƒ IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĹĽ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮĹçĹĞĻĦłƒlgƒlƒlĝd ĤĦĢĞÑłƒlgd ĦĴĽĨĬIJĮłĝĦhìd
                ĠĦĻĞçĝłƒlgîd ĠĦĻĞçğłƒlgîîîãf
        ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł ƒl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬÌgĠĦĻĞçĝ ł ĠĦĻĞçĝ
        ĽĦĬÌgĠĦĻĞçğ ł ĠĦĻĞçğ

    ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ łĴĤĞĻĦĽ
    ĤĦÌ ĴĹĦçłĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgĤĦĢĞÑf
            ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ƕ ċ
                âĝg w âĝg κ ĽĦĬÌgĤĦĢĞÑ ƕ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    ƒ UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
    ĤĦÌ łĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

        ƒ IÌ ĬĞÑĦĹ ĤIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĨĦ ĞĹĹĞÑĽd
        ƒ ĢĹĦĞĻĦ ĻĨĦİ ÌĨĬĬĦĤ ŃĨĻĨ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ĨĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĹĨĻçĢĞĢĨĦďãf
            ĬĞÑĦĹgŃĦĨĹĨĻçĬIJĬĦĮĻłĬĽ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgŃĦĨĹĨĻĽã
            ĬĞÑĦĹgŃĦĨĹĨĻçĢĞĢĨĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgŃĦĨĹĨĻĽã
            ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻłĬĽ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgĠĨĞĽĦĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgĠĨĞĽĦĽã

        ƒ UĴĤĞĻĦ ĬIJĬĦĮĻłĬ  ŃĨĻĨ ĢłĹĹĦĮĻ ĹĹĞĤĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨĹĨĻçĬIJĬĦĮĻłĬĽ ł ĽĦĬÌgĠĦĻĞçĝ ƕ ċ
                                  ĬĞÑĦĹgŃĦĨĹĨĻçĬIJĬĦĮĻłĬĽ κ ċ
                                  âĝ h ĽĦĬÌgĠĦĻĞçĝã ƕ ĬĞÑĦĹgĤŃĦĨĹĨĻĽ
        ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻłĬĽ ł ĽĦĬÌgĠĦĻĞçĝ ƕ ċ
                                ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻłĬĽ κ ċ
                                âĝ h ĽĦĬÌgĠĦĻĞçĝã ƕ ĬĞÑĦĹgĤĠĨĞĽĦĽ
        ƒ GĦĻ ĢIJĹĹĦĢĻĦĤ ĬIJĬĦĮĻłĬ
        ƒ ĽĦĬÌgĨĻĦĹĞĻĨIJĮ ĨĽ ƒl ĞĻ ÌĨĹĽĻ ĴĞĽĽ
        ƒ ĞĮĤ ŃĦ ĮĦĦĤ ĻIJ ĽĻĞĹĻ ŃĨĻĨ ĝ ĬĦĹĦ
        ŃĦĨĹĨĻçĬIJĬĦĮĻłĬĽçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨĹĨĻçĬIJĬĦĮĻłĬĽ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçĝ ƕƕ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã
        ĠĬĞĽçĬIJĬĦĮĻłĬĽçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻłĬĽ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçĝ ƕƕ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã
        ƒ UĴĤĞĻĦ ĢĞĢĨĦ ŃĨĻĨ ĽĶłĞĹĦĤ ĢłĹĹĦĮĻ ĹĹĞĤĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨĹĨĻçĢĞĢĨĦ ł ĽĦĬÌgĠĦĻĞçğ ƕ ĬĞÑĦĹgŃĦĨĹĨĻçĢĞĢĨĦ κ ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğã ƕ ĬĞÑĦĹgĤŃĦĨĹĨĻĽƕƕğ
```

```
        ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ ł ĽĦĬÌgĠĦĻĞçğ ǻ ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ κ ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğã ǻ ĬĞÑĦĹgĤĠĨĞĽĦĽǻǻğ
        ſ GĦĻ ĢIJĹĹĦĢĻĦĤ ĢĞĢĨĦ
        ŃĦĨÍÎĻçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢĨĦ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğ ǻǻ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã
        ĠĨĞĽçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğ ǻǻ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã

        ſ VĞĮĨĬĬĞ SGD ĴĞĹĞĠĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹĠĞĬĨŅĞĻĨIJĮ
        ſ ŃĨĻÎ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĨĦ
        ĬĞÑĦĹgŃĦĨÍÎĻĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ǻ ċ
                        ŃĦĨÍÎĻçĬIJĠĦĮĻŁĠĽçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĽĶĹĻâŃĦĨÍÎĻçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤã κ
                            ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ǻ ċ
                        ĠĨĞĽçĬIJĠĦĮĻŁĠĽçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĽĶĹĻâĠĨĞĽçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤã κ
                            ĽĦĬÌgĦĴĽĨĬIJĮã

    ſ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĠĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞĠĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ


ſ CIJĠĬIJĮ ĬIJĽĽ ĢĬĞĽĽ
ĢĬĞĽĽ LIJĽĽf

    ſ RĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽ ĢĞĬĢŁĬĞĻĨIJĮ
    ĤĦÌ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽâĽĦĬÌd ĬĞÑĦĹãf

        ſ fl ĠÑ ĤĦÌĞŁĬĻ
        ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ ł fl

        ſ Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ŃĦĨÍÎĻĽ
        ſ ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ fl
        ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ǻ ċ
                                    ĮĴgĽŁĬâĮĴgĞĠĽâĬĞÑĦĹgŃĦĨÍÎĻĽãã

        ſ Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ŃĦĨÍÎĻĽ
        ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ǻ ċ
                                    ĮĴgĽŁĬâĬĞÑĦĹgŃĦĨÍÎĻĽ ǻ ċ
                                            ĬĞÑĦĹgŃĦĨÍÎĻĽã
```

```
        ſi Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
        ſi ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ ſl
        ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń ſlf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ĸł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ő ċ
                                    ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgĠĨĞĽĦĽãã

        ſi Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
        ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ſlf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ĸł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                    ĮĴgĽŁİâĬĞÑĦĹgĠĨĞĽĦĽ ő ċ
                                            ĬĞÑĦĹgĠĨĞĽĦĽã

        ĹĦĻŁĹĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ

    ſi CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽĦĽ
    ſi ÍĨĿĦĮ ĬIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñãf

        ſi CĞĬĢŁĬĞĻĦ ĽĞĬĴĬĦ ĬIJĹĽĦĽ
        ĽĞĬĴĬĦçĬIJĹĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        ſi CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĹĽ
        ÂĞĻĞçĬIJĹĽ ł ĮĴgĬĦĞĮâĽĞĬĴĬĦçĬIJĹĽĦĽã

        ſi RĦĻŁĹĮ ĬIJĹĽ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĽ


ſi CĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ
ĢĬĞĹĽ LIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâLIJĹĽãf

    ſi FIJŃĞĹÂ ĴĞĹĽ
    ÂĦÌ ÌIJŃĞĹÂâĽĦĬÌd ÑçĴĹĦÂd ÑçĻĹŁĦãf

        ſi NŁĬĠĦĹ IJÌ ĽĞĬĴĬĦĽ ĨĮ Ğ ĠĞĻĢÎ
        ĽĞĬĴĬĦĽ ł ĬĦĮâÑçĴĹĦÂã

        ſi CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦĿĦĮĻ ÂĨĿĨĹĨIJĮ ĠÑ ſl
        ſi CĬĨĴ ĠIJĻÎ ĹĨÂĦĽ ĻIJ ĮIJĻ ÂĹĞÍ ĬĦĞĮ ĻIJŃĞĹÂĽ ĞĮÑ ĿĞĬŁĦ
        ÑçĴĹĦÂçĢĬĨĴĴĦÂ ł ĮĴgĢĬĨĴâÑçĴĹĦÂd ĝĦhìd ĝ h ĝĦhìã

        ſi PĹIJĠĞĠĨĬĨĻĨĦĽ ÌIJĹ ĻĞĹÍĦĻ ĿĞĬŁĦĽ h
        ſi IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĽ ł ÑçĴĹĦÂçĢĬĨĴĴĦÂä
                ĹĞĮÍĦâĽĞĬĴĬĦĽãd
                ÑçĻĹŁĦ
            ã
```

```
        # MĞĽĪ ĽĞĬŁĦĽ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJÂĦĦÂ ĬĞĠĦĬĽ
        ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÎÂĦĮĢĦĽ ł ĮĴgĽŁÌâ
                ÑçĴĹĦÂçĢĬĨĴĴĦÂ ấ ÑçĻĹŁĦd
                ĞŇĨĹłĝ
            ã

        # LIJĽĽĦĽ
        ĮĦÍĞĻĨĽĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĽ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨÎÂĦĮĢĦĽã
        ĹĦĻŁĹĮ ĮĦÍĞĻĨĽĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĽ

    # BĞĢĪŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ĠĞĢĪŃĞĹÂâĽĦĬÌd ÂĽĞĬŁĦĽd ÑçĻĹŁĦãf

        # NŁĬĠĦĹ IJÌ ĽĞĬĴĬĦĽ
        ĽĞĬĴĬĦĽ ł ĬĦĮâÂĽĞĬŁĦĽã
        # NŁĬĠĦĹ IJÌ ĬĞĠĦĬĽ ĨĮ ĦĽĦĹÑ ĽĞĬĴĬĦ
        # WĦďĬĬ ŁĽĦ ĻÎĦ ÌĨĹĽĻ ĽĞĬĴĬĦ ĻIJ ĢIJŁĮĻ ĻÎĦĬ
        ĬĞĠĦĬĽ ł ĬĦĮâÂĽĞĬŁĦĽäflãã

        # IÌ ĬĞĠĦĬĽ ĞĹĦ ĽĴĞĹĽĦd ĻŁĹĮ ĻÎĦĬ ĨĮĻIJ IJĮĦhÎIJĻ ĽĦĢĻIJĹ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ÑçĻĹŁĦ ł ĮĴgĦÑĦâĬĞĠĦĬĽãäÑçĻĹŁĦã

        # CĞĬĢŁĬĞĻĦ ÍĹĞÂĨĦĮĻ
        ĽĦĬÌgÂĨĮĴŁĻĽ ł hÑçĻĹŁĦ w ÂĽĞĬŁĦĽ
        # NIJĹĬĞĬĨŅĦ ÍĹĞÂĨĦĮĻ
        ĽĦĬÌgÂĨĮĴŁĻĽ ł ĽĦĬÌgÂĨĮĴŁĻĽ w ĽĞĬĴĬĦĽ


# SIJÌĻĬĞŇ ĢĬĞĽĽĨÌÌĦĹ h ĢIJĬĠĨĮĦÂ SIJÌĻĬĞŇ ĞĢĻĨĽĞĻĨIJĮ
# ĞĮÂ ĢĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ ÌIJĹ ÌĞĽĻĦĹ ĠĞĢĪŃĞĹÂ ĽĻĦĴ
ĢĬĞĽĽ AĢĻĨĽĞĻĨIJĮçSIJÌĻĬĞŇçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâãf

    # CĹĦĞĻĦĽ ĞĢĻĨĽĞĻĨIJĮ ĞĮÂ ĬIJĽĽ ÌŁĮĢĻĨIJĮ IJĠÏĦĢĻĽ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ĽĦĬÌgĞĢĻĨĽĞĻĨIJĮ ł AĢĻĨĽĞĻĨIJĮçSIJÌĻĬĞŇâã
        ĽĦĬÌgĬIJĽĽ ł LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

    # FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ĨĮĴŁĻĽd ÑçĻĹŁĦãf
        # OŁĻĴŁĻ ĬĞÑĦĹďĽ ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĞĢĻĨĽĞĻĨIJĮgÌIJĹŃĞĹÂâĨĮĴŁĻĽã
        # SĦĻ ĻÎĦ IJŁĻĴŁĻ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĽĦĬÌgĞĢĻĨĽĞĻĨIJĮgIJŁĻĴŁĻ
        # CĞĬĢŁĬĞĻĦ ĞĮÂ ĹĦĻŁĹĮ ĬIJĽĽ ĽĞĬŁĦ
        ĹĦĻŁĹĮ ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâĽĦĬÌgIJŁĻĴŁĻd ÑçĻĹŁĦã
```

```
    fi BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎHÌ ĠĞĢĪŃĞĹÎâĽHĬÌd ÎĹĞĬŁHĹd ÑçĻĹŁHãf

        fi NŁİĠHĹ IJÌ ĽĞİĴĬHĽ
        ĽĞİĴĬHĽ ł ĬHĮâÎĹĞĬŁHĽã

        fi IÌ ĬĞĠHĬĽ ĞĹH IJĮHhÎIJĻ HĮĢIJÎHÎd
        fi ĻŁĹĮ ĻÎHİ ĨĮĻIJ ÎĨĽĢĹHĻH ĹĞĬŁHĽ
        ĨÌ ĬHĮâÑçĻĹŁHgĽÎĞĴHã łł ğf
            ÑçĻĹŁH ł ĮĴgĞĹÍİĞŃâÑçĻĹŁHd ĞŃĨĽłĝã

        fi CIJĴÑ ĽIJ ŃH ĢĞĮ ĽĞÌHĬÑ İIJÎĨÌÑ
        ĽHĬÌgÎĨĮĴŁĻĽ ł ÎĹĞĬŁHĽgĢIJĴÑâã
        fi CĞĬĢŁĬĞĻH ÍĹĞÎĨHĮĻ
        ĽHĬÌgÎĨĮĴŁĻĽäĹĞĮÍHâĽĞİĴĬHĽãd ÑçĻĹŁHã hł ĝ
        fi NIJĹİĞĬĨŃH ÍĹĞÎĨHĮĻ
        ĽHĬÌgÎĨĮĴŁĻĽ ł ĽHĬÌgÎĨĮĴŁĻĽ w ĽĞİĴĬHĽ


fi CĹHĞĻH ÎĞĻĞĹHĻ
Xd Ñ ł ĽĴĨĹĞĬçÎĞĻĞâĽĞİĴĬHĽłĝflflfld ĢĬĞĹĽHĽłġã

fi CĹHĞĻH DHĮĽH ĬĞÑHĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌHĞĻŁĹHĽ ĞĮÎ ħġ IJŁĻĴŁĻ ĹĞĬŁHĽ
ÎHĮĽHĝ ł LĞÑHĹçDHĮĽHâğd ħġd ŃHĨÍÎĻçĹHÍŁĬĞĹĨŃHĹçĬğłĥHHġd
                                ĠĨĞĹçĹHÍŁĬĞĹĨŃHĹçĬğłĥHHġã

fi CĹHĞĻH RHLU ĞĢĻĨĹĞĻĨIJĮ âĻIJ ĠH ŁĽHÎ ŃĨĻÎ DHĮĽH ĬĞÑHĹãf
ĞĢĻĨĹĞĻĨIJĮĝ ł AĢĻĨĹĞĻĨIJĮçRHLUâã

fi CĹHĞĻH ÎĹIJĴIJŁĻ ĬĞÑHĹ
ÎĹIJĴIJŁĻĝ ł LĞÑHĹçDĹIJĴIJŁĻâflgĝã

fi CĹHĞĻH ĽHĢIJĮÎ DHĮĽH ĬĞÑHĹ ŃĨĻÎ ħġ ĨĮĴŁĻ ÌHĞĻŁĹHĽ âĞĽ ŃH ĻĞĬH IJŁĻĴŁĻ
fi IJÌ ĴĹHĹĨIJŁĽ ĬĞÑHĹ ÎHĹHã ĞĮÎ ġ IJŁĻĴŁĻ ĹĞĬŁHĽ âIJŁĻĴŁĻ ĹĞĬŁHĽã
ÎHĮĽHğ ł LĞÑHĹçDHĮĽHâħġd ġã

fi CĹHĞĻH SIJÌĻİĞÑ ĢĬĞĽĽĨÌĨHĹďĽ ĢIJİĠĨĮHÎ ĬIJĽĽ ĞĮÎ ĞĢĻĨĹĞĻĨIJĮ
ĬIJĽĽçĞĢĻĨĹĞĻĨIJĮ ł AĢĻĨĹĞĻĨIJĮçSIJÌĻİĞÑçLIJĽĽçCĞĻHÍIJĹĨĢĞĬCĹIJĽĽHĮĻĹIJĴÑâã

fi CĹHĞĻH IJĴĻĨİĨŃHĹ
IJĴĻĨİĨŃHĹ ł OĴĻĨİĨŃHĹçAÎĞİâĬHĞĹĮĨĮÍçĹĞĻHłflgflĥd ÎHĢĞÑłĥHHĥã

fi TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ HĴIJĢÎ ĨĮ ĹĞĮÍHâĝflflflĝãf

    fi PHĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ÎĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑHĹ
    ÎHĮĽHĝgÌIJĹŃĞĹÎâXã
```

```
ƒ PHĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻÎĿĞĻÎIJĮ ÌŁĮĢĻÎIJĮ
ƒ ĻĞÎHĽ ĻÎH IJŁĻĴŁĻ IJÌ ÌÎĹĽĻ ÂHĮĽH ÎĞÑHĹ ÎHĹH
ĞĢĻÎĿĞĻÎIJĮĝgÌIJĹŃĞĹÂâÂHĮĽHĝgIJŁĻĴŁĻã

ƒ PHĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ DĹIJĴIJŁĻ ÎĞÑHĹ
ÂĹIJĴIJŁĻĝgÌIJĹŃĞĹÂâĞĢĻÎĿĞĻÎIJĮĝgIJŁĻĴŁĻã

ƒ PHĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽHĢIJĮÂ DHĮĽH ÎĞÑHĹ
ƒ ĻĞÎHĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻÎĿĞĻÎIJĮ ÌŁĮĢĻÎIJĮ IJÌ ÌÎĹĽĻ ÎĞÑHĹ ĞĽ ÎĮĴŁĻĽ
ÂHĮĽHĝgÌIJĹŃĞĹÂâÂĹIJĴIJŁĻĝgIJŁĻĴŁĻã

ƒ PHĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĻÎH ĞĢĻÎĿĞĻÎIJĮwÎIJĽĽ ÌŁĮĢĻÎIJĮ
ƒ ĻĞÎHĽ ĻÎH IJŁĻĴŁĻ IJÌ ĽHĢIJĮÂ ÂHĮĽH ÎĞÑHĹ ÎHĹH ĞĮÂ ĹHĻŁĹĮĽ ÎIJĽĽ
ÂĞĻĞçÎIJĽĽ ł ÎIJĽĽçĞĢĻÎĿĞĻÎIJĮgÌIJĹŃĞĹÂâÂHĮĽHĝgIJŁĻĴŁĻd Ñã

ƒ CĞÎĢŁÎĞĻH ĹHÍŁÎĞĹÎŃĞĻÎIJĮ ĴHĮĞÎĻÑ
ĹHÍŁÎĞĹÎŃĞĻÎIJĮçÎIJĽĽ ł ċ
    ÎIJĽĽçĞĢĻÎĿĞĻÎIJĮgÎIJĽĽgĹHÍŁÎĞĹÎŃĞĻÎIJĮçÎIJĽĽâÂHĮĽHĝã к ċ
    ÎIJĽĽçĞĢĻÎĿĞĻÎIJĮgÎIJĽĽgĹHÍŁÎĞĹÎŃĞĻÎIJĮçÎIJĽĽâÂHĮĽHĝã

ƒ CĞÎĢŁÎĞĻH IJĿHĹĞÎÎ ÎIJĽĽ
ÎIJĽĽ ł ÂĞĻĞçÎIJĽĽ к ĹHÍŁÎĞĹÎŃĞĻÎIJĮçÎIJĽĽ

ƒ CĞÎĢŁÎĞĻH ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻÎĿĞĻÎIJĮğ ĞĮÂ ĻĞĹÍHĻĽ
ƒ ĢĞÎĢŁÎĞĻH ĿĞÎŁHĽ ĞÎIJĮÍ ÌÎĹĽĻ ĞŃÎĽ
ĴĹHÂÎĢĻÎIJĮĽ ł ĮĴgĞĹÍİĞŃâÎIJĽĽçĞĢĻÎĿĞĻÎIJĮgIJŁĻĴŁĻd ĞŃÎĽłĝã
ÎÌ ÎHĮâÑgĽÎĞĴHã łł ğf
    Ñ ł ĮĴgĞĹÍİĞŃâÑd ĞŃÎĽłĝã
ĞĢĢŁĹĞĢÑ ł ĮĴgİHĞĮâĴĹHÂÎĢĻÎIJĮĽłłÑã

ÎÌ ĮIJĻ HĴIJĢÎ ķ ĝflflf
    ĴĹÎĮĻâÌďHĴIJĢÎf ăHĴIJĢÎåd ď к
          ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď к
          ÌďÎIJĽĽf ăÎIJĽĽfgġÌå âď к
          ÌďÂĞĻĞçÎIJĽĽf ăÂĞĻĞçÎIJĽĽfgġÌåd ď к
          ÌďĹHÍçÎIJĽĽf ăĹHÍŁÎĞĹÎŃĞĻÎIJĮçÎIJĽĽfgġÌåãd ď к
          ÌďÎĹf ăIJĴĻÎİÎŃHĹgĢŁĹĹHĮĻçÎHĞĹĮÎĮÍçĹĞĻHåďã

ƒ BĞĢÎŃĞĹÂ ĴĞĽĽ
ÎIJĽĽçĞĢĻÎĿĞĻÎIJĮgĠĞĢÎŃĞĹÂâÎIJĽĽçĞĢĻÎĿĞĻÎIJĮgIJŁĻĴŁĻd Ñã
ÂHĮĽHĝgĠĞĢÎŃĞĹÂâÎIJĽĽçĞĢĻÎĿĞĻÎIJĮgÂÎĮĴŁĻĽã
ÂĹIJĴIJŁĻĝgĠĞĢÎŃĞĹÂâÂHĮĽHĝgÂÎĮĴŁĻĽã
ĞĢĻÎĿĞĻÎIJĮĝgĠĞĢÎŃĞĹÂâÂĹIJĴIJŁĻĝgÂÎĮĴŁĻĽã
ÂHĮĽHĝgĠĞĢÎŃĞĹÂâĞĢĻÎĿĞĻÎIJĮĝgÂÎĮĴŁĻĽã

ƒ UĴÂĞĻH ŃHÎÍÎĻĽ ĞĮÂ ĠÎĞĽHĽ
IJĴĻÎİÎŃHĹgĴĹHçŁĴÂĞĻHçĴĞĹĞİĽâã
IJĴĻÎİÎŃHĹgŁĴÂĞĻHçĴĞĹĞİĽâÂHĮĽHĝã
IJĴĻÎİÎŃHĹgŁĴÂĞĻHçĴĞĹĞİĽâÂHĮĽHĝã
IJĴĻÎİÎŃHĹgĴIJĽĻçŁĴÂĞĻHçĴĞĹĞİĽâã
```

```
ńi VĞĬĨĤĞĻĦ ĻĨĦ ĬIJĤĦĬ

ńi CĹĦĞĻĦ ĻĦĹĻ ĤĞĻĞĹĦĻ
XçĻĦĹĻd ÑçĻĦĹĻ ł ĹĴĨĹĞĬçĤĞĻĞâĹĞĬĴĬĦĹłĝńńd ĢĬĞĹĹĦĹłġã

ńi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ IJÌ IJłĹ ĻĦĹĻĨĮÍ ĤĞĻĞ ĻĨĹIJłÍĨ ĻĨĨĹ ĬĞÑĦĹ
ĤĦĮĹĦĝgÌIJĹŃĞĹĤâXçĻĦĹĻã

ńi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ ĻĨĹIJłÍĨ ĞĢĻĨĹ·ĞĻĨIJĮ ÌłĮĢĻĨIJĮ
ńi ĻĞĪĦĹ ĻĨĦ IJłĻĴłĻ IJÌ ÌĨĹĹĻ ĤĦĮĹĦ ĬĞÑĦĹ ĨĦĹĦ
ĞĢĻĨĹ·ĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĹĦĝgIJłĻĴłĻã

ńi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ ĻĨĹIJłÍĨ ĹĦĢIJĮĤ DĦĮĹĦ ĬĞÑĦĹ
ńi ĻĞĪĦĹ IJłĻĴłĻĹ IJÌ ĞĢĻĨĹ·ĞĻĨIJĮ ÌłĮĢĻĨIJĮ IJÌ ÌĨĹĹĻ ĬĞÑĦĹ ĞĹ ĨĮĴłĻĹ
ĤĦĮĹĦğgÌIJĹŃĞĹĤâĞĢĻĨĹ·ĞĻĨIJĮĝgIJłĻĴłĻã

ńi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĹĹ ĻĨĹIJłÍĨ ĻĨĦ ĞĢĻĨĹ·ĞĻĨIJĮwĬIJĹĹ ÌłĮĢĻĨIJĮ
ńi ĻĞĪĦĹ ĻĨĦ IJłĻĴłĻ IJÌ ĹĦĢIJĮĤ ĤĦĮĹĦ ĬĞÑĦĹ ĨĦĹĦ ĞĮĤ ĹĦĻłĹĮĹ ĬIJĹĹ
ĬIJĹĹ ł ĬIJĹĹçĞĢĻĨĹ·ĞĻĨIJĮgÌIJĹŃĞĹĤâĤĦĮĹĦğgIJłĻĴłĻd ÑçĻĦĹĻã

ńi CĞĬĢłĬĞĻĦ ĞĢĢłĹĞĢÑ ÌĹIJİ IJłĻĴłĻ IJÌ ĞĢĻĨĹ·ĞĻĨIJĮğ ĞĮĤ ĻĞĹĹĦĻĹ
ńi ĢĞĬĢłĬĞĻĦ L·ĞĬłĦĹ ĞĬIJĮĬ ÌĨĹĹĻ ĞŃĨĹ
ĴĹĦĤĨĢĻĨIJĮĹ ł ĮĴgĞĹĹĬĞŃâĬIJĹĹçĞĢĻĨĹ·ĞĻĨIJĮgIJłĻĴłĻd ĞŃĨĹłĝã
ĨÌ ĬĦĮâÑçĻĦĹĻgĹĬĞĴĦã łł ğf
    ÑçĻĦĹĻ ł ĮĴgĞĹĹĬĞŃâÑçĻĦĹĻd ĞŃĨĹłĝã
ĞĢĢłĹĞĢÑ ł ĮĴgĬĦĞĮâĴĹĦĤĨĢĻĨIJĮĹłłÑçĻĦĹĻã

ĴĹĨĮĻâÌďĹ·ĞĬĨĤĞĻĨIJĮd ĞĢĢf ăĞĢĢłĹĞĢÑfgġÌåd ĬIJĹĹf ăĬIJĹĹfgġÌåďã


ńńń
ĦĴIJĢĬf îîńńd ĞĢĢf ńgĥĥíd ĬIJĹĹf ńgìġġ âĤĞĻĞçĬIJĹĹf ńgìĝìd ĹĦÍçĬIJĹĹf ńgńĝĥãd ĬĹf
ńgńġġġĝĥîġġĥġĥĥġġì
ĦĴIJĢĬf ĝńńńńd ĞĢĢf ńgĥííd ĬIJĹĹf ńgìğì âĤĞĻĞçĬIJĹĹf ńgìĝĝd ĹĦÍçĬIJĹĹf ńgńĝĥãd ĬĹf
ńgńġġġġġġġġíĝġíğìĝ
L·ĞĬĨĤĞĻĨIJĮd ĞĢĢf ńgìĥìd ĬIJĹĹf ńgìĝğ
```

Fig 15.04: Model trained with dropout.

Anim 15.04: https://nnfs.io/efg

While our accuracy and loss have suffered considerably, we've found a scenario where our validation set performs *better* than our in-sample set (because we do not apply dropout when testing so you don't disable some of the connections). Further tweaking would likely fix the accuracy issue; for example, due to our regularization tactics, we can change our layer sizes to 512:

```
fi CĹĦĞĻĦ DĦĮĹĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĹ ĞĮĤ ĥĝğ IJŁĻĴŁĻ ĿĞĬŁĦĹ
ĤĦĮĹĦĝ ł LĞÑĦĹçDĦĮĹĦâğd ĥĝğd ŃĦĨÍĨĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦhǵd
                                ĠĨĞĹçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦhǵã

fi CĹĦĞĻĦ RĦLU ĞĢĻĨĿĞĻĨIJĮ âĻIJ ĠĦ ŁĹĦĤ ŃĨĻÎ DĦĮĹĦ ĬĞÑĦĹãf
ĞĢĻĨĿĞĻĨIJĮĝ ł AĢĻĨĿĞĻĨIJĮçRĦLUâã
```

```
ﬁ CĹĦĞĻĦ ĤĹIJĴIJŁĻ ĬĞÑĦĹ
ĤĹIJĴIJŁĻĝ ł LĞÑĦĹçDĹIJĴIJŁĻâﬂgĝã

ﬁ CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĬĻÎ ĥĝğ ĬĮĴŁĻ ÌĦĞĻŁĹĦĽ
ﬁ ĞĮĤ ġ IJŁĻĴŁĻ ŁĞĬŁĦĽ
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâĥĝğd ġã
```

Adding more neurons ends with:

```
ĦĴIJĢÎf ﬂd ĞĢĢf ﬂgġìġd ĬIJĹĽf ĝgﬂîî âĤĞĻĞçĬIJĹĽf ĝgﬂîîd ĹĦÍçĬIJĹĽf ﬂgﬂﬂﬂãd ĬĹf
ﬂgﬂĥ
ĦĴIJĢÎf ĝﬂﬂd ĞĢĢf ﬂgìĝîd ĬIJĹĽf ﬂgìġĥ âĤĞĻĞçĬIJĹĽf ﬂgħìğd ĹĦÍçĬIJĹĽf ﬂgﬂħġãd ĬĹf
ﬂgﬂġîìĥġìĝîﬂîﬂĥﬂğﬂğ
ĦĴIJĢÎf ğﬂﬂd ĞĢĢf ﬂgìíğd ĬIJĹĽf ﬂgħğì âĤĞĻĞçĬIJĹĽf ﬂgĥġíd ĹĦÍçĬIJĹĽf ﬂgﬂìîãd ĬĹf
ﬂgﬂġîĥﬂìġﬂĝġĥħĥﬂğíﬂħ
ĦĴIJĢÎf ġﬂﬂd ĞĢĢf ﬂgíﬂﬂd ĬIJĹĽf ﬂgħﬂġ âĤĞĻĞçĬIJĹĽf ﬂgĥğĝd ĹĦÍçĬIJĹĽf ﬂgﬂíğãd ĬĹf
ﬂgﬂġîğħġĥĝﬂĥĝììĥîĥ
ĦĴIJĢÎf ġﬂﬂd ĞĢĢf ﬂgíﬂğd ĬIJĹĽf ﬂgĥîĥ âĤĞĻĞçĬIJĹĽf ﬂgĥĝġd ĹĦÍçĬIJĹĽf ﬂgﬂíğãd ĬĹf
ﬂgﬂġîﬂğğﬂĝﬂííğííħġğ
ĦĴIJĢÎf ĥﬂﬂd ĞĢĢf ﬂgíﬂîd ĬIJĹĽf ﬂgĥħğ âĤĞĻĞçĬIJĹĽf ﬂgġíğd ĹĦÍçĬIJĹĽf ﬂgﬂìîãd ĬĹf
ﬂgﬂġíìíğíħìġĥħîġîĝğĥ
ĦĴIJĢÎf ħﬂﬂd ĞĢĢf ﬂgíġħd ĬIJĹĽf ﬂgĥğĝ âĤĞĻĞçĬIJĹĽf ﬂgġġĥd ĹĦÍçĬIJĹĽf ﬂgﬂìħãd ĬĹf
ﬂgﬂġíĥġħﬂġĥîğġĥĥîġĥ
ĦĴIJĢÎf ìﬂﬂd ĞĢĢf ﬂgíĝħd ĬIJĹĽf ﬂgĥġğ âĤĞĻĞçĬIJĹĽf ﬂgġĥìd ĹĦÍçĬIJĹĽf ﬂgﬂìħãd ĬĹf
ﬂgﬂġíġĝĝĥĝğħġġġħﬂĥĥħ
ĦĴIJĢÎf íﬂﬂd ĞĢĢf ﬂgíġîd ĬIJĹĽf ﬂgĥĝĥ âĤĞĻĞçĬIJĹĽf ﬂgġġğd ĹĦÍçĬIJĹĽf ﬂgﬂìġãd ĬĹf
ﬂgﬂġíﬂìîğġġĥìíĥíĥĥĝ
ĦĴIJĢÎf îﬂﬂd ĞĢĢf ﬂgíġğd ĬIJĹĽf ﬂgġîî âĤĞĻĞçĬIJĹĽf ﬂgġğħd ĹĦÍçĬIJĹĽf ﬂgﬂìğãd ĬĹf
ﬂgﬂġìíġîĝìîġíħĥìġĥğ
ĦĴIJĢÎf ĝﬂﬂﬂd ĞĢĢf ﬂgíġìd ĬIJĹĽf ﬂgġíﬂ âĤĞĻĞçĬIJĹĽf ﬂgġﬂíd ĹĦÍçĬIJĹĽf ﬂgﬂìĝãd
ĬĹf ﬂgﬂġìħğĝġĝĥġﬂﬂìğíħĝ
ggg
ĦĴIJĢÎf îíﬂﬂd ĞĢĢf ﬂgíġíd ĬIJĹĽf ﬂgġġġ âĤĞĻĞçĬIJĹĽf ﬂgġîĝd ĹĦÍçĬIJĹĽf ﬂgﬂĥğãd
ĬĹf ﬂgﬂġġĥĥíĝìġﬂîġﬂĥħíĝħ
ĦĴIJĢÎf îîﬂﬂd ĞĢĢf ﬂgíġĝd ĬIJĹĽf ﬂgġħí âĤĞĻĞçĬIJĹĽf ﬂgġĝħd ĹĦÍçĬIJĹĽf ﬂgﬂĥğãd
ĬĹf ﬂgﬂġġġĥîġġħġħħġìì
ĦĴIJĢÎf ĝﬂﬂﬂﬂd ĞĢĢf ﬂgíĥîd ĬIJĹĽf ﬂgġħí âĤĞĻĞçĬIJĹĽf ﬂgġĝìd ĹĦÍçĬIJĹĽf ﬂgﬂĥĝãd
ĬĹf ﬂgﬂġġġġġġġġíĝġíğìĝ
ŁĞĬĬĤĞĻĬIJĮd ĞĢĢf ﬂgíĥìd ĬIJĹĽf ﬂgġîì
```

Fig 15.05: Model trained with dropout and bigger hidden layer.

Anim 15.05: https://nnfs.io/fgh

Pretty good result, but worse compared to the "no dropout" model. Interestingly, validation accuracy is close to the training accuracy with dropout — usually validation accuracy will be higher, so we might suspect these as signs of overfitting here (validation loss is lower than expected).

Supplementary Material: https://nnfs.io/ch15
Chapter code, further resources, and errata for this chapter.

Chapter 16

Binary Logistic Regression

Now that we've learned how to create and train a neural network, let's consider an alternative output layer for a neural network. Until now, we've used an output layer that is a probability distribution, where all of the values represent a confidence level of a given class being the correct class, and where these confidences sum to *1*. We're now going to cover an alternate output layer option, where each neuron separately represents two classes — 0 for one of the classes, and a 1 for the other. A model with this type of output layer is called **binary logistic regression**. This single neuron could represent two classes like *cat* vs. *dog*, but it could also represent *cat* vs. *not cat* or any combination of 2 classes, and you could have many of these. For example, a model may have two binary output neurons. One of these neurons could be distinguishing between *person/not person*, and the other neuron could be deciding between *indoors/outdoors*. Binary logistic regression is a regressor type of algorithm, which will differ as we'll use a **sigmoid** activation function for the output layer rather than **softmax**, and **binary cross-entropy** rather than **categorical cross-entropy** for calculating loss.

Sigmoid Activation Function

The sigmoid activation function is used with regressors because it "squishes" a range of outputs from negative infinity to positive infinity to be between 0 and 1. The bounds represent the two possible classes. The sigmoid equation is:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

For the purpose of neural networks, we'll use our common notation:

$$\sigma_{i,j} = \frac{1}{1 + e^{-z_{i,j}}}$$

The denominator of the **Sigmoid** function contains *e* raised to the power of $z_{i,j}$, where *z*, given indices, means a singular output value of the layer that this activation function takes as input. The index *i* means the current sample, and the index *j* means the current output in this sample.

If we plot the sigmoid function:

Fig 16.1: The sigmoid function graph.

Note the output from this function averages at *0.5*, and squishes down to a flat line as it approaches *0* or *1*. The sigmoid function approaches both maximum and minimum values exponentially fast. For example, for an input of *2*, the output is *~0.88*, which is already pretty close to *1*. With an input of *3*, the output is *~0.95*, and so on. It's also similar for negative values: $\sigma(-2) \approx 0.12$ and $\sigma(-3) \approx 0.05$. This property makes the sigmoid activation function a good candidate to apply to the final layer's output with a binary logistic regression model.

For commonly-used functions, such as the sigmoid function, the derivatives are almost always public knowledge. Unless you're inventing a function, you won't need to calculate derivatives by hand, but it can still be a good exercise. The sigmoid function's derivative solves to $\sigma_{i,j}(1-\sigma_{i,j})$. If you would like to leverage this fact without diving into the mathematical derivation, feel free to skip to the next section.

Sigmoid Function Derivative

Let's define the derivative of the **Sigmoid** function with respect to its input:

$$\sigma_{i,j} = \frac{1}{1+e^{-z_{i,j}}} \quad \rightarrow \quad \sigma'_{i,j} = \frac{d}{dz_{i,j}}[\frac{1}{1+e^{-z_{i,j}}}]$$

At this point, we might start calculating the derivative of the division operation, but, since the numerator contains just the value *1*, the whole fraction is effectively just a reciprocal of its denominator and can be represented as its negative power:

$$\frac{1}{x} = x^{-1}$$

It's easier to calculate the derivative of the power operation than the derivative of the division operation, so let's update our equation to follow this:

$$\frac{d}{dz_{i,j}}(1+e^{-z_{i,j}})^{-1} =$$

Now, we can calculate the derivative of the expression raised to the power of the *-1*, which equals this exponent multiplied by the expression itself, raised to the power lowered by *1*. Then, following the chain rule, we have to calculate the derivative of the expression itself:

$$= -1 \cdot (1+e^{-z_{i,j}})^{-1-1} \cdot \frac{d}{dz_{i,j}}(1+e^{-z_{i,j}}) =$$

As we already learned, the derivative of the sum operation is the sum of derivatives:

$$= -(1+e^{-z_{i,j}})^{-2} \cdot (\frac{d}{dz_{i,j}}1 + \frac{d}{dz_{i,j}}e^{-z_{i,j}}) =$$

The derivative of *1* with respect to $z_{i,j}$ equals *0*, as the derivative of a constant is always *0*. The

derivative of the constant *e* raised to the power $-z_{i,j}$ equals this value multiplied by the derivative of the exponent:

$$= -(1 + e^{-z_{i,j}})^{-2} \cdot (0 + e^{-z_{i,j}} \cdot \frac{d}{dz_{i,j}}[-z_{i,j}]) =$$

The derivative of the $-z_{i,j}$ with respect to $z_{i,j}$ equals *-1* as *-1* is a constant and can be moved outside of the derivative, leaving us with the derivative of $z_{i,j}$ with respect to $z_{i,j}$ which, as we know, equals *1*:

$$= -(1+e^{-z_{i,j}})^{-2} \cdot (e^{-z_{i,j}} \cdot (-1 \cdot \frac{d}{dz_{i,j}} z_{i,j})) = -(1+e^{-z_{i,j}})^{-2} \cdot (e^{-z_{i,j}} \cdot (-1)) =$$

Now we can move the minus sign outside of the parentheses and cancel out the other minus:

$$= -(1 + e^{-z_{i,j}})^{-2} \cdot (-e^{-z_{i,j}}) = (1 + e^{-z_{i,j}})^{-2} \cdot e^{-z_{i,j}} =$$

Let's rewrite the resulting equation — the expression raised to the power of *-2* can be written as its reciprocal raised to the power of *2*, then the multiplier (the value we multiply by) from the equation can become the numerator of the resulting fraction:

$$= \frac{e^{-z_{i,j}}}{(1 + e^{-z_{i,j}})^2} =$$

The denominator of this fraction can be written as the multiplication of the expression by itself instead of raising it to the power of *2*:

$$= \frac{e^{-z_{i,j}}}{(1 + e^{-z_{i,j}})(1 + e^{-z_{i,j}})} =$$

Now we can split this fraction into two separate ones — one containing *1* in the numerator and the other one *e* to the power of $-z_{i,j}$, both having each of the expressions that are separated by the multiplication operator in the denominator in their respective denominators. We can do this as we are performing the multiplication operation between both fractions:

$$= \frac{1}{1 + e^{-z_{i,j}}} \cdot \frac{e^{-z_{i,j}}}{1 + e^{-z_{i,j}}} =$$

If you remember the equation of the sigmoid function, you might already see where we are going with this — the multiplicand (the value that is being multiplied by the multiplier) is the equation of the sigmoid function. Let's work on this equation further — it'd be ideal if the numerator of the

multiplicator could be represented as some sort of equation containing the sigmoid function's equation as well. What we can do is add *1* and remove *1* from it as it won't change its value:

$$= \frac{1}{1+e^{-z_{i,j}}} \cdot \frac{1+e^{-z_{i,j}}-1}{1+e^{-z_{i,j}}} =$$

What this allows us to do is split the multiplicator into two separate fractions by the minus sign in the multiplicator:

$$= \frac{1}{1+e^{-z_{i,j}}} \cdot (\frac{1+e^{-z_{i,j}}}{1+e^{-z_{i,j}}} - \frac{1}{1+e^{-z_{i,j}}}) =$$

The minuend (the value we are subtracting from) of the multiplicator equals *1* as the numerator, and the denominator of the fraction are equal, and the subtrahend (the value we are subtracting from the minuend) is actually the equation of the sigmoid function as well:

$$= \frac{1}{1+e^{-z_{i,j}}} \cdot (1 - \frac{1}{1+e^{-z_{i,j}}}) = \sigma_{i,j} \cdot (1 - \sigma_{i,j})$$

It turns out that the derivative of the sigmoid function equals this function multiplied by the difference of *1* and this function as well. That allows us to easily write this derivative in the code.

Full solution:

$$\sigma_{i,j} = \frac{1}{1+e^{-z_{i,j}}} \quad \rightarrow \quad \sigma'_{i,j} = \frac{d}{dz_{i,j}}[\frac{1}{1+e^{-z_{i,j}}}] = \frac{d}{dz_{i,j}}(1+e^{-z_{i,j}})^{-1} =$$

$$= -1 \cdot (1+e^{-z_{i,j}})^{-1-1} \cdot \frac{d}{dz_{i,j}}(1+e^{-z_{i,j}}) = -(1+e^{-z_{i,j}})^{-2} \cdot (\frac{d}{dz_{i,j}}1 + \frac{d}{dz_{i,j}}e^{-z_{i,j}}) =$$

$$= -(1+e^{-z_{i,j}})^{-2} \cdot (0 + e^{-z_{i,j}} \cdot \frac{d}{dz_{i,j}}[-z_{i,j}]) =$$

$$= -(1+e^{-z_{i,j}})^{-2} \cdot (e^{-z_{i,j}} \cdot (-1 \cdot \frac{d}{dz_{i,j}}z_{i,j})) = -(1+e^{-z_{i,j}})^{-2} \cdot (e^{-z_{i,j}} \cdot (-1)) =$$

$$= -(1+e^{-z_{i,j}})^{-2} \cdot (-e^{-z_{i,j}}) = (1+e^{-z_{i,j}})^{-2} \cdot e^{-z_{i,j}} =$$

$$= \frac{e^{-z_{i,j}}}{(1+e^{-z_{i,j}})^2} = \frac{e^{-z_{i,j}}}{(1+e^{-z_{i,j}})(1+e^{-z_{i,j}})} = \frac{1}{1+e^{-z_{i,j}}} \cdot \frac{e^{-z_{i,j}}}{1+e^{-z_{i,j}}} =$$

$$= \frac{1}{1+e^{-z_{i,j}}} \cdot \frac{1+e^{-z_{i,j}}-1}{1+e^{-z_{i,j}}} == \frac{1}{1+e^{-z_{i,j}}} \cdot (\frac{1+e^{-z_{i,j}}}{1+e^{-z_{i,j}}} - \frac{1}{1+e^{-z_{i,j}}}) =$$

$$= \frac{1}{1+e^{-z_{i,j}}} \cdot (1 - \frac{1}{1+e^{-z_{i,j}}}) = \sigma_{i,j} \cdot (1 - \sigma_{i,j})$$

Sigmoid Function Code

As with other activation functions, we'll write a forward pass method and a backward pass method. For the forward pass, we'll take the inputs and apply the sigmoid function. For the backward pass, we'll leverage the sigmoid function's derivative, which, as we figured out during derivation of the sigmoid function's derivative, equals the sigmoid output from the forward pass multiplied by the difference of 1 and this output.

```
ſi SĨÍİIJĨĤ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçSĨÍİIJĨĤf

    ſi FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ſi SĞĿĦ ĨĮĴŁĻ ĞĮĤ ĢĞĬĢŁĬĞĻĦwĽĞĿĦ IJŁĻĴŁĻ
        ſi IJÌ ĻĤĦ ĽĨÍİIJĨĤ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĝ w âĝ κ ĮĴgĦŃĴâhĨĮĴŁĻĽãã

    ſi BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĿĞĬŁĦĽãf
        ſi DĦĹĨĿĞĻĨĿĦ h ĢĞĬĢŁĬĞĻĦĽ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĻĤĦ ĽĨÍİIJĨĤ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĤĿĞĬŁĦĽ ắ âĝ h ĽĦĬÌgIJŁĻĴŁĻã ắ ĽĦĬÌgIJŁĻĴŁĻ
```

Now that we have the new activation function, we need to code our new calculation for the binary cross-entropy loss.

Binary Cross-Entropy Loss

To calculate binary cross-entropy loss, we will continue to use the negative log concept from categorical cross-entropy loss. Rather than only calculating this on the target class, we will sum the log-likelihoods of the correct and incorrect classes for each neuron separately. Because class values are either *0* or *1*, we can simplify the incorrect class to be *1-correct class* as this inverts the value. We can then calculate the negative log-likelihood of the correct and incorrect classes, adding them together. We are presenting two forms of the equation — the first is following the description just given, then the optimized version differentiating only in the minus signs being moved over and redundant parentheses removed:

$$L_{i,j} = (y_{i,j})(-log(\hat{y}_{i,j})) + (1 - y_{i,j})(-log(1 - \hat{y}_{i,j})) =$$

$$= -y_{i,j} \cdot log(\hat{y}_{i,j}) - (1 - y_{i,j}) \cdot log(1 - \hat{y}_{i,j})$$

In code, this will start as (but will be modified shortly, so do not commit this to your codebase yet):

```
ĽĞİĴĬĦçĬIJĽĽĦĽ ł hâÑçĻĹŁĦ ắ ĮĴgĬIJÍâÑçĴĹĦĤã κ
                ârĝ h ÑçĻĹŁĦã ắ ĮĴgĬIJÍârĝ h ÑçĴĹĦĤãã
```

Since a model can contain multiple binary outputs, and each of them, unlike in the cross-entropy loss, outputs its own prediction, loss calculated on a single output is going to be a vector of losses containing one value for each output. What we need is a sample loss and, to achieve that, we need to calculate a mean of all of these losses from a single sample:

$$L_i = \frac{1}{J} \sum_j L_{i,j}$$

Where index *i* means the current sample, the index *j* means the current output in this sample, and the *J* means the number of outputs. Since we are operating on a set of samples (the output is an array containing the set of loss vectors), we can use NumPy to perform this operation on a single call:

```
ĽĞİĴĬĦçĬIJĽĽĦĽ ł ĮĴgİĦĞĮâĽĞİĴĬĦçĬIJĽĽĦĽd ĞŇĨĽłhĝã
```

The last parameter, ĞŇĨĽłhĝ, informs NumPy to calculate the mean value along the last dimension. To make it easier to visualize, let's use a simple example. Assume that this is an output of the model containing 3 neurons in the output layer, and it's passed through the binary cross-entropy loss function:

```
IJłĻĴłĻĽ ł ĮĴgĞĹĹĞÑâäägd  ğd  ġād
                    äğd  ǵd  ħād
                    äfld  ĥd  ĝflād
                    äĝĝd ĝğd ĝġād
                    äĥd  ĝfld ĝĥāāã
```

These numbers are completely made up for this example. We want to take each of the output vectors, äĝd ğd ġā for example, and calculate a mean value from the numbers they hold, putting the result on the output vector. We then want to repeat this for the other vectors and return the resulting vector, which will be a one-dimensional array. Using NumPy:

```
ĮĴgİĦĞĮâIJłĻĴłĻĽd ĞŇĨĽłhĝã


ńńń
ĞĹĹĞÑâä ğgd  ǵgd  ĥgd ĝğgd ĝflgāã
```

If we calculate the mean value of the first output, it's indeed *2*, the mean value of the second output is indeed *4*, and so on.

We are also going to inherit from the **Loss** class, so the overall loss calculation will be handled by the ĢĞĬĢŁĬĞĻĦ method that we already created for the categorical cross-entropy loss class.

Binary Cross-Entropy Loss Derivative

To calculate the gradient from here, we already know that the derivative for the natural logarithm is *1/x* and that the derivative of *1-x* is *-1*. In simplified form, this gives us *-(y_true / y + (1 - y_true) / (1 - y)) · (-1)*.

To calculate the partial derivative of this loss function with respect to the predicted input, we'll use the latter version of the loss equation. It doesn't really matter in this case which one we use:

$$\frac{\partial L_{i,j}}{\partial \hat{y}_{i,j}} = \frac{\partial}{\partial \hat{y}_{i,j}}[-y_{i,j} \cdot log(\hat{y}_{i,j}) - (1 - y_{i,j}) \cdot log(1 - \hat{y}_{i,j})] =$$

The expression that we have to calculate the partial derivative of consists of two sub-expressions, which are components of the sum operation. We can write that as the sum of derivatives:

$$= \frac{\partial}{\partial \hat{y}_{i,j}}[-y_{i,j} \cdot log(\hat{y}_{i,j})] + \frac{\partial}{\partial \hat{y}_{i,j}}[-(1 - y_{i,j}) \cdot log(1 - \hat{y}_{i,j})] =$$

Both components contain $y_{i,j}$ (the target value) inside of their derivatives, which are the constants that we are deriving with respect to $y\text{-}hat_{i,j}$ (the predicted value, which is a different variable), so we can move them outside of the derivative along with the other constants and minus sign:

$$= -y_{i,j} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} log(\hat{y}_{i,j}) - (1 - y_{i,j}) \cdot \frac{\partial}{\partial \hat{y}_{i,j}} log(1 - \hat{y}_{i,j}) =$$

Now, like in the **Categorical Cross-Entropy** loss' derivative, we have to calculate the derivative of the logarithmic function, which equals the reciprocal of its parameter multiplied (following the chain rule) by the derivative of this parameter. Let's apply that to both of the partial derivatives:

$$= -y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j} - (1 - y_{i,j}) \cdot \frac{1}{1 - \hat{y}_{i,j}} \cdot \frac{\partial}{\partial \hat{y}_{i,j}}[1 - \hat{y}_{i,j}] =$$

Now the first partial derivative equals *1*, since the value we derive, and the value we derive with respect to, are the same values. The second partial derivative can be written as the difference of

the derivatives:

$$= -y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot 1 - (1 - y_{i,j}) \cdot \frac{1}{1 - \hat{y}_{i,j}} \cdot (\frac{\partial}{\partial \hat{y}_{i,j}} 1 - \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j}) =$$

From the two new derivatives, the first one equals *0* as the derivative of the constant always equals *0*, then the second derivative equals *1* as the value we derive, and the value we derive with respect to, are the same values:

$$= -y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot 1 - (1 - y_{i,j}) \cdot \frac{1}{1 - \hat{y}_{i,j}} \cdot (0 - 1) =$$

We can finally clean up to get the resulting equation:

$$= -\frac{y_{i,j}}{\hat{y}_{i,j}} + \frac{1 - y_{i,j}}{1 - \hat{y}_{i,j}} = -(\frac{y_{i,j}}{\hat{y}_{i,j}} - \frac{1 - y_{i,j}}{1 - \hat{y}_{i,j}})$$

The partial derivative of the **Binary Cross-Entropy** loss solves to a pretty simple equation that will be easy to implement in code.

Full solution:

$$\frac{\partial L_{i,j}}{\partial \hat{y}_{i,j}} = \frac{\partial}{\partial \hat{y}_{i,j}} [-y_{i,j} \cdot log(\hat{y}_{i,j}) - (1 - y_{i,j}) \cdot log(1 - \hat{y}_{i,j})] =$$

$$= \frac{\partial}{\partial \hat{y}_{i,j}} [-y_{i,j} \cdot log(\hat{y}_{i,j})] + \frac{\partial}{\partial \hat{y}_{i,j}} [-(1 - y_{i,j}) \cdot log(1 - \hat{y}_{i,j})] =$$

$$= -y_{i,j} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} log(\hat{y}_{i,j}) - (1 - y_{i,j}) \cdot \frac{\partial}{\partial \hat{y}_{i,j}} log(1 - \hat{y}_{i,j}) =$$

$$= -y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j} - (1 - y_{i,j}) \cdot \frac{1}{1 - \hat{y}_{i,j}} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} [1 - \hat{y}_{i,j}] =$$

$$= -y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot 1 - (1 - y_{i,j}) \cdot \frac{1}{1 - \hat{y}_{i,j}} \cdot (\frac{\partial}{\partial \hat{y}_{i,j}} 1 - \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j}) =$$

$$= -y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot 1 - (1 - y_{i,j}) \cdot \frac{1}{1 - \hat{y}_{i,j}} \cdot (0 - 1) =$$

$$= -\frac{y_{i,j}}{\hat{y}_{i,j}} + \frac{1 - y_{i,j}}{1 - \hat{y}_{i,j}} = -(\frac{y_{i,j}}{\hat{y}_{i,j}} - \frac{1 - y_{i,j}}{1 - \hat{y}_{i,j}})$$

This partial derivative is a derivative of the single output's loss and, with any type of output, we always need to calculate it with respect to a sample loss, not an atomic output loss, since we have to calculate the mean value of all output losses in a sample to form a *sample loss* during the forward pass:

$$L_i = \frac{1}{J}\sum_j L_{i,j}$$

For backpropagation, we have to calculate the partial derivative of the *sample loss* with respect to each input:

$$\frac{\partial L_i}{\partial \hat{y}_{i,j}} = \frac{\partial L_i}{\partial L_{i,j}} \cdot \frac{\partial L_{ij}}{\partial \hat{y}_{i,j}}$$

We have just calculated the second derivative, the partial derivative of the single output loss with respect to the related prediction. We have to calculate the partial derivative of the sample loss with respect to the single output loss:

$$\frac{\partial L_i}{\partial L_{i,j}} = \frac{\partial}{\partial L_{i,j}}[\frac{1}{J}\sum_j L_{i,j}] =$$

1 divided by *J* (the number of outputs), is a constant and can be moved outside of the derivative. Since we are calculating the derivative with respect to a given output, *j*, the sum of one element equals this element:

$$= \frac{1}{J} \cdot \frac{\partial}{\partial L_{i,j}} L_{i,j} =$$

The remaining derivative equals *1* as the derivative of a variable with respect to the same variable equals *1*.

$$= \frac{1}{J} \cdot 1 = \frac{1}{J}$$

Full solution:

$$\frac{\partial L_i}{\partial L_{i,j}} = \frac{\partial}{\partial L_{i,j}}[\frac{1}{J}\sum_j L_{i,j}] = \frac{1}{J} \cdot \frac{\partial}{\partial L_{i,j}} L_{i,j} = \frac{1}{J} \cdot 1 = \frac{1}{J}$$

Now we can update the equation of the partial derivative of a sample loss with respect to a single output loss by applying the chain rule:

$$\frac{\partial L_i}{\partial \hat{y}_{i,j}} = \frac{\partial L_i}{\partial L_{i,j}} \cdot \frac{\partial L_{ij}}{\partial \hat{y}_{i,j}} = \frac{1}{J} \cdot (-(\frac{y_{i,j}}{\hat{y}_{i,j}} - \frac{1 - y_{i,j}}{1 - \hat{y}_{i,j}})) = -\frac{1}{J} \cdot (\frac{y_{i,j}}{\hat{y}_{i,j}} - \frac{1 - y_{i,j}}{1 - \hat{y}_{i,j}}) =$$

We have to perform this normalization since each output returns its own derivative, and without normalization, each additional input will raise gradients and require changing other hyperparameters, including the learning rate.

Binary Cross-Entropy Code

In our code, this will be:

```
# NłİĠĦĹ IJÌ ĽĞİĴĬĦĽ
ĽĞİĴĬĦĽ ł ĬĦĮâĤŁĞĬłĦĽã
# NłİĠĦĹ IJÌ IJłĻĴłĻĽ ĨĮ ĦŁĦĹÑ ĽĞİĴĬĦ
# WĦďĬĬ łĽĦ ĻÎĦ ÌĨĹĽĻ ĽĞİĴĬĦ ĻIJ ĢIJłĮĻ ĻÎĦİ
IJłĻĴłĻĽ ł ĬĦĮâĤŁĞĬłĦĽäflãã

# CĞĬĢłĬĞĻĦ ÍĹĞĤĨĦĮĻ
ĽĦĬÌgĤĨĮĴłĻĽ ł hâÑçĻĹłĦ w ĢĬĨĴĴĦĤçĤŁĞĬłĦĽ h
                  âĝ h ÑçĻĹłĦã w âĝ h ĢĬĨĴĴĦĤçĤŁĞĬłĦĽãã w IJłĻĴłĻĽ
```

Similar to what we did in the categorical cross-entropy loss, we need to normalize gradient so it'll become invariant to the number of samples we calculate it for:

```
# NIJĹİĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
ĽĦĬÌgĤĨĮĴłĻĽ ł ĽĦĬÌgĤĨĮĴłĻĽ w ĽĞİĴĬĦĽ
```

Finally, we need to address the numerical instability of the logarithmic function. The sigmoid activation can return a value in the range of *0* to *1* (inclusive), but the *log(0)* presents a slight issue due to how it's calculated and will return *negative infinity*. This alone isn't necessarily a big deal, but any list with *-inf* in it will have a mean of *-inf*, which is the same for any list with positive infinity averaging to infinity.

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĮĴgĬIJÍâfãã


ńńń
ççİĞĨĮççfĝf RŁĮĻĨİĦWĞĹĮĨĮÍf ĤĨĿĨĤĦ ĠÑ ŅĦĹIJ ĦĮĢIJŁĮĻĦĹĦĤ ĨĮ ĬIJÍ
hĨĮÌ


ĴĹĨĮĻâĮĴgİĦĞĮâäĥd ğd ǵd ĮĴgĬIJÍâfãããã


ńńń
hĨĮÌ
```

This is a similar issue to the one we discussed earlier regarding categorical cross-entropy loss in chapter 5. To prevent this issue, we'll add clipping on the batch of values:

```
fi CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦĿĦĮĻ ĤĨĿĨĽĨIJĮ ĠÑ fl
fi CĬĨĴ ĠIJĻÎ ĽĨĤĦĽ ĻIJ ĮIJĻ ĤĹĞÍ İĦĞĮ ĻIJŃĞĹĤĽ ĞĮÑ ĿĞĬŁĦ
ÑçĴĹĦĤçĢĬĨĴĴĦĤ ł ĮĴgĢĬĨĴâÑçĴĹĦĤd ĝĦhìd ĝ h ĝĦhìã
```

We now will use these clipped values for the forward pass, rather than the originals:

```
fi CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦhŃĨĽĦ ĬIJĹĽ
ĽĞİĴĬĦçĬIJĹĽĦĽ ł hâÑçĻĹŁĦ ấ ĮĴgĬIJÍâÑçĴĹĦĤçĢĬĨĴĴĦĤã κ
                  âĝ h ÑçĻĹŁĦã ấ ĮĴgĬIJÍâĝ h ÑçĴĹĦĤçĢĬĨĴĴĦĤãã
```

As we perform the division operation during the derivative calculation, the gradient passed in may contain both values, *0* and *1*. Either of these values will cause a problem in either the `ÑçĻĹŁĦ w ĤĿĞĬŁĦĽ` or `âĝ h ÑçĻĹŁĦã w âĝ h ĤĿĞĬŁĦĽã` parts respectively (*0* in the first and *1-1=0* in the second case will also cause division by *0*), so we need to clip this gradient as well:

```
fi CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦĿĦĮĻ ĤĨĿĨĽĨIJĮ ĠÑ fl
fi CĬĨĴ ĠIJĻÎ ĽĨĤĦĽ ĻIJ ĮIJĻ ĤĹĞÍ İĦĞĮ ĻIJŃĞĹĤĽ ĞĮÑ ĿĞĬŁĦ
ĢĬĨĴĴĦĤçĤĿĞĬŁĦĽ ł ĮĴgĢĬĨĴâĤĿĞĬŁĦĽd ĝĦhìd ĝ h ĝĦhìã
```

Now, similar to the forward pass, we can use these clipped values:

```
fi CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
ĽĦĬÌgĤĨĮĴŁĻĽ ł hâÑçĻĹŁĦ w ĢĬĨĴĴĦĤçĤĿĞĬŁĦĽ h
                  âĝ h ÑçĻĹŁĦã w âĝ h ĢĬĨĴĴĦĤçĤĿĞĬŁĦĽãã w IJŁĻĴŁĻĽ
```

The full code now for the binary cross-entropy:

```
ƒ BĨĮĞĹÑ ĢĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ
ĢĬĞĽĹ LIJĹĽçBĨĮĞĹÑCĹIJĹĽĦĮĻĹIJĴÑâLIJĹĽãf

    ƒ FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        ƒ CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ĤĨŁĨĹĨIJĮ ĠÑ ƒ
        ƒ CĬĨĴ ĠIJĻĤ ĹĨĤĦĹ ĻIJ ĮIJĻ ĤĹĞĬ ĬĦĞĮ ĻIJŃĞĹĤĽ ĞĮÑ ŁĞĬŁĦ
        ÑçĴĹĦĤçĢĬĨĴĴĦĤ ł ĮĴgĢĬĨĴâÑçĴĹĦĤd ĝĦhìd ĝ h ĝĦhìã

        ƒ CĞĬĢŁĬĞĻĦ ĹĞĬĴĬĦhŃĨĹĦ ĬIJĹĽ
        ĹĞĬĴĬĦçĬIJĹĽĦĹ ł hâÑçĻĹŁĦ ắ ĮĴgĬIJÍâÑçĴĹĦĤçĢĬĨĴĴĦĤã ĸ
                                  âĝ h ÑçĻĹŁĦã ắ ĮĴgĬIJÍâĝ h ÑçĴĹĦĤçĢĬĨĴĴĦĤãã
        ĹĞĬĴĬĦçĬIJĹĽĦĹ ł ĮĴgĬĦĞĮâĹĞĬĴĬĦçĬIJĹĽĦĹd ĞŃĨĹłhĝã

        ƒ RĦĻŁĹĮ ĬIJĹĽĦĹ
        ĹĦĻŁĹĮ ĹĞĬĴĬĦçĬIJĹĽĦĹ

    ƒ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ĤĹĞĬŁĦĹd ÑçĻĹŁĦãf

        ƒ NŁĬĠĦĹ IJÌ ĹĞĬĴĬĦĹ
        ĹĞĬĴĬĦĹ ł ĬĦĮâĤĹĞĬŁĦĹã
        ƒ NŁĬĠĦĹ IJÌ IJŁĻĴŁĻĹ ĨĮ ĦĹĦĹÑ ĹĞĬĴĬĦ
        ƒ WĦďĬĬ ŁĹĦ ĻĨĦ ÌĨĹĹĻ ĹĞĬĴĬĦ ĻIJ ĢIJŁĮĻ ĻĨĦĬ
        IJŁĻĴŁĻĹ ł ĬĦĮâĤĹĞĬŁĦĹäflãã

        ƒ CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ĤĨŁĨĹĨIJĮ ĠÑ ƒ
        ƒ CĬĨĴ ĠIJĻĤ ĹĨĤĦĹ ĻIJ ĮIJĻ ĤĹĞĬ ĬĦĞĮ ĻIJŃĞĹĤĽ ĞĮÑ ŁĞĬŁĦ
        ĢĬĨĴĴĦĤçĤĹĞĬŁĦĹ ł ĮĴgĢĬĨĴâĤĹĞĬŁĦĹd ĝĦhìd ĝ h ĝĦhìã

        ƒ CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł hâÑçĻĹŁĦ w ĢĬĨĴĴĦĤçĤĹĞĬŁĦĹ h
                                 âĝ h ÑçĻĹŁĦã w âĝ h ĢĬĨĴĴĦĤçĤĹĞĬŁĦĹãã w IJŁĻĴŁĻĹ
        ƒ NIJĹĬĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĹĦĬÌgĤĨĮĴŁĻĹ w ĹĞĬĴĬĦĹ
```

Now that we have this new activation function and loss calculation, we'll make edits to our existing softmax classifier to implement the binary logistic regression model.

Implementing Binary Logistic Regression and Binary Cross-Entropy Loss

With these new classes, our code changes will be in the execution of actual code (instead of modifying the classes). The first change is to make the `ĽĴĨĹĞĬçĤĞĻĞ` object output 2 classes, rather than 3, like so:

```
ſı CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝſſſd ĢĬĞĽĽĦĽłğã
```

Next, we'll reshape our labels, as they're not sparse anymore. They're binary, *0* or *1*:

```
ſı RĦĽÎĞĴĦ ĬĞĠĦĬĽ ĻIJ ĠĦ Ğ ĬĨĽĻ IJÌ ĬĨĽĻĽ
ſı IĮĮĦĹ ĬĨĽĻ ĢIJĮĻĞĨĮĽ IJĮĦ IJŁĻĴŁĻ âĦĨĻÎĦĹ ſ IJĹ ĝã
ſı ĴĦĹ ĦĞĢÎ IJŁĻĴŁĻ ĮĦŁĹIJĮd ĝ ĨĮ ĻÎĨĽ ĢĞĽĦ
Ñ ł ÑgĹĦĽÎĞĴĦâhĝd ĝã
```

Consider the difference here. Initially, the *y* output from the `ĽĴĨĹĞĬçĤĞĻĞ` function would look something like:

```
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝſſſd ĢĬĞĽĽĦĽłğã
ĴĹĨĮĻâÑäfĥãã


ńńń
äſ ſ ſ ſ ſã
```

Then we reshape it here for binary logistic regression:

```
Ñ ł ÑgĹĦĽÎĞĴĦâhĝd ĝã
ĴĹĨĮĻâÑäfĥãã


ńńń
ääſã
 äſã
 äſã
 äſã
 äſãã
```

Why have we done this? Initially, with the softmax classifier, the values from ĽĴĨĹĞĬçĤĞĻĞ could be used directly as the target labels, as they contain the correct class labels in numerical form — an index of the correct class, where each neuron in the output layer is a separate class, for example äfld ĝd ĝd fld ĝā. In this case, however, we're trying to represent some binary outputs, where each neuron represents 2 possible classes on its own. For the example we're currently working on, we have a single output neuron so the output from our neural network should be a tensor (array), containing one value, of a target value of either *0* or *1*, for example, ääflād äĝād äĝād äflād äĝāā. The gĹĦĽĨĞĴĦâhĝd ĝã means to reshape the data into 2 dimensions, where the second dimension contains a single element, and the first dimension contains how many elements the result will contain (hĝ) following other conditions. You are allowed to use hĝ only once in a shape with NumPy, letting you have that dimension be variable. Thanks to this ability, we do not always need the same number of samples every time, and NumPy can handle the calculation for us. In the case above, they're all *0* because the ĽĴĨĹĞĬçĤĞĻĞ function makes the dataset one class at a time, starting with *0*. We will also need to reshape the y-testing data in the same way.

Let's create our layers and use the appropriate activation functions:

```
# CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĝflfld ğã

# RĦĽĨĞĴĦ ĬĞĠĦĬĽ ĻIJ ĠĦ Ğ ĬĨĽĻ IJÌ ĬĨĽĻĽ
# IĮĮĦĹ ĬĨĽĻ ĢIJĮĻĞĨĮĽ IJĮĦ IJŁĻĴŁĻ âĦĨĻĨĦĹ fl IJĹ ĝã
# ĴĦĹ ĦĞĢĨ IJŁĻĴŁĻ ĮĦŁĹIJĮd ĝ ĨĮ ĻĨĨĽ ĢĞĽĦ
Ñ ł ÑgĹĦĽĨĞĴĦâhĝd ĝã

# CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻĨ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮĤ ħĝ IJŁĻĴŁĻ ŁĠĬŁĦĽ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħĝd ŃĦĨÍĨĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦħĝd
                                ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦħĝã

# CĹĦĞĻĦ RĦLU ĞĢĻĨŁĠĻĨIJĮ âĻIJ ĠĦ ŁĽĦĤ ŃĨĻĨ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨŁĠĻĨIJĮĝ ł AĢĻĨŁĠĻĨIJĮçRĦLUâã

# CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻĨ ħĝ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJŁĻĴŁĻ
# IJÌ ĴĹĦŁĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ĝ IJŁĻĴŁĻ ŁĠĬŁĦ
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâħĝd ĝã

# CĹĦĞĻĦ SĨÍĬIJĨĤ ĞĢĻĨŁĠĻĨIJĮf
ĞĢĻĨŁĠĻĨIJĮğ ł AĢĻĨŁĠĻĨIJĮçSĨÍĬIJĨĤâã
```

Notice that we're still using the rectified linear activation for the hidden layer. The hidden layer activation functions don't necessarily need to change, even though we're effectively building a different type of classifier. You should also notice that because this is now a binary classifier, the ĤĦĮĽĦğ object has only 1 output. Its output represents exactly 2 classes (*0* or *1*) being mapped to one neuron. We can now select a loss function and optimizer. For the **Adam** optimizer settings,

we are going to use the default learning rate and the decaying of *5e-7*:

```
ƒı CĹĦĞĻĦ ĬIJĹĽ ÌŁĮĢĻĨIJĮ
ĬIJĹĽçÌŁĮĢĻĨIJĮ ł LIJĹĽçBĨĮĞĹÑCĹIJĹĽĦĮĻĹIJĴÑâã

ƒı CĹĦĞĻĦ IJĴĻĨĬĨĨĨĨĨĦĹ
IJĴĻĨĬĨĨĨĨĦĹ ł OĴĻĨĬĨĨĨĦĹçAĤĞİâĤĦĢĞÑłĥĦhìã
```

While we require a different calculation for loss (since we use a different activation function for the output layer), we can still use the same optimizer as in the softmax classifier. Another small change is how we measure predictions. With probability distributions, we use *argmax* and determine which index is associated with the largest value, which becomes the classification result. With a binary classifier, we are determining if the output is closer to *0* or to *1*. To do this, we simplify the output to:

```
ĴĹĦĤĨĢĻĨIJĮĽ ł âĞĢĻĨŁĞĻĨIJĮğgIJŁĻĴŁĻ ń flgĥã ấ ĝ
```

This results in *True/False* evaluations to the statement that the output is above *05* for all values. *True* and *False*, when treated as numbers, are *1* and *0*, respectively. For example, if we execute: *ĨĮĻ*âTĹŁĦã, the result will be ĝ and *ĨĮĻ*âFĞĬĽĦã will be fl. If we want to convert a list of *True/False* boolean values to numbers, we can't just wrap the list in *ĨĮĻ*ââã. However, we *can* perform math operations directly on an array of boolean values and return the arithmetic answer. For example, we can run:

```
ĨİĴIJĹĻ IłİĴÑ ĞĽ IĴ
Ğ ł IĴgĞĹĹĞÑâäTĹŁĦd FĞĬĽĦd TĹŁĦāã
ĴĹĨĮĻâĞã

ńńń
ä TĹŁĦ FĞĬĽĦ  TĹŁĦā
```

And then:

```
Ġ ł Ğấĝ
ĴĹĨĮĻâĠã

ńńń
äĝ fl ĝā
```

Thus, to evaluate predictive accuracy, we can do the following in our code:

```
ĴĹĦĤĨĢĻĨIJĮĽ ł âĞĢĻĨŁĞĻĨIJĮğgIJŁĻĴŁĻ ń flgĥã ấ ĝ
ĞĢĢŁĹĞĢÑ ł IĴgİĦĞĮâĴĹĦĤĨĢĻĨIJĮĽłłÑçĻĦĹĻã
```

The ấ ĝ multiplication turns an array of boolean True/False values into numerical 1/0 values, respectively. We will need to implement this accuracy calculation for validation data too.

Full code up to this point:

```
ĩİĴIJĹĻ ĮłİĴÑ ĞĽ ĮĴ
ĩİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĩİĴIJĹĻ ĽĴĩĹĞĭçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


ﬁ DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ﬁ LĞÑĦĹ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴłĻĽd ĮçĮĦłĹIJĮĽd
                 ŃĦĨÍÎĻçĹĦÍłĬĞĹĨŅĦĹçĬĝłﬁd ŃĦĨÍÎĻçĹĦÍłĬĞĹĨŅĦĹçĬğłﬁd
                 ĠĨĞĽçĹĦÍłĬĞĹĨŅĦĹçĬĝłﬁd ĠĨĞĽçĹĦÍłĬĞĹĨŅĦĹçĬğłﬁãf
        ﬁ IĮĨĻĨĞĬĨŅĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍÎĻĽ ł ﬁgﬁĝ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĮçĨĮĴłĻĽd ĮçĮĦłĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦłĹIJĮĽãã
        ﬁ SĦĻ ĹĦÍłĬĞĹĨŅĞĻĨIJĮ ĽĻĹĦĮÍĻÎ
        ĽĦĬÌgŃĦĨÍÎĻçĹĦÍłĬĞĹĨŅĦĹçĬĝ ł ŃĦĨÍÎĻçĹĦÍłĬĞĹĨŅĦĹçĬĝ
        ĽĦĬÌgŃĦĨÍÎĻçĹĦÍłĬĞĹĨŅĦĹçĬğ ł ŃĦĨÍÎĻçĹĦÍłĬĞĹĨŅĦĹçĬğ
        ĽĦĬÌgĠĨĞĽçĹĦÍłĬĞĹĨŅĦĹçĬĝ ł ĠĨĞĽçĹĦÍłĬĞĹĨŅĦĹçĬĝ
        ĽĦĬÌgĠĨĞĽçĹĦÍłĬĞĹĨŅĦĹçĬğ ł ĠĨĞĽçĹĦÍłĬĞĹĨŅĦĹçĬğ

    ﬁ FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴłĻĽãf
        ﬁ RĦİĦİĠĦĹ ĨĮĴłĻ ŁĞĬłĦĽ
        ĽĦĬÌgĨĮĴłĻĽ ł ĨĮĴłĻĽ
        ﬁ CĞĬĢłĬĞĻĦ IJłĻĴłĻ ŁĞĬłĦĽ ÌĹIJİ ĨĮĴłĻĽd ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgIJłĻĴłĻ ł ĮĴgĤIJĻâĨĮĴłĻĽd ĽĦĬÌgŃĦĨÍÎĻĽã ĸ ĽĦĬÌgĠĨĞĽĦĽ

    ﬁ BĞĢĨŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤŁĞĬłĦĽãf
        ﬁ GĹĞĤĨĦĮĻĽ IJĮ ĴĞĹĞĬĦĻĦĹĽ
        ĽĦĬÌgĤŃĦĨÍÎĻĽ ł ĮĴgĤIJĻâĽĦĬÌgĨĮĴłĻĽgTd ĤŁĞĬłĦĽã
        ĽĦĬÌgĤĠĨĞĽĦĽ ł ĮĴgĽłĬâĤŁĞĬłĦĽd ĞŃĨĽłﬁd ĪĦĦĴĤĨĬĽłTĹłĦã
```

```
        ńı GĹĞĤĨĦĮĻĽ ĲĮ ĹĦÍŁĬĞĹĨŅĞĻĨĲĮ
        ńı Lĝ ĲĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ĤLĝ ł ĮĴgĲĮĦĽçĬĨĨĦâĽĦĬÌgŃĦĨÍÎĻĽã
            ĤLĝäĽĦĬÌgŃĦĨÍÎĻĽ l· flā ł hĝ
            ĽĦĬÌgĤŃĦĨÍÎĻĽ κł ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ő ĤLĝ
        ńı Lğ ĲĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĽĦĬÌgĤŃĦĨÍÎĻĽ κł ğ ő ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                ĽĦĬÌgŃĦĨÍÎĻĽ
        ńı Lĝ ĲĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ĤLĝ ł ĮĴgĲĮĦĽçĬĨĨĦâĽĦĬÌgĠĨĞĽĦĽã
            ĤLĝäĽĦĬÌgĠĨĞĽĦĽ l· flā ł hĝ
            ĽĦĬÌgĤĠĨĞĽĦĽ κł ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ő ĤLĝ
        ńı Lğ ĲĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĽĦĬÌgĤĠĨĞĽĦĽ κł ğ ő ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                ĽĦĬÌgĠĨĞĽĦĽ

        ńı GĹĞĤĨĦĮĻ ĲĮ L·ĞĬŁĦĽ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĮĴgĤĲĻâĤL·ĞĬŁĦĽd ĽĦĬÌgŃĦĨÍÎĻĽgTã


ńı DĹĲĴĲŁĻ
ĢĬĞĽĽ LĞÑĦĹçDĹĲĴĲŁĻf

    ńı IĮĨĻ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĹĞĻĦãf
        ńı SĻĲĹĦ ĹĞĻĦd ŃĦ ĨĮL·ĦĹĻ ĨĻ ĞĽ ÌĲĹ ĦŃĞĬĴĬĦ ÌĲĹ ĤĹĲĴĲŁĻ
        ńı ĲÌ flgĝ ŃĦ ĮĦĦĤ ĽŁĢĢĦĽĽ ĹĞĻĦ ĲÌ flgî
        ĽĦĬÌgĹĞĻĦ ł ĝ h ĹĞĻĦ

    ńı FĲĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌĲĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ńı SĞL·Ħ ĨĮĴŁĻ L·ĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ńı GĦĮĦĹĞĻĦ ĞĮĤ ĽĞL·Ħ ĽĢĞĬĦĤ ĬĞĽĨ
        ĽĦĬÌgĠĨĮĞĹÑçĬĞĽĨ ł ĮĴgĹĞĮĤĲĬgĠĨĮĲĬĨĞĬâĝd ĽĦĬÌgĹĞĻĦd
                            ĽĨĢĦłĨĮĴŁĻĽgĽĨĞĴĦã w ĽĦĬÌgĹĞĻĦ
        ńı AĴĴĬÑ ĬĞĽĨ ĻĲ ĲŁĻĴŁĻ L·ĞĬŁĦĽ
        ĽĦĬÌgĲŁĻĴŁĻ ł ĨĮĴŁĻĽ ő ĽĦĬÌgĠĨĮĞĹÑçĬĞĽĨ

    ńı BĞĢĨŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤL·ĞĬŁĦĽãf
        ńı GĹĞĤĨĦĮĻ ĲĮ L·ĞĬŁĦĽ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĤL·ĞĬŁĦĽ ő ĽĦĬÌgĠĨĮĞĹÑçĬĞĽĨ
```

```
# RHLU ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçRHLUf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽãf
        # RHİHİĠHĹ ĨĮĴŁĻ ĿĞĬŁHĽ
        ĽHĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        # CĞĬĢŁĬĞĻH IJŁĻĴŁĻ ĿĞĬŁHĽ ÌĹIJİ ĨĮĴŁĻĽ
        ĽHĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĨİŁİâfld ĨĮĴŁĻĽã

    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ÂĿĞĬŁHĽãf
        # SĨĮĢH ŃH ĮHHĤ ĻIJ ÌIJĤĨÌÑ IJĹĨÍĨĮĞĬ ĿĞĹĨĞĠĬHd
        # ĬHĻďĽ İĞĪH Ğ ĢIJĴÑ IJÌ ĿĞĬŁHĽ ÌĨĹĽĻ
        ĽHĬÌgĤĨĮĴŁĻĽ ł ÂĿĞĬŁHĽgĢIJĴÑâã

        # ZHĹIJ ÍĹĞĤĨHĮĻ ŃÎHĹH ĨĮĴŁĻ ĿĞĬŁHĽ ŃHĹH ĮHÍĞĻĨĿH
        ĽHĬÌgĤĨĮĴŁĻĽäĽHĬÌgĨĮĴŁĻĽ l·ł flã ł fl


# SIJÌĻİĞŃ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŃf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽãf
        # RHİHİĠHĹ ĨĮĴŁĻ ĿĞĬŁHĽ
        ĽHĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ

        # GHĻ ŁĮĮIJĹİĞĬĨŅHĤ ĴĹIJĠĞĠĨĬĨĻĨHĽ
        HŃĴçĿĞĬŁHĽ ł ĮĴgHŃĴâĨĮĴŁĻĽ h ĮĴgİĞŃâĨĮĴŁĻĽd ĞŃĨĽłĝd
                                                ĪHHĴĤĨİĽłTĹŁHãã
        # NIJĹİĞĬĨŅH ĻĨHİ ÌIJĹ HĞĢĨ ĽĞİĴĬH
        ĴĹIJĠĞĠĨĬĨĻĨHĽ ł HŃĴçĿĞĬŁHĽ w ĮĴgĽŁİâHŃĴçĿĞĬŁHĽd ĞŃĨĽłĝd
                                                ĪHHĴĤĨİĽłTĹŁHã

        ĽHĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĨĬĨĻĨHĽ

    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ÂĿĞĬŁHĽãf

        # CĹHĞĻH ŁĮĨĮĨĻĨĞĬĨŅHĤ ĞĹĹĞŃ
        ĽHĬÌgĤĨĮĴŁĻĽ ł ĮĴgHİĴĻŃçĬĨĨHâÂĿĞĬŁHĽã

        # EĮŁİHĹĞĻH IJŁĻĴŁĻĽ ĞĮĤ ÍĹĞĤĨHĮĻĽ
        ÌIJĹ ĨĮĤHŃd âĽĨĮĹĬHçIJŁĻĴŁĻd ĽĨĮĹĬHçÂĿĞĬŁHĽã ĨĮ ċ
                HĮŁİHĹĞĻHâŅĨĴâĽHĬÌgIJŁĻĴŁĻd ÂĿĞĬŁHĽãã f
            # FĬĞĻĻHĮ IJŁĻĴŁĻ ĞĹĹĞŃ
            ĽĨĮĹĬHçIJŁĻĴŁĻ ł ĽĨĮĹĬHçIJŁĻĴŁĻgĹHĽÎĞĴHâhĝd ĝã
```

```
            # CĞĬĢŁĬĞĻĦ JĞĢIJĠĬĞĮ ĬĞĻĹĨŇ IJÌ ĻÎĦ IJŁĻĴŁĻ ĞĮÎ
            ÏĞĢIJĠĬĞĮçĬĞĻĹĨŇ ł ĮĴgÎĬĞÍÌĬĞĻâĽĨĮÍĬĦçIJŁĻĴŁĻã h ċ
                                ĮĴgÎIJĻâĽĨĮÍĬĦçIJŁĻĴŁĻd ĽĨĮÍĬĦçIJŁĻĴŁĻgTã
            # CĞĬĢŁĬĞĻĦ ĽĞĬĴĬĦhŃĨĽĦ ÍĹĞÎĨĦĮĻ
            # ĞĮÎ ĞÎÎ ĨĻ ĻIJ ĻÎĦ ĞĹĹĞŇ IJÌ ĽĞĬĴĬĦ ÍĹĞÎĨĦĮĻĽ
            ĽĦĬÌgÎĨĮĴŁĻĽäĨĮÎĦŇã ł ĮĴgÎIJĻâÏĞĢIJĠĬĞĮçĬĞĻĹĨŇd
                                        ĽĨĮÍĬĦçÎĻĞĬŁĦĽã


# SĨÍĬIJĨÎ ĞĢĻĨĻĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĻĞĻĨIJĮçSĨÍĬIJĨÎf

    # FIJĹŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ÌIJĹŃĞĹÎâĽĦĬÌd ĨĮĴŁĻĽãf
        # SĞĻĦ ĨĮĴŁĻ ĞĮÎ ĢĞĬĢŁĬĞĻĦwĽĞĻĦ IJŁĻĴŁĻ
        # IJÌ ĻÎĦ ĽĨÍĬIJĨÎ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĝ w âĝ κ ĮĴgĦŃĴâhĨĮĴŁĻĽãã

    # BĞĢĨŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ĠĞĢĨŃĞĹÎâĽĦĬÌd ÎĻĞĬŁĦĽãf
        # DĦĹĨĻĞĻĨĻĦ h ĢĞĬĢŁĬĞĻĦĽ ÌĹIJĬ IJŁĻĴŁĻ IJÌ ĻÎĦ ĽĨÍĬIJĨÎ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgÎĨĮĴŁĻĽ ł ÎĻĞĬŁĦĽ ắ âĝ h ĽĦĬÌgIJŁĻĴŁĻã ắ ĽĦĬÌgIJŁĻĴŁĻ


# SGD IJĴĻĨĬĨŅĦĹ
ĢĬĞĽĽ OĴĻĨĬĨŅĦĹçSGDf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨĬĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮÍĽd
    # ĬĦĞĹĮĨĮÍ ĹĞĻĦ IJÌ ĝg ĨĽ ÎĦÌĞŁĬĻ ÌIJĹ ĻÎĨĽ IJĴĻĨĬĨŅĦĹ
    ÎĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ÎĦĢĞÑłflgd ĬIJĬĦĮĻŁĬłflgãf
        ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgÎĦĢĞÑ ł ÎĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĬIJĬĦĮĻŁĬ ł ĬIJĬĦĮĻŁĬ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴÎĞĻĦĽ
    ÎĦÌ ĴĹĦçŁĴÎĞĻĦçĴĞĹĞĬĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgÎĦĢĞÑf
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĽĦĬÌgÎĦĢĞÑ ắ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã
```

```
ƒ UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
ĤĦÌ ŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

    ƒ IÌ ŃĦ ŁĽĦ İIJİĦĮĻŁİ
    ÕÌ ĽĦĬÌgİIJİĦĮĻŁİf

        ƒ IÌ ĬĞÑĦĹ ĤIJĦĽ ĮIJĻ ǦIJĮĻĞÕĮ İIJİĦĮĻŁİ ĞĹĹĞÑĽd ǦĹĦĞĻĦ ĻĤĦİ
        ƒ ÌÕĬĬĦĤ ŃÕĻĤ ŅĦĹIJĽ
        ÕÌ ĮIJĻ ĤĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦÕĬĤĻçİIJİĦĮĻŁİĽďãf
            ĬĞÑĦĹgŃĦÕĬĤĻçİIJİĦĮĻŁİĽ ł ĮĴgŅĦĹIJĽçĬÕĬĦâĬĞÑĦĹgŃĦÕĬĤĻĽã
            ƒ IÌ ĻĤĦĹĦ ÕĽ ĮIJ İIJİĦĮĻŁİ ĞĹĹĞÑ ÌIJĹ ŃĦÕĬĤĻĽ
            ƒ TĤĦ ĞĹĹĞÑ ĤIJĦĽĮďĻ ĦŃÕĽĻ ÌIJĹ ĠÕĞĽĦĽ ÑĦĻ ĦÕĬĤĻg
            ĬĞÑĦĹgĠÕĞĽçİIJİĦĮĻŁİĽ ł ĮĴgŅĦĹIJĽçĬÕĬĦâĬĞÑĦĹgĠÕĞĽĦĽã

        ƒ BŁÕĬĤ ŃĦÕĬĤĻ ŁĴĤĞĻĦĽ ŃÕĻĤ İIJİĦĮĻŁİ h ĻĞÕĦ ĴĹĦĽÕIJŁĽ
        ƒ ŁĴĤĞĻĦĽ İŁĬĻÕĴĬÕĦĤ ĠÑ ĹĦĻĞÕĮ ÌĞǦĻIJĹ ĞĮĤ ŁĴĤĞĻĦ ŃÕĻĤ
        ƒ ǦŁĹĹĦĮĻ ĹĹĞĤÕĦĮĻĽ
        ŃĦÕĬĤĻçŁĴĤĞĻĦĽ ł ċ
            ĽĦĬÌgİIJİĦĮĻŁİ ǻ ĬĞÑĦĹgŃĦÕĬĤĻçİIJİĦĮĻŁİĽ h ċ
            ĽĦĬÌgǦŁĹĹĦĮĻçĬĦĞĹĮĬĮĬçĹĞĻĦ ǻ ĬĞÑĦĹgĤŃĦÕĬĤĻĽ
        ĬĞÑĦĹgŃĦÕĬĤĻçİIJİĦĮĻŁİĽ ł ŃĦÕĬĤĻçŁĴĤĞĻĦĽ

        ƒ BŁÕĬĤ ĠÕĞĽ ŁĴĤĞĻĦĽ
        ĠÕĞĽçŁĴĤĞĻĦĽ ł ċ
            ĽĦĬÌgİIJİĦĮĻŁİ ǻ ĬĞÑĦĹgĠÕĞĽçİIJİĦĮĻŁİĽ h ċ
            ĽĦĬÌgǦŁĹĹĦĮĻçĬĦĞĹĮĬĮĬçĹĞĻĦ ǻ ĬĞÑĦĹgĤĠÕĞĽĦĽ
        ĬĞÑĦĹgĠÕĞĽçİIJİĦĮĻŁİĽ ł ĠÕĞĽçŁĴĤĞĻĦĽ

    ƒ VĞĮÕĬĬĞ SGD ŁĴĤĞĻĦĽ âĞĽ ĠĦÌIJĹĦ İIJİĦĮĻŁİ ŁĴĤĞĻĦã
    ĦĬĽĦf
        ŃĦÕĬĤĻçŁĴĤĞĻĦĽ ł hĽĦĬÌgǦŁĹĹĦĮĻçĬĦĞĹĮĬĮĬçĹĞĻĦ ǻ ċ
                            ĬĞÑĦĹgĤŃĦÕĬĤĻĽ
        ĠÕĞĽçŁĴĤĞĻĦĽ ł hĽĦĬÌgǦŁĹĹĦĮĻçĬĦĞĹĮĬĮĬçĹĞĻĦ ǻ ċ
                          ĬĞÑĦĹgĤĠÕĞĽĦĽ

    ƒ UĴĤĞĻĦ ŃĦÕĬĤĻĽ ĞĮĤ ĠÕĞĽĦĽ ŁĽÕĮĬ ĦÕĻĤĦĹ
    ƒ ŁĞĮÕĬĬĞ IJĹ İIJİĦĮĻŁİ ŁĴĤĞĻĦĽ
    ĬĞÑĦĹgŃĦÕĬĤĻĽ κł ŃĦÕĬĤĻçŁĴĤĞĻĦĽ
    ĬĞÑĦĹgĠÕĞĽĦĽ κł ĠÕĞĽçŁĴĤĞĻĦĽ

ƒ CĞĬĬ IJĮǦĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
ĤĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
    ĽĦĬÌgÕĻĦĹĞĻÕIJĮĽ κł ĝ
```

```
ƒ AĤĞÍĹĞĤ IJĴĻĨĬĬĨŅĦĹ
ĢĬĞĹĹ OĴĻĨĬĬĨŅĦĹçAĤĞÍĹĞĤf

    ƒ IĮĨĻĨĞĬĬĨŅĦ IJĴĻĨĬĬĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮÍĹ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ĤĦĢĞÑłflgd ĦĴĹĨĬIJĮłĝĦhìãf
        ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgĦĴĹĨĬIJĮ ł ĦĴĹĨĬIJĮ

    ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦĹ
    ĤĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgĤĦĢĞÑf
            ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ő ċ
                âĝg w âĝg κ ĹĦĬÌgĤĦĢĞÑ ő ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    ƒ UĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĹ
    ĤĦÌ ŁĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌd ĬĞÑĦĹãf

        ƒ IÌ ĬĞÑĦĹ ĤIJĦĹ ĮIJĮ ĢIJĮĻĞĨĮ ĢĞĢĨĦ ĞĹĹĞÑĹd
        ƒ ĢĹĦĞĻĦ ĻĨĦĬ ÌĨĬĬĦĤ ŃĨĻĨ ŅĦĹIJĹ
        ĨÌ ĮIJĮ ĨĞĹĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍĨĻçĢĞĢĨĦďãf
            ĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgŃĦĨÍĨĻĹã
            ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã

        ƒ UĴĤĞĻĦ ĢĞĢĨĦ ŃĨĻĨ ĹĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ÍĹĞĤĨĦĮĻĹ
        ĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦ κł ĬĞÑĦĹgĤŃĦĨÍĨĻĹááğ
        ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ κł ĬĞÑĦĹgĤĠĨĞĹĦĹááğ

        ƒ VĞĮĨĬĬĞ SGD ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦ κ IJĹĬĞĬĨŅĞĻĨIJĮ
        ƒ ŃĨĻĨ ĹĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĨĦ
        ĬĞÑĦĹgŃĦĨÍĨĻĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ő ċ
                        ĬĞÑĦĹgĤŃĦĨÍĨĻĹ w ċ
                        âĮĴgĹĶĹĻâĬĞÑĦĹgŃĦĨÍĨĻçĢĞĢĨĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĹĦĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ő ċ
                       ĬĞÑĦĹgĤĠĨĞĹĦĹ w ċ
                       âĮĴgĹĶĹĻâĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã

    ƒ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦĹ
    ĤĦÌ ĴIJĹĻçŁĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌãf
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κł ĝ
```

```
ƒ RMSĴĹIJĴ IJĴĻĨĬĨŅĦĹ
ĢĬĞĽĽ OĴĻĨĬĨŅĦĹçRMSĴĹIJĴf

    ƒ IĮĨĻĨĞĬĨŅĦ IJĴĻĨĬĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĹĽ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮĹçĹĞĻĦłflgflflĝd ÂĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìd
                ĹÎIJłflgîãf
        ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgÂĦĢĞÑ ł ÂĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬÌgĹÎIJ ł ĹÎIJ

    ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴÂĞĻĦĽ
    ÂĦÌ ĴĹĦçłĴÂĞĻĦçĴĞĹĞĬĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgÂĦĢĞÑf
            ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĽĦĬÌgÂĦĢĞÑ ắ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    ƒ UĴÂĞĻĦ ĴĞĹĞĬĦĻĦĹĽ
    ÂĦÌ łĴÂĞĻĦçĴĞĹĞĬĽâĽĦĬÌd ĬĞÑĦĹãf

        ƒ IÌ ĬĞÑĦĹ ÂIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĬĦ ĞĹĹĞÑĽd
        ƒ ĢĹĦĞĻĦ ĻĬĦĬ ÌĬĬĬĦÂ ŃĨĻĬ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ĬĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĬĬĻçĢĞĢĬĦďãf
            ĬĞÑĦĹgŃĦĨĬĬĻçĢĞĢĬĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨĬĬĻĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢĬĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

        ƒ UĴÂĞĻĦ ĢĞĢĬĦ ŃĨĻĬ ĽĶŁĞĹĦÂ ĢŁĹĹĦĮĻ ĬĹĞÂĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨĬĬĻçĢĞĢĬĦ ł ĽĦĬÌgĹÎIJ ắ ĬĞÑĦĹgŃĦĨĬĬĻçĢĞĢĬĦ κ ċ
            âĝ h ĽĦĬÌgĹÎIJã ắ ĬĞÑĦĹgÂŃĦĨĬĬĻĽắắğ
        ĬĞÑĦĹgĠĨĞĽçĢĞĢĬĦ ł ĽĦĬÌgĹÎIJ ắ ĬĞÑĦĹgĠĨĞĽçĢĞĢĬĦ κ ċ
            âĝ h ĽĦĬÌgĹÎIJã ắ ĬĞÑĦĹgÂĠĨĞĽĦĽắắğ

        ƒ VĞĮĨĬĬĞ SGD ĴĞĹĞĬĦĻĦĹ łĴÂĞĻĦ κ ĮIJĹĬĞĬĨŅĞĻĨIJĮ
        ƒ ŃĨĻĬ ĽĶŁĞĹĦ ĹIJIJĻĦÂ ĢĞĢĬĦ
        ĬĞÑĦĹgŃĦĨĬĬĻĽ κł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ắ ċ
                        ĬĞÑĦĹgÂŃĦĨĬĬĻĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨĬĬĻçĢĞĢĬĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ắ ċ
                       ĬĞÑĦĹgÂĠĨĞĽĦĽ w ċ
                       âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢĬĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã

    ƒ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴÂĞĻĦĽ
    ÂĦÌ ĴIJĽĻçłĴÂĞĻĦçĴĞĹĞĬĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ
```

```
ƒ AĤĞİ IJĴĻĨİĨŅĦĹ
ĢĬĞĹĹ OĴĻĨİĨŅĦĹçAĤĞİf

    ƒ IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮĺĹ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮĺçĹĞĻĦłflgflflĝd ĤĦĢĞÑłflgd ĦĴĹĨĬIJĮłĝĦhìd
                ĠĦĻĞçĝłflgîd ĠĦĻĞçğłflgîîîãf
        ĹĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĹĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgĦĴĹĨĬIJĮ ł ĦĴĹĨĬIJĮ
        ĹĦĬÌgĠĦĻĞçĝ ł ĠĦĻĞçĝ
        ĹĦĬÌgĠĦĻĞçğ ł ĠĦĻĞçğ

    ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴĤĞĻĦĹ
    ĤĦÌ ĴĹĦçłĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgĤĦĢĞÑf
            ĹĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĹĦĬÌgĤĦĢĞÑ ắ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    ƒ UĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĹ
    ĤĦÌ łĴĤĞĻĦçĴĞĹĞĬĹâĹĦĬÌd ĬĞÑĦĹãf

        ƒ IÌ ĬĞÑĦĹ ĤIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĬĦ ĞĹĹĞÑĹd
        ƒ ĢĹĦĞĻĦ ĻĬĦĬ ÌĨĬĬĦĤ ÑĨĻĬ ŅĦĹIJĹ
        ĨÌ ĮIJĻ ĬĞĹĞĻĻĹâĬĞÑĦĹd ďŃĦĨĺĬĻçĢĞĢĬĦďãf
            ĬĞÑĦĹgŃĦĨĺĬĻçĬIJĬĦĮĻłĬĹ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgŃĦĨĺĬĻĹã
            ĬĞÑĦĹgŃĦĨĺĬĻçĢĞĢĬĦ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgŃĦĨĺĬĻĹã
            ĬĞÑĦĹgĠĨĞĹçĬIJĬĦĮĻłĬĹ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgĠĨĞĹĦĹã
            ĬĞÑĦĹgĠĨĞĹçĢĞĢĬĦ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgĠĨĞĹĦĹã

        ƒ UĴĤĞĻĦ ĬIJĬĦĮĻłĬ  ÑĨĻĬ ĢłĹĹĦĮĻ ĺĹĞĤĨĦĮĻĹ
        ĬĞÑĦĹgŃĦĨĺĬĻçĬIJĬĦĮĻłĬĹ ł ĹĦĬÌgĠĦĻĞçĝ ắ ċ
                                  ĬĞÑĦĹgŃĦĨĺĬĻçĬIJĬĦĮĻłĬĹ κ ċ
                                  âĝ h ĹĦĬÌgĠĦĻĞçĝã ắ ĬĞÑĦĹgĤŃĦĨĺĬĻĹ
        ĬĞÑĦĹgĠĨĞĹçĬIJĬĦĮĻłĬĹ ł ĹĦĬÌgĠĦĻĞçĝ ắ ċ
                                ĬĞÑĦĹgĠĨĞĹçĬIJĬĦĮĻłĬĹ κ ċ
                                âĝ h ĹĦĬÌgĠĦĻĞçĝã ắ ĬĞÑĦĹgĤĠĨĞĹĦĹ
        ƒ GĦĻ ĢIJĹĹĦĢĻĦĤ ĬIJĬĦĮĻłĬ
        ƒ ĹĦĬÌgĨĻĦĹĞĻĨIJĮ ĨĹ fl ĞĻ ÌĨĹĹĻ ĴĞĹĹ
        ƒ ĞĮĤ ŃĦ ĮĦĦĤ ĻIJ ĹĻĞĹĻ ÑĨĻĬ ĝ ĬĦĹĦ
        ŃĦĨĺĬĻçĬIJĬĦĮĻłĬĹçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨĺĬĻçĬIJĬĦĮĻłĬĹ w ċ
            âĝ h ĹĦĬÌgĠĦĻĞçĝ ắắ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
        ĠĨĞĹçĬIJĬĦĮĻłĬĹçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĹçĬIJĬĦĮĻłĬĹ w ċ
            âĝ h ĹĦĬÌgĠĦĻĞçĝ ắắ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
        ƒ UĴĤĞĻĦ ĢĞĢĬĦ ŃĨĻĬ ĹĶłĞĹĦĤ ĢłĹĹĦĮĻ ĺĹĞĤĨĦĮĻĹ
        ĬĞÑĦĹgŃĦĨĺĬĻçĢĞĢĬĦ ł ĹĦĬÌgĠĦĻĞçğ ắ ĬĞÑĦĹgŃĦĨĺĬĻçĢĞĢĬĦ κ ċ
            âĝ h ĹĦĬÌgĠĦĻĞçğã ắ ĬĞÑĦĹgĤŃĦĨĺĬĻĹắắğ
        ĬĞÑĦĹgĠĨĞĹçĢĞĢĬĦ ł ĹĦĬÌgĠĦĻĞçğ ắ ĬĞÑĦĹgĠĨĞĹçĢĞĢĬĦ κ ċ
            âĝ h ĹĦĬÌgĠĦĻĞçğã ắ ĬĞÑĦĹgĤĠĨĞĹĦĹắắğ
```

```
        # GHL GIJLLHGLHH GGGIH
        NHIIILGGGIHGIJLLHGLHH ł IGNHLgNHIIILGGGIH w ċ
            âĝ h LHIIgGHLGg ąą âLHIIgILHLGLIIJIL κ ĝãã
        GIGLGGGIHGIJLLHGLHH ł IGNHLgGIGLGGGIH w ċ
            âĝ h LHIIgGHLGg ąą âLHIIgILHLGLIIJIL κ ĝãã

        # VGIIIIG SGD JGLGIHLHL ŁJHGLH κ IJLIGIIINGLIIJI
        # NILI LKŁGLH LIJJLHH GGGIH
        IGNHLgNHIIILL κł hLHIIgGŁLLHILGIHGLIIIGLGLH ą ċ
                        NHIIILGIJIHILŁILGIJLLHGLHH w ċ
                        âIJgLKLLâNHIIILGGGIHGIJLLHGLHHã κ
                            LHIIgHJLIIIJIã
        IGNHLgGIGLHL κł hLHIIgGŁLLHILGIHGLIIIGLGLH ą ċ
                        GIGLGIJIHILŁILGIJLLHGLHH w ċ
                        âIJgLKLLâGIGLGGGIHGIJLLHGLHHã κ
                            LHIIgHJLIIIJIã

    # CGII IJIGH GILHL GIN JGLGIHLHL ŁJHGLHL
    HHI JIJLLGŁJHGLHGJGLGILâLHIIãf
        LHIIgILHLGLIIJIL κł g


# CIJIIJI IJLL GIGLL
GIGLL LIJLLf

    # RHŁIGLIINGLIIJI IJLL GGIGŁIGLIIJI
    HHI LHŁIGLIINGLIIJIGIJLLâLHIId IGNHLãf

        # fl GN HHIGŁIL
        LHŁIGLIINGLIIJIGIJLL ł fl

        # Lĝ LHŁIGLIINGLIIJI h NHIIILL
        # GGIGŁIGLH IJIIN NHI IGGLIJL LLHGLHL LIGI fl
        II IGNHLgNHIIILGLHŁIGLINHLGIĝ ń flf
            LHŁIGLINGLIIJIGIJLL κł IGNHLgNHIIILGLHŁIGLINHLGIĝ ą ċ
                                IJgLŁIâIJgGGLâIGNHLgNHIIILLãã

        # Lg LHŁIGLINGLIIJI h NHIIILL
        II IGNHLgNHIIILGLHŁIGLINHLGIg ń flf
            LHŁIGLINGLIIJIGIJLL κł IGNHLgNHIIILGLHŁIGLINHLGIg ą ċ
                                IJgLŁIâIGNHLgNHIIILL ą ċ
                                        IGNHLgNHIIILLã

        # Lĝ LHŁIGLINGLIIJI h GIGLHL
        # GGIGŁIGLH IJIIN NHI IGGLIJL LLHGLHL LIGI fl
        II IGNHLgGIGLGLHŁIGLINHLGIĝ ń flf
            LHŁIGLINGLIIJIGIJLL κł IGNHLgGIGLGLHŁIGLINHLGIĝ ą ċ
                                IJgLŁIâIJgGGLâIGNHLgGIGLHLãã
```

```
        fi Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
        ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń fIf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ κł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ắ ċ
                                    ĮĴgĽŁİâĬĞÑĦĹgĠĨĞĽĦĽ ắ ċ
                                            ĬĞÑĦĹgĠĨĞĽĦĽã

        ĹĦĻŁĹĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ

    fi CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽĦĽ
    fi ÍĨĿĦĮ ĬIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñãf

        fi CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦ ĬIJĹĽĦĽ
        ĽĞİĴĬĦçĬIJĹĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        fi CĞĬĢŁĬĞĻĦ İĦĞĮ ĬIJĹĽ
        ÂĞĻĞçĬIJĹĽ ł ĮĴgİĦĞĮâĽĞİĴĬĦçĬIJĹĽĦĽã

        fi RĦĻŁĹĮ ĬIJĹĽ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĽ


fi CĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ
ĢĬĞĽĽ LIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâLIJĹĽãf

    fi FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ÑçĴĹĦÂd ÑçĻĹŁĦãf

        fi NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ ĨĮ Ğ ĠĞĻĢÎ
        ĽĞİĴĬĦĽ ł ĬĦĮâÑçĴĹĦÂã

        fi CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦĿĦĮĻ ÂĨĿĨĽĨIJĮ ĠÑ fl
        fi CĬĨĴ ĠIJĻÎ ĽĨÂĦĽ ĻIJ ĮIJĻ ÂĹĞÍ İĦĞĮ ĻIJŃĞĹÂĽ ĞĮÑ ĿĞĬŁĦ
        ÑçĴĹĦÂçĢĬĨĴĴĦÂ ł ĮĴgĢĬĨĴâÑçĴĹĦÂd ĝĦhìd ĝ h ĝĦhìã

        fi PĹIJĠĞĠĨĬĨĻĨĦĽ ÌIJĹ ĻĞĹÍĦĻ ĿĞĬŁĦĽ h
        fi IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĽ ł ÑçĴĹĦÂçĢĬĨĴĴĦÂä
                ĹĞĮÍĦâĽĞİĴĬĦĽãd
                ÑçĻĹŁĦ
            ã

        fi MĞĽĨ ĿĞĬŁĦĽ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJÂĦÂ ĬĞĠĦĬĽ
        ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĽ ł ĮĴgĽŁİâ
                ÑçĴĹĦÂçĢĬĨĴĴĦÂ ắ ÑçĻĹŁĦd
                ĞÑĨĽłĝ
            ã
```

```
        # LIJĹĽĦĹ
        ĮĦÍĞĻĨĹĦçĬIJÍçĬĨĨĦĬĨĨIJIJĤĹ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĹã
        ĹĦĻłĹĮ ĮĦÍĞĻĨĹĦçĬIJÍçĬĨĨĦĬĨĨIJIJĤĹ

    # BĞĢĪŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ĤĹĞĬłĦĹd ÑçĻĹłĦãf

        # NłİĠĦĹ IJÌ ĽĞİĴĬĦĹ
        ĽĞİĴĬĦĹ ł ĬĦĮâĤĹĞĬłĦĹã
        # NłİĠĦĹ IJÌ ĬĞĠĦĬĹ ĨĮ ĦĹĦĹÑ ĽĞİĴĬĦ
        # WĦďĬĬ łĹĦ ĻĤĦ ÌĨĹĹĻ ĽĞİĴĬĦ ĻIJ ĢIJłĮĻ ĻĤĦİ
        ĬĞĠĦĬĹ ł ĬĦĮâĤĹĞĬłĦĹäflãã

        # IÌ ĬĞĠĦĬĹ ĞĹĦ ĹĴĞĹĹĦd ĻłĹĮ ĻĤĦİ ĨĮĻIJ IJĮĦhĤIJĻ ĹĦĢĻIJĹ
        ĨÌ ĬĦĮâÑçĻĹłĦgĹĤĞĴĦã łł ĝf
            ÑçĻĹłĦ ł ĮĴgĦÑĦâĬĞĠĦĬĹãäÑçĻĹłĦã

        # CĞĬĢłĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴłĻĹ ł hÑçĻĹłĦ w ĤĹĞĬłĦĹ
        # NIJĹİĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴłĻĹ ł ĽĦĬÌgĤĨĮĴłĻĹ w ĽĞİĴĬĦĹ


# SIJÌĻİĞÑ ĢĬĞĹĹĨÌÌĨĦĹ h ĢIJİĠĨĮĦĤ SIJÌĻİĞÑ ĞĢĻĨĹĞĻĨIJĮ
# ĞĮĤ ĢĹIJĹĹhĦĮĻĹIJĴÑ ĬIJĹĹ ÌIJĹ ÌĞĹĻĦĹ ĠĞĢĪŃĞĹĤ ĹĻĦĴ
ĢĬĞĹĹ AĢĻĨĹĞĻĨIJĮçSIJÌĻİĞÑçLIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑâãf

    # CĹĦĞĻĦĹ ĞĢĻĨĹĞĻĨIJĮ ĞĮĤ ĬIJĹĹ ÌłĮĢĻĨIJĮ IJĠÏĦĢĻĹ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌãf
        ĽĦĬÌgĞĢĻĨĹĞĻĨIJĮ ł AĢĻĨĹĞĻĨIJĮçSIJÌĻİĞÑâã
        ĽĦĬÌgĬIJĹĹ ł LIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑâã

    # FIJĹŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ĨĮĴłĻĹd ÑçĻĹłĦãf
        # OłĻĴłĻ ĬĞÑĦĹďĹ ĞĢĻĨĹĞĻĨIJĮ ÌłĮĢĻĨIJĮ
        ĽĦĬÌgĞĢĻĨĹĞĻĨIJĮgÌIJĹŃĞĹĤâĨĮĴłĻĹã
        # SĦĻ ĻĤĦ IJłĻĴłĻ
        ĽĦĬÌgIJłĻĴłĻ ł ĽĦĬÌgĞĢĻĨĹĞĻĨIJĮgIJłĻĴłĻ
        # CĞĬĢłĬĞĻĦ ĞĮĤ ĹĦĻłĹĮ ĬIJĹĹ ĹĞĬłĦ
        ĹĦĻłĹĮ ĽĦĬÌgĬIJĹĹgĢĞĬĢłĬĞĻĦâĽĦĬÌgIJłĻĴłĻd ÑçĻĹłĦã

    # BĞĢĪŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ĤĹĞĬłĦĹd ÑçĻĹłĦãf

        # NłİĠĦĹ IJÌ ĽĞİĴĬĦĹ
        ĽĞİĴĬĦĹ ł ĬĦĮâĤĹĞĬłĦĹã
```

```
        fi IÌ ĬĞĠĦĬĽ ĞĹĦ IJĮĦhÎIJĻ ĦĮĢIJÂĦÂd
        fi ĻŁĹĮ ĻÎĦİ ĨĮĻIJ ÂĨĽĢĹĦĻĦ ĽĞĬŁĦĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĹÎĞĴĦã łł ğf
            ÑçĻĹŁĦ ł ĮĴgĞĹÍİĞÑâÑçĻĹŁĦd ĞÑĨĽłĝã

        fi CIJĴÑ ĹIJ ŃĦ ĢĞĮ ĹĞÌĦĬÑ İIJÂĨÌÑ
        ĽĦĬÌgÂĨĮĴŁĻĽ ł ÂĽĞĬŁĦĽgĢIJĴÑâã
        fi CĞĬĢŁĬĞĻĦ ÍĹĞÂĨĦĮĻ
        ĽĦĬÌgÂĨĮĴŁĻĽäĹĞĮÍĦâĽĞİĴĬĦĽãd ÑçĻĹŁĦã hł ĝ
        fi NIJĹİĞĬĨŅĦ ÍĹĞÂĨĦĮĻ
        ĽĦĬÌgÂĨĮĴŁĻĽ ł ĽĦĬÌgÂĨĮĴŁĻĽ w ĽĞİĴĬĦĽ


fi BĨĮĞĹÑ ĢĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ
ĢĬĞĽĽ LIJĽĽçBĨĮĞĹÑCĹIJĽĽĦĮĻĹIJĴÑâLIJĽĽãf

    fi FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ÑçĴĹĦÂd ÑçĻĹŁĦãf

        fi CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦĽĦĮĻ ÂĨĽĨĽĨIJĮ ĠÑ fl
        fi CĬĨĴ ĠIJĻÎ ĽĨÂĦĽ ĻIJ ĮIJĻ ÂĹĞÍ İĦĞĮ ĻIJŃĞĹÂĽ ĞĮÑ ĽĞĬŁĦ
        ÑçĴĹĦÂçĢĬĨĴĴĦÂ ł ĮĴgĢĬĨĴâÑçĴĹĦÂd ĝĦhìd ĝ h ĝĦhìã

        fi CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦhŃĨĽĦ ĬIJĽĽ
        ĽĞİĴĬĦçĬIJĽĽĦĽ ł hâÑçĻĹŁĦ ǻ ĮĴgĬIJÍâÑçĴĹĦÂçĢĬĨĴĴĦÂã κ
                            âĝ h ÑçĻĹŁĦã ǻ ĮĴgĬIJÍâĝ h ÑçĴĹĦÂçĢĬĨĴĴĦÂãã
        ĽĞİĴĬĦçĬIJĽĽĦĽ ł ĮĴgİĦĞĮâĽĞİĴĬĦçĬIJĽĽĦĽd ĞÑĨĽłhĝã

        fi RĦĻŁĹĮ ĬIJĽĽĦĽ
        ĹĦĻŁĹĮ ĽĞİĴĬĦçĬIJĽĽĦĽ

    fi BĞĢĨŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ĠĞĢĨŃĞĹÂâĽĦĬÌd ÂĽĞĬŁĦĽd ÑçĻĹŁĦãf

        fi NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâÂĽĞĬŁĦĽã
        fi NŁİĠĦĹ IJÌ IJŁĻĴŁĻĽ ĨĮ ĦĽĦĹÑ ĽĞİĴĬĦ
        fi WĦďĬĬ ŁĽĦ ĻÎĦ ÌĨĹĽĻ ĽĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻÎĦİ
        IJŁĻĴŁĻĽ ł ĬĦĮâÂĽĞĬŁĦĽäflãã

        fi CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦĽĦĮĻ ÂĨĽĨĽĨIJĮ ĠÑ fl
        fi CĬĨĴ ĠIJĻÎ ĽĨÂĦĽ ĻIJ ĮIJĻ ÂĹĞÍ İĦĞĮ ĻIJŃĞĹÂĽ ĞĮÑ ĽĞĬŁĦ
        ĢĬĨĴĴĦÂçÂĽĞĬŁĦĽ ł ĮĴgĢĬĨĴâÂĽĞĬŁĦĽd ĝĦhìd ĝ h ĝĦhìã

        fi CĞĬĢŁĬĞĻĦ ÍĹĞÂĨĦĮĻ
        ĽĦĬÌgÂĨĮĴŁĻĽ ł hâÑçĻĹŁĦ w ĢĬĨĴĴĦÂçÂĽĞĬŁĦĽ h
                            âĝ h ÑçĻĹŁĦã w âĝ h ĢĬĨĴĴĦÂçÂĽĞĬŁĦĽãã w IJŁĻĴŁĻĽ
        fi NIJĹİĞĬĨŅĦ ÍĹĞÂĨĦĮĻ
        ĽĦĬÌgÂĨĮĴŁĻĽ ł ĽĦĬÌgÂĨĮĴŁĻĽ w ĽĞİĴĬĦĽ
```

```
fi CĹĦĞĻĦ ÂĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçÂĞĻĞâĽĞİĴĬĦĽłĝﬂﬂd ĢĬĞĽĽĦĽłğã

fi RĦĽÎĞĴĦ ĬĞĠĦĬĽ ĻIJ ĠĦ Ğ ĬĨĽĻ IJÌ ĬĨĽĻĽ
fi IĮĮĦĹ ĬĨĽĻ ĢIJĮĻĞĨĮĽ IJĮĦ IJŁĻĴŁĻ âĦĨĻÎĦĹ ﬂ IJĹ ĝã
fi ĴĦĹ ĦĞĢÎ IJŁĻĴŁĻ ĮĦŁĹIJĮd ĝ ĨĮ ĻÎĨĽ ĢĞĽĦ
Ñ ł ÑgĹĦĽÎĞĴĦâhĝd ĝã

fi CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ğ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ ĞĮÂ ħǵ IJŁĻĴŁĻ ĽĞĬŁĦĽ
ÂĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâğd ħǵd ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦhǵd
                                ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦhǵã

fi CĹĦĞĻĦ RĦLU ĞĢĻĨĽĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦÂ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĽĞĻĨIJĮĝ ł AĢĻĨĽĞĻĨIJĮçRĦLUâã

fi CĹĦĞĻĦ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĬĦ IJŁĻĴŁĻ
fi IJÌ ĴĹĦĽĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮÂ ĝ IJŁĻĴŁĻ ĽĞĬŁĦ
ÂĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâħǵd ĝã

fi CĹĦĞĻĦ SĨÍİIJĨÂ ĞĢĻĨĽĞĻĨIJĮf
ĞĢĻĨĽĞĻĨIJĮğ ł AĢĻĨĽĞĻĨIJĮçSĨÍİIJĨÂâã

fi CĹĦĞĻĦ ĬIJĽĽ ÌŁĮĢĻĨIJĮ
ĬIJĽĽçÌŁĮĢĻĨIJĮ ł LIJĽĽçBĨĮĞĹÑCĹIJĽĽĦĮĻĹIJĴÑâã

fi CĹĦĞĻĦ IJĴĻĨİĨŅĦĹ
IJĴĻĨİĨŅĦĹ ł OĴĻĨİĨŅĦĹçAÂĞİâÂĦĢĞÑŁĥĦhìã

fi TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮĬĦâĝﬂﬂﬂĝãf

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮĬ ÂĞĻĞ ĻÎĹIJŁĬÎ ĻÎĨĽ ĬĞÑĦĹ
    ÂĦĮĽĦĝgÌIJĹŃĞĹÂâXã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁĬÎ ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    fi ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĽĻ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨĽĞĻĨIJĮĝgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJŁĻĴŁĻã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁĬÎ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ
    fi ĻĞĨĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    fi IJÌ ÌĨĹĽĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
    ÂĦĮĽĦğgÌIJĹŃĞĹÂâĞĢĻĨĽĞĻĨIJĮĝgIJŁĻĴŁĻã

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁĬÎ ĞĢĻĨĽĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    fi ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨĽĞĻĨIJĮğgÌIJĹŃĞĹÂâÂĦĮĽĦğgIJŁĻĴŁĻã

    fi CĞĬĢŁĬĞĻĦ ĻÎĦ ÂĞĻĞ ĬIJĽĽ
    ÂĞĻĞçĬIJĽĽ ł ĬIJĽĽçÌŁĮĢĻĨIJĮgĢĞĬĢŁĬĞĻĦâĞĢĻĨĽĞĻĨIJĮğgIJŁĻĴŁĻd Ñã
```

```
    fi CĞĬĢŁĬĞĻĦ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĴĦĮĞĬĻÑ
    ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ ł ċ
        ĬIJĹĹçÌŁĮĢĻĨIJĮgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹâĤĦĮĹĦĝã κ ċ
        ĬIJĹĹçÌŁĮĢĻĨIJĮgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹâĤĦĮĹĦğã

    fi CĞĬĢŁĬĞĻĦ IJĹĦĹĞĬĬ ĬIJĹĹ
    ĬIJĹĹ ł ĤĞĻĞçĬIJĹĹ κ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ

    fi CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĹĞĻĨIJĮğ ĞĮĤ ĻĞĹÍĦĻĹ
    fi PĞĹĻ ĨĮ ĻĤĦ ĠĹĞĢĨĦĻĹ ĹĦĻŁĹĮĹ Ğ ĠĨĮĞĹÑ İĞĹĨ h ĞĹĹĞÑ ĢIJĮĹĨĹĻĨĮÍ
    fi IJÌ TĹŁĦwFĞĬĹĦ ĹĞĬŁĦĹd İŁĬĻĨĴĬÑĨĮÍ ĨĻ ĠÑ ĝ ĢĤĞĮÍĦĹ ĨĻ ĨĮĻIJ ĞĹĹĞÑ
    fi IJÌ ĝĹ ĞĮĤ flĹ
    ĴĹĦĤĨĢĻĨIJĮĹ ł âĞĢĻĨĹĞĻĨIJĮğgIJŁĻĴŁĻ ń flgh̑ã ắ ĝ
    ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĴĹĦĤĨĢĻĨIJĮĹłłÑã

    ĨÌ ĮIJĻ ĦĴIJĢĤ ķ ĝflflf
        ĴĹĨĮĻâÌďĦĴIJĢĤf ăĦĴIJĢĤåd ď κ
              ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ďκ
              ÌďĬIJĹĹf ăĬIJĹĹfgġÌå âď κ
              ÌďĤĞĻĞçĬIJĹĹf ăĤĞĻĞçĬIJĹĹfgġÌåd ď κ
              ÌďĹĦÍçĬIJĹĹf ăĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹfgġÌåãd ď κ
              ÌďĬĹf ăIJĴĻĨÌĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

    fi BĞĢĨŃĞĹĤ ĴĞĹĹ
    ĬIJĹĹçÌŁĮĢĻĨIJĮgĠĞĢĨŃĞĹĤâĞĢĻĨĹĞĻĨIJĮğgIJŁĻĴŁĻd Ñã
    ĞĢĻĨĹĞĻĨIJĮğgĠĞĢĨŃĞĹĤâĬIJĹĹçÌŁĮĢĻĨIJĮgĤĨĮĴŁĻĹã
    ĤĦĮĹĦğgĠĞĢĨŃĞĹĤâĞĢĻĨĹĞĻĨIJĮğgĤĨĮĴŁĻĹã
    ĞĢĻĨĹĞĻĨIJĮĝgĠĞĢĨŃĞĹĤâĤĦĮĹĦğgĤĨĮĴŁĻĹã
    ĤĦĮĹĦĝgĠĞĢĨŃĞĹĤâĞĢĻĨĹĞĻĨIJĮĝgĤĨĮĴŁĻĹã

    fi UĴĤĞĻĦ ŃĦĨÍÎĻĹ ĞĮĤ ĠĨĞĹĦĹ
    IJĴĻĨİĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĹâã
    IJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĹâĤĦĮĹĦĝã
    IJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĹâĤĦĮĹĦğã
    IJĴĻĨİĨŅĦĹgĴIJĹĻçŁĴĤĞĻĦçĴĞĹĞİĹâã


fi VĞĬĨĤĞĻĦ ĻĤĦ İIJĤĦĬ

fi CĹĦĞĻĦ ĻĦĹĻ ĤĞĻĞĹĦĻ
XçĻĦĹĻd ÑçĻĦĹĻ ł ĹĴĨĹĞĬçĤĞĻĞâĹĞİĴĬĦĹłĝflfld ĢĬĞĹĹĦĹłğã

fi RĦĹĤĞĴĦ ĬĞĠĦĬĹ ĻIJ ĠĦ Ğ ĬĨĹĻ IJÌ ĬĨĹĻĹ
fi IĮĮĦĹ ĬĨĹĻ ĢIJĮĻĞĨĮĹ IJĮĦ IJŁĻĴŁĻ âĦĨĻĤĦĹ fl IJĹ ĝã
fi ĴĦĹ ĦĞĢĤ IJŁĻĴŁĻ ĮĦŁĹIJĮd ĝ ĨĮ ĻĤĨĹ ĢĞĹĦ
ÑçĻĦĹĻ ł ÑçĻĦĹĻgĹĦĹĤĞĴĦâhĝd ĝã
```

```
fi PHĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĹ IJÌ IJŁĹ ĻHĽĻĨĮÍ ĤĞĻĞ ĻĨĹIJŁÍĨ ĻĨĨĽ ĬĞÑHĹ
ĤHĮĽHĝgÌIJĹŃĞĹĤâXçĻHĽĻã

fi PHĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĹ ĻĨĹIJŁÍĨ ĞĢĻĨĹĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
fi ĻĞĪHĽ ĻĨH IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ĤHĮĽH ĬĞÑHĹ ĨHĹH
ĞĢĻĨĹĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤHĮĽHĝgIJŁĻĴŁĻã

fi PHĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĹ ĻĨĹIJŁÍĨ ĽHĢIJĮĤ DHĮĽH ĬĞÑHĹ
fi ĻĞĪHĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĹĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ÌĨĹĽĻ ĬĞÑHĹ ĞĽ ĨĮĴŁĻĽ
ĤHĮĽHğgÌIJĹŃĞĹĤâĞĢĻĨĹĞĻĨIJĮĝgIJŁĻĴŁĻã

fi PHĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĹ ĻĨĹIJŁÍĨ ĞĢĻĨĹĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
fi ĻĞĪHĽ ĻĨH IJŁĻĴŁĻ IJÌ ĽHĢIJĮĤ ĤHĮĽH ĬĞÑHĹ ĨHĹH
ĞĢĻĨĹĞĻĨIJĮğgÌIJĹŃĞĹĤâĤHĮĽHğgIJŁĻĴŁĻã

fi CĞĬĢŁĬĞĻH ĻĨH ĤĞĻĞ ĬIJĽĽ
ĬIJĽĽ ł ĬIJĽĽçÌŁĮĢĻĨIJĮgĢĞĬĢŁĬĞĻHâĞĢĻĨĹĞĻĨIJĮğgIJŁĻĴŁĻd ÑçĻHĽĻã

fi CĞĬĢŁĬĞĻH ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĹĞĻĨIJĮğ ĞĮĤ ĻĞĹĹHĻĽ
fi PĞĹĻ ĨĮ ĻĨH ĠĹĞĢĪHĻĽ ĹHĻŁĹĮĽ Ğ ĠĨĮĞĹÑ ĮĞĽĨ h ĞĹĹĞÑ ĢIJĮĽĨĽĻĨĮÍ IJÌ
fi TĹŁHwFĞĬĽH ĹĞĬŁHĽd ĮŁĬĻĨĴĬÑĨĮÍ ĨĻ ĠÑ ĝ ĢĨĞĮÍHĽ ĨĻ ĨĮĻIJ ĞĹĹĞÑ
fi IJÌ ĝĽ ĞĮĤ flĽ
ĴĹHĤĨĢĻĨIJĮĽ ł âĞĢĻĨĹĞĻĨIJĮğgIJŁĻĴŁĻ ń flgĥã ấ ĝ
ĞĢĢŁĹĞĢÑ ł ĮĴgİHĞĮâĴĹHĤĨĢĻĨIJĮĽłłÑçĻHĽĻã

ĴĹĨĮĻâÌďĹĞĬĨĤĞĻĨIJĮd ĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ĬIJĽĽf ăĬIJĽĽfgġÌåďã


ńńń
HĴIJĢĴf fld ĞĢĢf flgĥflfld ĬIJĽĽf flgħiġ âĤĞĻĞçĬIJĽĽf flgħiġd ĹHÍçĬIJĽĽf flgflflfld ĬĹf
flgflflĝ
HĴIJĢĴf ĝflfld ĞĢĢf flgħġfld ĬIJĽĽf flgħìġ âĤĞĻĞçĬIJĽĽf flgħìġd ĹHÍçĬIJĽĽf flgflflĝad ĬĹf
flgflflflîîîîĥflĥflĝġĥflĝĝíì
HĴIJĢĴf ĝflfld ĞĢĢf flgħĝĥd ĬIJĽĽf flgħĥî âĤĞĻĞçĬIJĽĽf flgħĥíd ĹHÍçĬIJĽĽf flgflflĝad ĬĹf
flgflflflîîîîflflĥflîííîĝĥĥĝ
HĴIJĢĴf ġflfld ĞĢĢf flgħĥfld ĬIJĽĽf flgħħġ âĤĞĻĞçĬIJĽĽf flgħħġd ĹHÍçĬIJĽĽf flgflflĝad ĬĹf
flgflflflîîîíĥflĥĝĝġġĥîflî
HĴIJĢĴf ġflfld ĞĢĢf flgħĥfld ĬIJĽĽf flgħĥî âĤĞĻĞçĬIJĽĽf flgħĥìd ĹHÍçĬIJĽĽf flgflflĝad ĬĹf
flgflflflîîîíflflĥġîîĝġĝĝĥ
HĴIJĢĴf ĥflfld ĞĢĢf flgħìĥd ĬIJĽĽf flgħġì âĤĞĻĞçĬIJĽĽf flgħġġd ĹHÍçĬIJĽĽf flgflflġad ĬĹf
flgflflflîîîìĥflĥħĝĝġġìĝĝĥ
HĴIJĢĴf ħflfld ĞĢĢf flgìĝfld ĬIJĽĽf flgħġĝ âĤĞĻĞçĬIJĽĽf flgħĝĥd ĹHÍçĬIJĽĽf flgflflħad ĬĹf
flgflflflîîîìflflĥííħìġġîĝî
ggg
HĴIJĢĴf ĝĥflfld ĞĢĢf flgíflĥd ĬIJĽĽf flgĥflġ âĤĞĻĞçĬIJĽĽf flgġħġd ĹHÍçĬIJĽĽf flgflġîad
ĬĹf flgflflflîîîĝĥĝflħĝġĝîĥġġĥ
ggg
HĴIJĢĴf ĝĥflfld ĞĢĢf flgíĥĥd ĬIJĽĽf flgġġfl âĤĞĻĞçĬIJĽĽf flgġìîd ĹHÍçĬIJĽĽf flgflĥĝad
ĬĹf flgflflflîîíìĥĝflĥîġflĝîflĝĥ
```

```
ggg
ĦĴIJĢÎf ǵĥflfld ĞĢĢf flgîĝfld ĬIJĹĹf flgġǵħ âĤĞĻĞçĬIJĹĹf flgğíĥd ĹĦÍçĬIJĹĹf flgflħĝãd
ĬĹf flgflflfliîììĥĥĥǵííîğìħĥí
ĦĴIJĢÎf ǵħflfld ĞĢĢf flgîflĥd ĬIJĹĹf flgġǵfl âĤĞĻĞçĬIJĹĹf flgğìíd ĹĦÍçĬIJĹĹf flgflħğãd
ĬĹf flgflflfliîììflĥììĥĥħîflìî
ĦĴIJĢÎf ǵìflfld ĞĢĢf flgîĝfld ĬIJĹĹf flgġǵfl âĤĞĻĞçĬIJĹĹf flgğħíd ĹĦÍçĬIJĹĹf flgflħğãd
ĬĹf flgflflfliîìħĥħflflìğĝĝflĥìì
ĦĴIJĢÎf ǵíflfld ĞĢĢf flgîğfld ĬIJĹĹf flgġğħ âĤĞĻĞçĬIJĹĹf flgğħġd ĹĦÍçĬIJĹĹf flgflħġãd
ĬĹf flgflflfliîìħflħğǵǵíĝìîĥíì
ggg
ĦĴIJĢÎf ħĝflfld ĞĢĢf flgîǵfld ĬIJĹĹf flgğîĝ âĤĞĻĞçĬIJĹĹf flgğğǵd ĹĦÍçĬIJĹĹf flgflħîãd
ĬĹf flgflflfliîħîĥîììĝĝììîǵĥ
ggg
ĦĴIJĢÎf ħħflfld ĞĢĢf flgîĥfld ĬIJĹĹf flgğìî âĤĞĻĞçĬIJĹĹf flgğĝĝd ĹĦÍçĬIJĹĹf flgflħíãd
ĬĹf flgflflfliîħìĝĝǵĥflíîìîĝǵ
ĦĴIJĢÎf ħìflfld ĞĢĢf flgîĥĥd ĬIJĹĹf flgğìğ âĤĞĻĞçĬIJĹĹf flgğflġd ĹĦÍçĬIJĹĹf flgflħîãd
ĬĹf flgflflfliîħħħĝħíĝħîìĝĥĥħ
ĦĴIJĢÎf ħíflfld ĞĢĢf flgîĥĥd ĬIJĹĹf flgğħî âĤĞĻĞçĬIJĹĹf flgğflfld ĹĦÍçĬIJĹĹf flgflħîãd
ĬĹf flgflflfliîħĝğflĝìǵǵħħî
ĦĴIJĢÎf ħîflfld ĞĢĢf flgîħfld ĬIJĹĹf flgğħħ âĤĞĻĞçĬIJĹĹf flgĝîìd ĹĦÍçĬIJĹĹf flgflħîãd
ĬĹf flgflflfliîħĥħğǵĥíĝǵĥĥìħì
ggg
ĦĴIJĢÎf îíflfld ĞĢĢf flgîħĥd ĬIJĹĹf flgğğğ âĤĞĻĞçĬIJĹĹf flgĝĥíd ĹĦÍçĬIJĹĹf flgflħġãd
ĬĹf flgflflfliîĥĝğǵǵíflħflħîħħ
ĦĴIJĢÎf îîflfld ĞĢĢf flgîħĥd ĬIJĹĹf flgğğĝ âĤĞĻĞçĬIJĹĹf flgĝĥìd ĹĦÍçĬIJĹĹf flgflħġãd
ĬĹf flgflflfliîĥflìǵíìħíîħìîîǵ
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgîħĥd ĬIJĹĹf flgğĝî âĤĞĻĞçĬIJĹĹf flgĝĥħd ĹĦÍçĬIJĹĹf flgflħġãd
ĬĹf flgflflfliîĥflğĥǵìflħĥîǵííĥ
ĻĞĬĨĤĞĻĨIJĮd ĞĢĢf flgîǵĥd ĬIJĹĹf flgğflì
```

The model performed quite well here! You should have some intuition about tweaking the output layer to better fit the problem you're attempting to solve while keeping your hidden layers mostly the same. In the next chapter, we're going to work on regression, where our intended output is not a classification at all, but rather to predict a scalar value, like the price of a house.

Supplementary Material: https://nnfs.io/ch16

Chapter code, further resources, and errata for this chapter.

Chapter 17

Regression

Up until this point, we’ve been working with classification models, where we try to determine *what* something is. Now we’re curious about determining a *specific* value based on an input. For example, you might want to use a neural network to predict what the temperature will be tomorrow or what the price of a car should be. For a task like this, we need something with a much more granular output. This also means that we require a new way to measure loss, as well as a new output layer activation function! It also means our data are different. We need training data that have target scalar values, not classes.

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĨİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĨİĴIJĹĻ ĽĨĮĦçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


Xd Ñ ł ĽĨĮĦçĤĞĻĞâã
```

```
ĴĬĻgĴĬIJĻâXd Ñã
ĴĬĻgĽÎIJŃâã
```

The data above will produce a graph like:

Fig 17.01: The sine data graph.

Linear Activation

Since we're no longer using classification labels and want to predict a scalar value, we're going to use a linear activation function for the output layer. This linear function does not modify its input and passes it to the output: *y=x*. For the backward pass, we already know the derivative of *f(x)=x* is *1*; thus, the full class for our new linear activation function is:

```
class LinearActivation:
ĝĭĝĺĺ AĝĻĩĹĝĻĩIJĮçLĩĮĦĝĹf

    def FIJĹŃĝĹĤ ĵĝĺĺ
    ĤĦĬ ÌIJĹŃĝĹĤâĺĦĭÌd ĩĮĵłĻĺãf
        # JłĺĻ ĹĦİĦİĠĦĹ ĹĝĭłĦĺ
        ĺĦĭÌgĩĮĵłĻĺ ł ĩĮĵłĻĺ
        ĺĦĭÌgIJłĻĵłĻ ł ĩĮĵłĻĺ

    # BĝĢĩŃĝĹĤ ĵĝĺĺ
    ĤĦĬ ĠĝĢĩŃĝĹĤâĺĦĭÌd ĤĹĝĭłĦĺãf
        # ĤĦĹĩĹĝĻĩĹĦ ĩĺ ĝd ĝ ắ ĤĹĝĭłĦĺ ł ĤĹĝĭłĦĺ h ĻĤĦ ĢĤĝĩĮ ĹłĭĦ
        ĺĦĭÌgĤĩĮĵłĻĺ ł ĤĹĝĭłĦĺgĢIJÑâã
```

This might raise a question — why do we even write some code that does nothing? We just pass inputs to outputs for the forward pass and do the same with gradients during the backward pass since, to apply the chain rule, we multiply incoming gradients by the derivative, which is *1*. We do this only for completeness and clarity to see the activation function of the output layer in the model definition code. From a computational time point of view, this adds almost nothing to the processing time, at least not enough to noticeably impact training times.

Now we just need to figure out loss!

Mean Squared Error Loss

Since we aren't working with classification labels anymore, we cannot calculate cross-entropy. Instead, we need some new methods. The two main methods for calculating error in regression are **mean squared error** (MSE) and **mean absolute error** (MAE).

With **mean squared error**, you square the difference between the predicted and true values of single outputs (as the model can have multiple regression outputs) and average those squared values.

$$L_i = \frac{1}{J} \sum_j (y_{i,j} - \hat{y}_{i,j})^2$$

Where *y* means the target value, *y-hat* means predicted value, index *i* means the current sample, index *j* means the current output in this sample, and the *J* means the number of outputs.

The idea here is to penalize more harshly the further away we get from the intended target.

Mean Squared Error Loss Derivative

The partial derivative of squared error with respect to the predicted value is:

$$\frac{\partial}{\partial \hat{y}_{i,j}} L_i = \frac{\partial}{\partial \hat{y}_{i,j}} [\frac{1}{J} \sum_j (y_{i,j} - \hat{y}_{i,j})^2] =$$

1 divided by *J* (the number of outputs) is a constant and can be moved outside of the derivative. Since we are calculating the derivative with respect to the given output, *j*, the sum of one element equals this element:

$$= \frac{1}{J} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} (y_{i,j} - \hat{y}_{i,j})^2 =$$

To calculate the partial derivative of an expression to the power of some value, we need to multiply this exponent by the expression, subtract 1 from the exponent, and multiply this by the partial derivative of the inner function:

$$= \frac{1}{J} \cdot 2 \cdot (y_{i,j} - \hat{y}_{i,j})^{2-1} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} [y_{i,j} - \hat{y}_{i,j}] =$$

The partial derivative of the subtraction equals the subtraction of the partial derivatives:

$$= \frac{2}{J} \cdot (y_{i,j} - \hat{y}_{i,j})^1 \cdot (\frac{\partial}{\partial \hat{y}_{i,j}} y_{i,j} - \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j}) =$$

The partial derivative of the ground truth value with respect to the predicted value equals *0* since we treat other variables as constants. The partial derivative of the predicted value with respect to itself equals *1*, which results in *0-1=-1*. This is multiplied by the rest of the equation and forms the solution:

$$= \frac{2}{J} \cdot (y_{i,j} - \hat{y}_{i,j}) \cdot (0 - 1) = \frac{2}{J} \cdot (y_{i,j} - \hat{y}_{i,j}) \cdot (-1) = -\frac{2}{J} \cdot (y_{i,j} - \hat{y}_{i,j})$$

Full solution:

$$\frac{\partial}{\partial \hat{y}_{i,j}} L_i = \frac{\partial}{\partial \hat{y}_{i,j}} [\frac{1}{J} \sum_j (y_{i,j} - \hat{y}_{i,j})^2] = \frac{1}{J} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} (y_{i,j} - \hat{y}_{i,j})^2 =$$

$$= \frac{1}{J} \cdot 2 \cdot (y_{i,j} - \hat{y}_{i,j})^{2-1} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} [y_{i,j} - \hat{y}_{i,j}] =$$

$$= \frac{2}{J} \cdot (y_{i,j} - \hat{y}_{i,j})^1 \cdot (\frac{\partial}{\partial \hat{y}_{i,j}} y_{i,j} - \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j}) = \frac{2}{J} \cdot (y_{i,j} - \hat{y}_{i,j}) \cdot (0 - 1) =$$

$$= \frac{2}{J} \cdot (y_{i,j} - \hat{y}_{i,j}) \cdot (-1) = -\frac{2}{J} \cdot (y_{i,j} - \hat{y}_{i,j})$$

The partial derivative equals *-2*, multiplied by the subtraction of the true and predicted values, and then divided by the number of outputs to normalize the gradients, making their magnitude invariant to the number of outputs.

Mean Squared Error (MSE) Loss Code

The code for MSE includes an implementation of the equation to calculate the sample loss from multiple outputs. ĞŇĨĽłhĝ with the mean calculation was explained in the previous chapter in detail and, in short words, it informs NumPy to calculate mean across outputs, for each sample separately. For the backward pass, we implemented the derivative equation, which results in *-2* multiplied by the difference of true and predicted values, and normalized by the number of outputs. Similarly to the other loss function implementations, we also normalize gradients by the number of samples to make them invariant to the batch size, or the number of samples in general:

```
class MseLoss(Loss):
    def forward(self, y_pred, y_true):
        sample_losses = np.mean(np.sum(np.square(y_true - y_pred), axis=-1), axis=-1)
        return sample_losses

    def backward(self, dvalues, y_true):
        samples = len(dvalues)
        outputs = len(dvalues[0])
        self.dinputs = -2 * (y_true - dvalues) / outputs
        self.dinputs = self.dinputs / samples
```

```
        # RHĻŁĹĮ ĬIJĽĽHĽ
        ĹHĻŁĹĮ ĽĞİĴĬHçĬIJĽĽHĽ

    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ĤŁĞĬŁHĽd ÑçĻĹŁHãf

        # NŁİĠHĹ IJÌ ĽĞİĴĬHĽ
        ĽĞİĴĬHĽ ł ĬHĮâĤŁĞĬŁHĽã
        # NŁİĠHĹ IJÌ IJŁĻĴŁĻĽ ĨĮ HŁHĹÑ ĽĞİĴĬH
        # WHďĬĬ ŁĽH ĻĤH ÌĨĹĽĻ ĽĞİĴĬH ĻIJ ĢIJŁĮĻ ĻĤHİ
        IJŁĻĴŁĻĽ ł ĬHĮâĤŁĞĬŁHĽäflãã

        # GĹĞĤĨHĮĻ IJĮ ŁĞĬŁHĽ
        ĽHĬÌgĤĨĮĴŁĻĽ ł hğ ấ âÑçĻĹŁH h ĤŁĞĬŁHĽã w IJŁĻĴŁĻĽ
        # NIJĹİĞĬĨŅH ÍĹĞĤĨHĮĻ
        ĽHĬÌgĤĨĮĴŁĻĽ ł ĽHĬÌgĤĨĮĴŁĻĽ w ĽĞİĴĬHĽ
```

Mean Absolute Error Loss

With **mean absolute error**, you take the absolute difference between the predicted and true values in a single output and average those absolute values.

$$L_i = \frac{1}{J} \sum_j |y_{i,j} - \hat{y}_{i,j}|$$

Where *y* means the target value, *y-hat* means predicted value, index *i* means the current sample, index *j* means the current output in this sample, and the *J* means the number of outputs.

This function, used as a loss, penalizes the error linearly. It produces sparser results and is robust to outliers, which can be both advantageous and disadvantageous. In reality, L1 (MAE) loss is used less frequently than L2 (MSE) loss.

Mean Absolute Error Loss Derivative

The partial derivative for absolute error with respect to the predicted values is:

$$\frac{\partial}{\partial \hat{y}_{i,j}} L_i = \frac{\partial}{\partial \hat{y}_{i,j}} [\frac{1}{J} \sum_j |y_{i,j} - \hat{y}_{i,j}|]$$

1 divided by *J* (the number of outputs) is a constant, and can be moved outside of the derivative. Since we are calculating the derivative with respect to the given output, *j*, the sum of one element equals this element:

$$= \frac{1}{J} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} |y_{i,j} - \hat{y}_{i,j}| =$$

We already calculated the partial derivative of an absolute value for the L1 regularization, which is similar to the L1 loss. The derivative of an absolute value equals *1* if this value is greater than *0*, or *-1* if it's less than *0*. The derivative does not exist for a value of *0*:

$$= \frac{1}{J} \cdot \begin{cases} 1 & y_{i,j} - \hat{y}_{i,j} > 0 \\ -1 & y_{i,j} - \hat{y}_{i,j} < 0 \end{cases}$$

Full solution:

$$\frac{\partial}{\partial \hat{y}_{i,j}} L_i = \frac{\partial}{\partial \hat{y}_{i,j}} [\frac{1}{J} \sum_j |y_{i,j} - \hat{y}_{i,j}|] = \frac{1}{J} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} |y_{i,j} - \hat{y}_{i,j}| = \frac{1}{J} \cdot \begin{cases} 1 & y_{i,j} - \hat{y}_{i,j} > 0 \\ -1 & y_{i,j} - \hat{y}_{i,j} < 0 \end{cases}$$

Mean Absolute Error Loss Code

The code for mean absolute error is very similar to the mean squared error. The forward pass includes NumPy's `ĮĴgĞĠĽâã` to calculate absolute values before calculating the mean. For the backward pass, we'll use `ĮĴgĽĨÍĮâãd` which returns 1 or -1 given the sign of the input and *0* if the parameter equals *0*, then normalize gradients by the number of samples to make them invariant to the batch size, or number of samples in general:

```
# MĦĞĮ AĠĽIJĬŁĻĦ EĹĹIJĹ ĬIJĽĽ
ĢĬĞĽĽ LIJĽĽçMĦĞĮAĠĽIJĬŁĻĦEĹĹIJĹâLIJĽĽãf  # Lĝ ĬIJĽĽ

    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        # CĞĬĢŁĬĞĻĦ ĬIJĽĽ
        ĽĞİĴĬĦçĬIJĽĽĦĽ ł ĮĴgİĦĞĮâĮĴgĞĠĽâÑçĻĹŁĦ h ÑçĴĹĦĤãd ĞÑĨĽłhĝã

        # RĦĻŁĹĮ ĬIJĽĽĦĽ
        ĹĦĻŁĹĮ ĽĞİĴĬĦçĬIJĽĽĦĽ

    # BĞĢĨŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤĽĞĬŁĦĽd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâĤĽĞĬŁĦĽã
        # NŁİĠĦĹ IJÌ IJŁĻĴŁĻĽ ĨĮ ĦĽĦĹÑ ĽĞİĴĬĦ
        # WĦďĬĬ ŁĽĦ ĻĤĦ ÌĨĹĽĻ ĽĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦİ
        IJŁĻĴŁĻĽ ł ĬĦĮâĤĽĞĬŁĦĽäflãã

        # CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĮĴgĽĨÍĮâÑçĻĹŁĦ h ĤĽĞĬŁĦĽã w IJŁĻĴŁĻĽ
        # NIJĹİĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽĞİĴĬĦĽ
```

Accuracy in Regression

Now that we've got data, an activation function, and a loss calculation for regression, we'd like to measure performance.

With cross-entropy, we were able to count the number of matches (situations where the prediction equals the ground truth target), and then divide it by the number of samples to measure the model's accuracy. With a regression model, we have two problems: the first problem is that each output neuron in the model (there might be many) is a separate output — like in a binary regression model and unlike in a classifier, where all outputs contribute toward a common prediction. The second problem is that the prediction is a float value, and we can't simply check if the output value equals the ground truth one, as it most likely won't — if it differs even slightly, the accuracy will be a 0. For example, if your model predicts home prices and one of the samples has the target price of $192,500, and the predicted value is $192,495, then a pure "is it equal" assessment would return False. We'd likely consider the predicted price to be correct or "close enough" in this scenario, given the magnitude of the numbers in consideration. There's no perfect way to show accuracy with regression. Still, it is preferable to have some accuracy metric.

For example, Keras, a popular deep learning framework, shows both accuracy and loss for regression models, and we'll also make our own accuracy metric.

First, we need some "limit" value, which we'll call "precision." To calculate this precision, we'll calculate the standard deviation from the ground truth target values and then divide it by *250*. This value can certainly vary depending on your goals. The larger the number you divide by, the more "strict" the accuracy metric will be. *250* is our value of choice. Code to represent this:

```
ĞĢĢŁĹĞĢÑçĴĹĦĢĨĹĨIJĮ ł ĮĴgĹĻĤâÑã w ğĥfl
```

Then we could use this precision value as a sort of "cushion allowance" for regression outputs when comparing targets and predicted values for accuracy. We perform the comparison by applying the absolute value on the difference between the ground truth values and the predictions. Then we check if the difference is smaller than our previously calculated precision:

```
ĴĹĦĤĨĢĻĨIJĮĹ ł ĞĢĻĨĿĞĻĨIJĮğgIJŁĻĴŁĻ
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĮĴgĞĠĹIJĬŁĻĦâĴĹĦĤĨĢĻĨIJĮĹ h Ñã l·
          ĞĢĢŁĹĞĢÑçĴĹĦĢĨĹĨIJĮã
```

Regression Model Training

With this new activation function, loss, and way of calculating accuracy, we now create our model:

```
# CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĨĮĦçĤĞĻĞâã

# CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ĝ ĨĮĴłĻ ÌĦĞĻłĹĦ ĞĮĤ ħǵ IJłĻĴłĻ ĽĞĬłĦĽ
ĤĦĮĽĦĝ ł LĞÑĦĹçDĦĮĽĦâĝd ħǵã

# CĹĦĞĻĦ RĦLU ĞĢĻĨĽĞĻĨIJĮ âĻIJ ĠĦ łĽĦĤ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĽĞĻĨIJĮĝ ł AĢĻĨĽĞĻĨIJĮçRĦLUâã

# CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴłĻ ÌĦĞĻłĹĦĽ âĞĽ ŃĦ ĻĞĪĦ IJłĻĴłĻ
# IJÌ ĴĹĦĽĨIJłĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ĝ IJłĻĴłĻ ĽĞĬłĦ
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâħǵd ĝã
# CĹĦĞĻĦ LĨĮĦĞĹ ĞĢĻĨĽĞĻĨIJĮf
ĞĢĻĨĽĞĻĨIJĮğ ł AĢĻĨĽĞĻĨIJĮçLĨĮĦĞĹâã

# CĹĦĞĻĦ ĬIJĽĽ ÌłĮĢĻĨIJĮ
ĬIJĽĽçÌłĮĢĻĨIJĮ ł LIJĽĽçMĦĞĮSĶłĞĹĦĤEĹĹIJĹâã

# CĹĦĞĻĦ IJĴĻĨİĨŅĦĹ
IJĴĻĨİĨŅĦĹ ł OĴĻĨİĨŅĦĹçAĤĞİâã

# AĢĢłĹĞĢÑ ĴĹĦĢĨĽĨIJĮ ÌIJĹ ĞĢĢłĹĞĢÑ ĢĞĬĢłĬĞĻĨIJĮ
# TĨĦĹĦ ĞĹĦ ĮIJ ĹĦĞĬĬÑ ĞĢĢłĹĞĢÑ ÌĞĢĻIJĹ ÌIJĹ ĹĦĹĦĽĽĨIJĮ ĴĹIJĠĬĦİd
# ĠłĻ ŃĦ ĢĞĮ ĽĨİłĬĞĻĦwĞĴĴĹIJŃĨİĞĻĦ ĨĻg WĦďĬĬ ĢĞĬĢłĬĞĻĦ ĨĻ ĠÑ ĢĨĦĢĨĨĮĹ
# ÎIJŃ İĞĮÑ ĽĞĬłĦĽ ÎĞĽĦ Ğ ĤĨÌÌĦĹĦĮĢĦ ĻIJ ĻĨĦĨĹ ĹĹIJłĮĤ ĻĹłĻÎ ĦĶłĨĽĞĬĦĮĻ
# ĬĦĽĽ ĻÎĞĮ ĹĨĽĦĮ ĴĹĦĢĨĽĨIJĮ
# WĦďĬĬ ĢĞĬĢłĬĞĻĦ ĻÎĨĽ ĴĹĦĢĨĽĨIJĮ ĞĽ Ğ ÌĹĞĢĻĨIJĮ IJÌ ĽĻĞĮĤĞĹĤ ĤĦĽĨĞĻĨIJĮ
# IJÌ ĞĬ ĻÎĦ ĹĹIJłĮĤ ĻĹłĻÎ ĽĞĬłĦĽ
ĞĢĢłĹĞĢÑçĴĹĦĢĨĽĨIJĮ ł ĮĴgĽĻĤâÑã w ğĥfl

# TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮĹĦâĝflflflĝãf
```

```
ſi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ÂĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑĦĹ
ÎĦĮĽĦĝgÌIJĹŃĞĹÎâXã

ſi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
ſi ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ÎĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨĿĞĻĨIJĮĝgÌIJĹŃĞĹÎâÎĦĮĽĦĝgIJŁĻĴŁĻã

ſi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽĦĢIJĮÎ DĦĮĽĦ ĬĞÑĦĹ
ſi ĻĞĨĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
ſi IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
ÎĦĮĽĦğgÌIJĹŃĞĹÎâĞĢĻĨĿĞĻĨIJĮĝgIJŁĻĴŁĻã

ſi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÎ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĿĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
ſi ĻĞĨĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮÎ ÎĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻĨĿĞĻĨIJĮğgÌIJĹŃĞĹÎâÎĦĮĽĦğgIJŁĻĴŁĻã

ſi CĞĬĢŁĬĞĻĦ ĻÎĦ ÎĞĻĞ ĬIJĽĽ
ÎĞĻĞçĬIJĽĽ ł ĬIJĽĽçÌŁĮĢĻĨIJĮgĢĞĬĢŁĬĞĻĦâĞĢĻĨĿĞĻĨIJĮğgIJŁĻĴŁĻd Ñã

ſi CĞĬĢŁĬĞĻĦ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĴĦĮĞĬĻÑ
ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ ł ċ
    ĬIJĽĽçÌŁĮĢĻĨIJĮgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽâÎĦĮĽĦĝã κ ċ
    ĬIJĽĽçÌŁĮĢĻĨIJĮgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽâÎĦĮĽĦğã

ſi CĞĬĢŁĬĞĻĦ IJĿĦĹĞĬĬ ĬIJĽĽ
ĬIJĽĽ ł ÎĞĻĞçĬIJĽĽ κ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ

ſi CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĿĞĻĨIJĮğ ĞĮÎ ĻĞĹÍĦĻĽ
ſi TIJ ĢĞĬĢŁĬĞĻĦ ĨĻ ŃĦďĹĦ ĻĞĨĨĮÍ ĞĠĽIJĬŁĻĦ ÎĨÌÌĦĹĦĮĢĦ ĠĦĻŃĦĦĮ
ſi ĴĹĦÎĨĢĻĨIJĮĽ ĞĮÎ ÍĹIJŁĮÎ ĻĹŁĻÎ ĿĞĬŁĦĽ ĞĮÎ ĢIJİĴĞĹĦ ĨÌ ÎĨÌÌĦĹĦĮĢĦĽ
ſi ĞĹĦ ĬIJŃĦĹ ĻÎĞĮ ÍĨĿĦĮ ĴĹĦĢĨĽĨIJĮ ĿĞĬŁĦ
ĴĹĦÎĨĢĻĨIJĮĽ ł ĞĢĻĨĿĞĻĨIJĮğgIJŁĻĴŁĻ
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĮĴgĞĠĽIJĬŁĻĦâĴĹĦÎĨĢĻĨIJĮĽ h Ñã l·
                    ĞĢĢŁĹĞĢÑçĴĹĦĢĨĽĨIJĮã

ĨÌ ĮIJĻ ĦĴIJĢÎ ķ ĝflflf
    ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
        ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
        ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď κ
        ÌďÎĞĻĞçĬIJĽĽf ăÎĞĻĞçĬIJĽĽfgġÌåd ď κ
        ÌďĹĦÍçĬIJĽĽf ăĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽfgġÌåãd ď κ
        ÌďĬĹf ăIJĴĻĨÌĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

ſi BĞĢĨŃĞĹÎ ĴĞĽĽ
ĬIJĽĽçÌŁĮĢĻĨIJĮgĠĞĢĨŃĞĹÎâĞĢĻĨĿĞĻĨIJĮğgIJŁĻĴŁĻd Ñã
ĞĢĻĨĿĞĻĨIJĮğgĠĞĢĨŃĞĹÎâĬIJĽĽçÌŁĮĢĻĨIJĮgÎĨĮĴŁĻĽã
ÎĦĮĽĦğgĠĞĢĨŃĞĹÎâĞĢĻĨĿĞĻĨIJĮğgÎĨĮĴŁĻĽã
ĞĢĻĨĿĞĻĨIJĮĝgĠĞĢĨŃĞĹÎâÎĦĮĽĦğgÎĨĮĴŁĻĽã
ÎĦĮĽĦĝgĠĞĢĨŃĞĹÎâĞĢĻĨĿĞĻĨIJĮĝgÎĨĮĴŁĻĽã
```

```
    ńı UĴĤĞĻĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
    IJĴĻĨİĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâã
    IJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĤĦĮĽĦĝã
    IJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĤĦĮĽĦğã
    IJĴĻĨİĨŅĦĹgĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâã


ńńń
ĦĴIJĢĴf fld ĞĢĢf flgflflğd ĬIJĽĽf flgĥflfl âĤĞĻĞçĬIJĽĽf flgĥflfld ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflĝ
ĦĴIJĢĴf ĝflfld ĞĢĢf flgflflġd ĬIJĽĽf flgġǵħ âĤĞĻĞçĬIJĽĽf flgġǵħd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflĝ
ggg
ĦĴIJĢĴf îîflfld ĞĢĢf flgflflġd ĬIJĽĽf flgĝǵĥ âĤĞĻĞçĬIJĽĽf flgĝǵĥd ĹĦÍçĬIJĽĽf flgflflfãd
ĬĹf flgflflĝ
ĦĴIJĢĴf ĝflflflfld ĞĢĢf flgflflǵd ĬIJĽĽf flgĝǵĥ âĤĞĻĞçĬIJĽĽf flgĝǵĥd ĹĦÍçĬIJĽĽf flgflflfãd
ĬĹf flgflflĝ
```

Training didn't work out here very well! Let's add an ability to draw the testing data and let's also do a forward pass on the testing data, drawing output data on the same plot as well:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ

XçĻĦĽĻd ÑçĻĦĽĻ ł ĽĨĮĦçĤĞĻĞâã

ĤĦĮĽĦĝgÌIJĹŃĞĹĤâXçĻĦĽĻã
ĞĢĻĨĹĞĻĨIJĮĝgÌIJĹŃĞĹĤâĤĦĮĽĦĝgIJŁĻĴŁĻã
ĤĦĮĽĦğgÌIJĹŃĞĹĤâĞĢĻĨĹĞĻĨIJĮĝgIJŁĻĴŁĻã
ĞĢĻĨĹĞĻĨIJĮğgÌIJĹŃĞĹĤâĤĦĮĽĦğgIJŁĻĴŁĻã

ĴĬĻgĴĬIJĻâXçĻĦĽĻd ÑçĻĦĽĻã
ĴĬĻgĴĬIJĻâXçĻĦĽĻd ĞĢĻĨĹĞĻĨIJĮğgIJŁĻĴŁĻã
ĴĬĻgĽĨIJŃâã
```

First, we are importing matplotlib, then creating a new set of data. Next, we have 4 lines of the code that are the same as the forward pass from our code above. We could call it a prediction or, in the context of what we are going to do, validation. We'll cover both topics and explain what validation and prediction are in the future chapters. For now, it's enough to know that what we are doing here is predicting on the same feature-set that we've used to train the model in order to see what the model learned and returns for our data — seeing how close outputs are to the training ground-true values. We are then plotting the training data, which are obviously a sine, and prediction data, what we'd hope to form a sine as well. Let's run this code again and take a look at the generated image:

Fig 17.02: Model prediction - could not fit the sine data.

Animation of the training process:

Fig 17.03: Model stoppped training immediately.

Anim 17.03: https://nnfs.io/ghi

Recall the rectified linear activation function and how its nonlinear behavior allowed us to map nonlinear functions, but we also needed two or more hidden layers. In this case, we have only 1 hidden layer followed by the output layer. As we should know by now, this is simply not enough!

If we add just one more layer:

```
fi CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĨĮĦçĤĞĻĞâã

fi CĹĦĞĻĦ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ĝ ĨĮĴŁĻ ÌĦĞĻŁĹĦ ĞĮĤ ħǵ IJŁĻĴŁĻ ĿĞĬŁĦĽ
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâĝd ħǵã

fi CĹĦĞĻĦ RĦLU ĞĢĻĨĿĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦĤ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĿĞĻĨIJĮĝ ł AĢĻĨĿĞĻĨIJĮçRĦLUâã

fi CĹĦĞĻĦ ĽĦĢIJĮĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJŁĻĴŁĻ
fi IJÌ ĴĹĦĿĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ħǵ IJŁĻĴŁĻ ĿĞĬŁĦĽ
ĤĦĮĽĦğ ł LĞÑĦĹçDĦĮĽĦâħǵd ħǵã

fi CĹĦĞĻĦ RĦLU ĞĢĻĨĿĞĻĨIJĮ âĻIJ ĠĦ ŁĽĦĤ ŃĨĻÎ DĦĮĽĦ ĬĞÑĦĹãf
ĞĢĻĨĿĞĻĨIJĮğ ł AĢĻĨĿĞĻĨIJĮçRĦLUâã

fi CĹĦĞĻĦ ĻÎĨĹĤ DĦĮĽĦ ĬĞÑĦĹ ŃĨĻÎ ħǵ ĨĮĴŁĻ ÌĦĞĻŁĹĦĽ âĞĽ ŃĦ ĻĞĨĦ IJŁĻĴŁĻ
fi IJÌ ĴĹĦĿĨIJŁĽ ĬĞÑĦĹ ÎĦĹĦã ĞĮĤ ĝ IJŁĻĴŁĻ ĿĞĬŁĦ
ĤĦĮĽĦġ ł LĞÑĦĹçDĦĮĽĦâħǵd ĝã

fi CĹĦĞĻĦ LĨĮĦĞĹ ĞĢĻĨĿĞĻĨIJĮf
ĞĢĻĨĿĞĻĨIJĮġ ł AĢĻĨĿĞĻĨIJĮçLĨĮĦĞĹâã

fi CĹĦĞĻĦ ĬIJĽĽ ÌŁĮĢĻĨIJĮ
ĬIJĽĽçÌŁĮĢĻĨIJĮ ł LIJĽĽçMĦĞĮSĶŁĞĹĦĤEĹĹIJĹâã

fi CĹĦĞĻĦ IJĴĻĨİĨŅĦĹ
IJĴĻĨİĨŅĦĹ ł OĴĻĨİĨŅĦĹçAĤĞİâã

fi AĢĢŁĹĞĢÑ ĴĹĦĢĨĽĨIJĮ ÌIJĹ ĞĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ
fi TĤĦĹĦ ĞĹĦ ĮIJ ĹĦĞĬĬÑ ĞĢĢŁĹĞĢÑ ÌĞĢĻIJĹ ÌIJĹ ĹĦĹĦĽĽĨIJĮ ĴĹIJĠĬĦİd
fi ĠŁĻ ŃĦ ĢĞĮ ĽĨİŁĬĞĻĦwĞĴĴĹIJŃĨİĞĻĦ ĨĻg WĦďĬĬ ĢĞĬĢŁĬĞĻĦ ĨĻ ĠÑ ĢĤĦĢĨĨĮĹ
fi ÎIJŃ İĞĮÑ ĿĞĬŁĦĽ ÎĞĿĦ Ğ ĤĨÌÌĦĹĦĮĢĦ ĻIJ ĻÎĦĨĹ ĹĹIJŁĮĤ ĻĹŁĻÎ ĦĶŁĨĿĞĬĦĮĻ
fi ĬĦĽĽ ĻÎĞĮ ĹĨĿĦĮ ĴĹĦĢĨĽĨIJĮ
fi WĦďĬĬ ĢĞĬĢŁĬĞĻĦ ĻÎĨĽ ĴĹĦĢĨĽĨIJĮ ĞĽ Ğ ÌĹĞĢĻĨIJĮ IJÌ ĽĻĞĮĤĞĹĤ ĤĦĿĨĞĻĨIJĮ
fi IJÌ ĞĬ ĻÎĦ ĹĹIJŁĮĤ ĻĹŁĻÎ ĿĞĬŁĦĽ
ĞĢĢŁĹĞĢÑçĴĹĦĢĨĽĨIJĮ ł ĮĴgĽĻĤâÑã w ğĥfl

fi TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮĹĦâĝflflflĝãf

    fi PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹĤ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮĹ ĤĞĻĞ ĻÎĹIJŁĹÎ ĻÎĨĽ ĬĞÑĦĹ
    ĤĦĮĽĦĝgÌIJĹŃĞĹĤâXã
```

```
ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻÎĿĞĻÎIJĮ ÌŁĮĢĻÎIJĮ
ﬁ ĻĞĪĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌÎĹĽĻ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻÎĿĞĻÎIJĮĝgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJŁĻĴŁĻã

ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ
ﬁ ĻĞĪĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻÎĿĞĻÎIJĮ ÌŁĮĢĻÎIJĮ
ﬁ IJÌ ÌÎĹĽĻ ĬĞÑĦĹ ĞĽ ÎĮĴŁĻĽ
ÂĦĮĽĦğgÌIJĹŃĞĹÂâĞĢĻÎĿĞĻÎIJĮĝgIJŁĻĴŁĻã

ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻÎĿĞĻÎIJĮ ÌŁĮĢĻÎIJĮ
ﬁ ĻĞĪĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻÎĿĞĻÎIJĮğgÌIJĹŃĞĹÂâÂĦĮĽĦğgIJŁĻĴŁĻã

ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĻÎÎĹÂ DĦĮĽĦ ĬĞÑĦĹ
ﬁ ĻĞĪĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻÎĿĞĻÎIJĮ ÌŁĮĢĻÎIJĮ IJÌ ĽĦĢIJĮÂ ĬĞÑĦĹ ĞĽ ÎĮĴŁĻĽ
ÂĦĮĽĦġgÌIJĹŃĞĹÂâĞĢĻÎĿĞĻÎIJĮğgIJŁĻĴŁĻã

ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻÎĿĞĻÎIJĮ ÌŁĮĢĻÎIJĮ
ﬁ ĻĞĪĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ĻÎÎĹÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
ĞĢĻÎĿĞĻÎIJĮġgÌIJĹŃĞĹÂâÂĦĮĽĦġgIJŁĻĴŁĻã

ﬁ CĞĬĢŁĬĞĻĦ ĻÎĦ ÂĞĻĞ ĬIJĽĽ
ÂĞĻĞçĬIJĽĽ ł ĬIJĽĽçÌŁĮĢĻÎIJĮgĢĞĬĢŁĬĞĻĦâĞĢĻÎĿĞĻÎIJĮġgIJŁĻĴŁĻd Ñã

ﬁ CĞĬĢŁĬĞĻĦ ĹĦÍŁĬĞĹÎŃĞĻÎIJĮ ĴĦĮĞĬĻÑ
ĹĦÍŁĬĞĹÎŃĞĻÎIJĮçĬIJĽĽ ł ċ
    ĬIJĽĽçÌŁĮĢĻÎIJĮgĹĦÍŁĬĞĹÎŃĞĻÎIJĮçĬIJĽĽâÂĦĮĽĦĝã κ ċ
    ĬIJĽĽçÌŁĮĢĻÎIJĮgĹĦÍŁĬĞĹÎŃĞĻÎIJĮçĬIJĽĽâÂĦĮĽĦğã κ ċ
    ĬIJĽĽçÌŁĮĢĻÎIJĮgĹĦÍŁĬĞĹÎŃĞĻÎIJĮçĬIJĽĽâÂĦĮĽĦġã

ﬁ CĞĬĢŁĬĞĻĦ IJĿĦĹĞĬĬ ĬIJĽĽ
ĬIJĽĽ ł ÂĞĻĞçĬIJĽĽ κ ĹĦÍŁĬĞĹÎŃĞĻÎIJĮçĬIJĽĽ

ﬁ CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻÎĿĞĻÎIJĮğ ĞĮÂ ĻĞĹÍĦĻĽ
ﬁ TIJ ĢĞĬĢŁĬĞĻĦ ÎĻ ŃĦďĹĦ ĻĞĪÎĮÍ ĞĠĽIJĬŁĻĦ ÂÎÌÌĦĹĦĮĢĦ ĠĦĻŃĦĦĮ
ﬁ ĴĹĦÂÎĢĻÎIJĮĽ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ĿĞĬŁĦĽ ĞĮÂ ĢIJĬĴĞĹĦ ÎÌ ÂÎÌÌĦĹĦĮĢĦĽ
ﬁ ĞĹĦ ĬIJŃĦĹ ĻÎĞĮ ÍÎĿĦĮ ĴĹĦĢÎĽÎIJĮ ĿĞĬŁĦ
ĴĹĦÂÎĢĻÎIJĮĽ ł ĞĢĻÎĿĞĻÎIJĮġgIJŁĻĴŁĻ
ĞĢĢŁĹĞĢÑ ł ĮĴgĬĦĞĮâĮĴgĞĠĽIJĬŁĻĦâĴĹĦÂÎĢĻÎIJĮĽ h Ñã l·
                                ĞĢĢŁĹĞĢÑçĴĹĦĢÎĽÎIJĮã

ÎÌ ĮIJĻ ĦĴIJĢÎ ķ ĝﬂﬂf
    ĴĹÎĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
        ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
        ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď κ
        ÌďÂĞĻĞçĬIJĽĽf ăÂĞĻĞçĬIJĽĽfgġÌåd ď κ
        ÌďĹĦÍçĬIJĽĽf ăĹĦÍŁĬĞĹÎŃĞĻÎIJĮçĬIJĽĽfgġÌåãd ď κ
        ÌďĬĹf ăIJĴĻÎÌÎŃĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮÎĮÍçĹĞĻĦåďã
```

```
    fi BĞĢĪŃĞĹÎ ĴĞĹĹ
    ĬIJĹĹçÌŁIĢĻĨIJIĢġĞĢĪŃĞĹÎâĞĢĻĨĽĞĻĨIJIĢġgIJŁĻĴŁĻd Ñã
    ĞĢĻĨĽĞĻĨIJIġgĠĞĢĪŃĞĹÎâĬIJĹĹçÌŁIĢĻĨIJIĢgÎĨIĴŁĻĹã
    ÎĦIĴĹĦġgĠĞĢĪŃĞĹÎâĞĢĻĨĽĞĻĨIJIĢġgÎĨIĴŁĻĹã
    ĞĢĻĨĽĞĻĨIJIĢğgĠĞĢĪŃĞĹÎâÎĦIĴĹĦġgÎĨIĴŁĻĹã
    ÎĦIĴĹĦğgĠĞĢĪŃĞĹÎâĞĢĻĨĽĞĻĨIJIĢğgÎĨIĴŁĻĹã
    ĞĢĻĨĽĞĻĨIJIĢĝgĠĞĢĪŃĞĹÎâÎĦIĴĹĦğgÎĨIĴŁĻĹã
    ÎĦIĴĹĦĝgĠĞĢĪŃĞĹÎâĞĢĻĨĽĞĻĨIJIĢĝgÎĨIĴŁĻĹã

    fi UĴÎĞĻĦ ŃĦĨÍÎĻĹ ĞĴÎ ĠĨĞĹĦĹ
    IJĴĻĨĬĨŅĦĹgĴĹĦçŁĴÎĞĻĦçĴĞĹĞĬĹâã
    IJĴĻĨĬĨŅĦĹgŁĴÎĞĻĦçĴĞĹĞĬĹâÎĦIĴĹĦĝã
    IJĴĻĨĬĨŅĦĹgŁĴÎĞĻĦçĴĞĹĞĬĹâÎĦIĴĹĦğã
    IJĴĻĨĬĨŅĦĹgŁĴÎĞĻĦçĴĞĹĞĬĹâÎĦIĴĹĦġã
    IJĴĻĨĬĨŅĦĹgĴIJĹĻçŁĴÎĞĻĦçĴĞĹĞĬĹâã


ĨĬĴIJĹĻ ĬĞĻĴĬIJĻĨĨĠgĴÑĴĬIJĻ ĞĹ ĴĬĻ

XçĻĦĹĻd ÑçĻĦĹĻ ł ĹĨĨĮĦçÎĞĻĞâã

ÎĦIĴĹĦĝgÌIJĹŃĞĹÎâXçĻĦĹĻã
ĞĢĻĨĽĞĻĨIJIĢĝgÌIJĹŃĞĹÎâÎĦIĴĹĦĝgIJŁĻĴŁĻã
ÎĦIĴĹĦğgÌIJĹŃĞĹÎâĞĢĻĨĽĞĻĨIJIĢĝgIJŁĻĴŁĻã
ĞĢĻĨĽĞĻĨIJIĢğgÌIJĹŃĞĹÎâÎĦIĴĹĦğgIJŁĻĴŁĻã
ÎĦIĴĹĦġgÌIJĹŃĞĹÎâĞĢĻĨĽĞĻĨIJIĢğgIJŁĻĴŁĻã
ĞĢĻĨĽĞĻĨIJIĢġgÌIJĹŃĞĹÎâÎĦIĴĹĦġgIJŁĻĴŁĻã

ĴĬĻgĴĬIJĻâXçĻĦĹĻd ÑçĻĦĹĻã
ĴĬĻgĴĬIJĻâXçĻĦĹĻd ĞĢĻĨĽĞĻĨIJIĢġgIJŁĻĴŁĻã
ĴĬĻgĹÎIJŃâã


ńńń
ĦĴIJĢÎf fld ĞĢĢf flgflflğd ĬIJĹĹf flgĥflfl âÎĞĻĞçĬIJĹĹf flgĥflfld ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflĝ
ĦĴIJĢÎf ĝflfld ĞĢĢf flgflflġd ĬIJĹĹf flgĝíì âÎĞĻĞçĬIJĹĹf flgĝííd ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflĝ
ggg
ĦĴIJĢÎf îîflfld ĞĢĢf flgħĝìd ĬIJĹĹf flgflġĝ âÎĞĻĞçĬIJĹĹf flgflġĝd ĹĦÍçĬIJĹĹf flgflflfãd
ĬĹf flgflflĝ
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgħğfld ĬIJĹĹf flgflġĝ âÎĞĻĞçĬIJĹĹf flgflġĝd ĹĦÍçĬIJĹĹf flgflflfãd
ĬĹf flgflflĝ
```

Fig 17.04: Model prediction - better fit to the data.

Fig 17.05: Model trained to better fit the sine data.

Anim 17.05: https://nnfs.io/hij

Our model's accuracy is not very good, and loss seems stuck at a pretty high level for this model. The image shows us why this is the case, the model has some trouble fitting our data, and it looks like it might be stuck in a local minimum. As we have already learned, to try to help the model with being stuck at a local minimum, we might use a higher learning rate and add a learning rate decay. In the previous model, we have used the default learning rate, which is *0.001*. Let's set it to *0.01* and add learning rate decaying:

```
IJĴĻĨĬĬĨŅĦĹ ł OĴĻĨĬĬĨŅĦĹçAĤĞĬâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflĝd ĤĦĢĞÑłĝĦhġã


ńńń
ĦĴIJĢĴf fld ĞĢĢf flgflflğd ĬIJĹĹf flgĥflfl âĤĞĻĞçĬIJĹĹf flgĥflfld ĹĦĬçĬIJĹĹf flgflflfld ĬĹf
flgflĝ
ĦĴIJĢĴf ĝflfld ĞĢĢf flgflğìd ĬIJĹĹf flgflĥĝ âĤĞĻĞçĬIJĹĹf flgflĥĝd ĹĦĬçĬIJĹĹf flgflflfld ĬĹf
flgflflîflîîĝíĝflìġìflġġĥí
ggg
ĦĴIJĢĴf îîflfld ĞĢĢf flgĥĥĥd ĬIJĹĹf flgflġĝ âĤĞĻĞçĬIJĹĹf flgflġĝd ĹĦĬçĬIJĹĹf flgflflfld
ĬĹf flgflflflîĝìĥĝĥġĥíġíĝġĝflġ
ĦĴIJĢĴf ĝflflflfld ĞĢĢf flgĥĥġd ĬIJĹĹf flgflġĝ âĤĞĻĞçĬIJĹĹf flgflġĝd ĹĦĬçĬIJĹĹf flgflflfld
ĬĹf flgflflflîflîĝìġĥĥĝġĝíġîġ
```

Fig 17.06: Model prediction - similar fit to the data.

Fig 17.07: Model trained to fit the sine data, similar fit.

Anim 17.07: https://nnfs.io/ijk

Our model seems to still be stuck with even lower accuracy this time. Let's try to use an even bigger learning rate then:

```
IJĴĻĨĬĨŅĦĹ ł OĴĻĨĬĨŅĦĹçAĤĞİâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflĥd ĤĦĢĞÑłĝĦhġã


ńńń
ĦĴIJĢÎf fld ĞĢĢf flgflflğd ĬIJĹĹf flgĥflfl âĤĞĻĞçĬIJĹĹf flgĥflfld ĹĦÍçĬIJĹĹf flgflflfld ĬĹf
flgflĥ
ĦĴIJĢÎf ĝflfld ĞĢĢf flgflíìd ĬIJĹĹf flgflġĝ âĤĞĻĞçĬIJĹĹf flgflġĝd ĹĦÍçĬIJĹĹf flgflflfld ĬĹf
flgflĝĥĝîĥîflĥġĥíĥĝĥíġ
ggg
ĦĴIJĢÎf îîflfld ĞĢĢf flgğìĥd ĬIJĹĹf flgflġĝ âĤĞĻĞçĬIJĹĹf flgflġĝd ĹĦÍçĬIJĹĹf flgflflfld
ĬĹf flgflflĝĥíìĥìĥíĝĝîĝĝĝflĝ
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgğĝîd ĬIJĹĹf flgflġĝ âĤĞĻĞçĬIJĹĹf flgflġĝd ĹĦÍçĬIJĹĹf flgflflfld
ĬĹf flgflflĝĥĝĥíĥìíflĥĝĥĝĝîĥĥ
```

Fig 17.08: Model prediction - similar fit to the data.

Fig 17.09: Model trained to fit the sine data, similar fit.

Anim 17.09: https://nnfs.io/jkl

It's getting even worse. Accuracy drops significantly, and we can observe the lower part of the sine being of a worse shape as well. It seems like we are not able to make this model learn the data, but after multiple tests and tuning hyperparameters, we could find a learning rate of 0.005:

```
IJĴĻĨĬĨŅĦĹ ł OĴĻĨĬĨŅĦĹçAĤĞİâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflflĥd ĤĦĢĞÑłĝĦhġã


ńńń
ĦĴIJĢĴf fld ĞĢĢf flgflfl̇d ĬIJĹĹf flgǵîħ âĤĞĻĞçĬIJĹĹf flgǵîħd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflĥ
ĦĴIJĢĴf ĝflfld ĞĢĢf flgflĝìd ĬIJĹĹf flgflǵí âĤĞĻĞçĬIJĹĹf flgflǵíd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflǵĥǵîĥîflĥǵħíĥĝħíǵ
ggg
ĦĴIJĢĴf îîflfld ĞĢĢf flgîíĝd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflflãd
ĬĹf flgflflflǵĥíìĥìħíǵĝîĝĝĝflĝħ
ĦĴIJĢĴf ĝflflflfld ĞĢĢf flgîíĝd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflflãd
ĬĹf flgflflflǵĥǵĥíħìíflħĝħǵĝîħǵ
```

Fig 17.10: Model prediction - good fit to the data.

Fig 17.11: Model trained to fit the sine data.

Anim 17.11: https://nnfs.io/klm

This time model has learned pretty well, but the curious part is that both lower and higher learning rates than what we used here initially caused accuracy to be pretty low and loss be stuck at the same value when the learning rate in between them actually worked.

Debugging such a problem is usually a pretty hard task and out of the scope of this book. The accuracy and loss suggest that updates to the parameters are not big enough, but the rising learning rate makes things only worse, and there is just this single spot that we were able to find that lets our model learn. You might recall that, back in chapter 3, we were discussing parameter initialization methods and why it's important to initialize them wisely. It turns out that, in the current case, we can help the model learn by changing the factor of 0.01 to 0.1 in the Dense layer's weight initialization. But then you might ask — since the learning rate is being used to decide how much of a gradient to apply to the parameters, why does changing these initial values help instead? As you may recall, the back-propagated gradient is calculated using weights, and the learning rate does not affect it. That's why it's important to use right weight initialization, and so far, we have been using the same values for each model.

If we, for example, take a look at the source code of Keras, a neural network framework, we can learn that:

```
ÂĦÌ ÍĬIJĹIJĻçŁĮĨĨÌIJĹİâĹĦĦÂŁNIJĮĦãf
    ðððGĬIJĹIJĻ ŁĮĨÌIJĹİ ĨĮĨĻĨĞĬĨŅĦĹd ĞĬĹIJ ĢĞĬĬĦÂ XĞŁĨĦĹ ŁĮĨÌIJĹİ ĨĮĨĻĨĞĬĨŅĦĹg

    IĻ ÂĹĞŃĹ ĹĞİĴĬĦĹ ÌĹIJİ Ğ ŁĮĨÌIJĹİ ÂĨĹĻĹĨĠŁĻĨIJĮ ŃĨĻÎĨĮ ähĬĨÌĨĻd ĬĨÌĨĻā
    ŃÎĦĹĦ ẂĬĨÌĨĻẂ ĨĹ ẂĹĶĹĻâħ w âÌĞĮçĨĮ κ ÌĞĮçIJŁĻããẂ
    ŃÎĦĹĦ ẂÌĞĮçĨĮẂ ĨĹ ĻÎĦ ĮŁİĠĦĹ IJÌ ĨĮĴŁĻ ŁĮĨĻĹ ĨĮ ĻÎĦ ŃĦĨÍÎĻ ĻĦĮĹIJĹ
    ĞĮÂ ẂÌĞĮçIJŁĻẂ ĨĹ ĻÎĦ ĮŁİĠĦĹ IJÌ IJŁĻĴŁĻ ŁĮĨĻĹ ĨĮ ĻÎĦ ŃĦĨÍÎĻ ĻĦĮĹIJĹg

    fi AĹÍŁİĦĮĻĹ
        ĹĦĦÂf A PÑĻÎIJĮ ĨĮĻĦÍĦĹg UĹĦÂ ĻIJ ĹĦĦÂ ĻÎĦ ĹĞĮÂIJİ ÍĦĮĦĹĞĻIJĹg

    fi RĦĻŁĹĮĹ
        AĮ ĨĮĨĻĨĞĬĨŅĦĹg

    fi RĦÌĦĹĦĮĢĦĹ
        GĬIJĹIJĻ Ţ BĦĮÍĨIJd AISTATS ğflĝfl
        ÎĻĻĴfwwÏİĬĹgIJĹÍwĴĹIJĢĦĦÂĨĮÍĹwĴĞĴĦĹĹwŁîwÍĬIJĹIJĻĝflĞwÍĬIJĹIJĻĝflĞgĴÂÌ
    ððð
    ĹĦĻŁĹĮ VĞĹĨĞĮĢĦSĢĞĬĨĮÍâĹĢĞĬĦŁĝgd
                            İIJÂĦŁďÌĞĮçĞŁÍďd
                            ÂĨĹĻĹĨĠŁĻĨIJĮŁďŁĮĨÌIJĹİďd
                            ĹĦĦÂŁĹĦĦÂã
```

This code is part of the Keras 2 library. The important part of the above is actually the comment section, which describes how it initializes weights. We can find there are important pieces of information to remember — the fraction that multiplies the draw from the uniform distribution depends on the number of inputs and the number of neurons and is not constant like in our case. This method of initialization is called **Glorot uniform**. We (the authors of this book) actually have had a very similar problem in one of our projects, and changing the way weights were initialized changed the model from not learning at all to a learning state.

For the purposes of this model, let's change the factor multiplying the draw from the normal distribution in the weight initialization of the *Dense* layer to 0.1 and re-run all four of the above attempts to compare results:

```
ĹĦĬÌgŃĦĨÍÎĻĹ ł flgĝ ǻ ĮĴgĹĞĮÂIJİgĹĞĮÂĮâĮçĨĮĴŁĻĹd ĮçĮĦŁĹIJĮĹã
```

And all above tests re-ran:

```
IJĴĻĨĬĨÑŃĦĹ ł OĴĻĨĬĨÑŃĦĹçAĤĞİâã


ńńń
ĦĴIJĢĴf ﬁd ĞĢĢf ﬁgﬁﬁġd ĬIJĹĹf ﬁgǵîħ âĤĞĻĞçĬIJĹĹf ﬁgǵîħd ĹĦĺçĬIJĹĹf ﬁgﬁﬁﬁãd ĬĹf
ﬁgﬁﬁĝ
ĦĴIJĢĴf ĝﬁﬁd ĞĢĢf ﬁgﬁﬁĥd ĬIJĹĹf ﬁgĝĝǵ âĤĞĻĞçĬIJĹĹf ﬁgĝĝǵd ĹĦĺçĬIJĹĹf ﬁgﬁﬁﬁãd ĬĹf
ﬁgﬁﬁĝ
ggg
ĦĴIJĢĴf îîﬁﬁd ĞĢĢf ﬁgíħîd ĬIJĹĹf ﬁgﬁﬁﬁ âĤĞĻĞçĬIJĹĹf ﬁgﬁﬁﬁd ĹĦĺçĬIJĹĹf ﬁgﬁﬁﬁãd
ĬĹf ﬁgﬁﬁĝ
ĦĴIJĢĴf ĝﬁﬁﬁﬁd ĞĢĢf ﬁgííġd ĬIJĹĹf ﬁgﬁﬁﬁ âĤĞĻĞçĬIJĹĹf ﬁgﬁﬁﬁd ĹĦĺçĬIJĹĹf ﬁgﬁﬁﬁãd
ĬĹf ﬁgﬁﬁĝ
```

Fig 17.12: Model prediction - good fit to the data with different weight initialization.

Fig 17.13: Model trained to fit the sine data after replacing weight initialization.

Anim 17.13: https://nnfs.io/lmn

This model was previously stuck and has now achieved high accuracy. There are some visible imperfections like at the bottom side of this sine, but the overall result is better.

```
IJĴĻĨĬĨŅĦĹ ł OĴĻĨĬĨŅĦĹçAĤĞİâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflĝd ĤĦĢĞÑłĝĦhġã


ńńń
ĦĴIJĢĴf fld ĞĢĢf flgflflġd ĬIJĹĹf flgġîħ âĤĞĻĞçĬIJĹĹf flgġîħd ĹĦĬçĬIJĹĹf flgflflflãd ĬĹf
flgflĝ
ĦĴIJĢĴf ĝflfld ĞĢĢf flgflĦĥd ĬIJĹĹf flgflĝĝ âĤĞĻĞçĬIJĹĹf flgflĝĝd ĹĦĬçĬIJĹĹf flgflflflãd ĬĹf
flgflflîflîîĝíĝflìġìflġġĦí
ggg
ĦĴIJĢĴf îîflfld ĞĢĢf flgîĥíd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦĬçĬIJĹĹf flgflflflãd
ĬĹf flgflflflîĝìĥĝĥġĦíġíġġĝflġ
```

```
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgîġîd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflflãd
ĬĹf flgflflflîflîĝìġĤħĝĝĝĝíġîġ
```

Fig 17.14: Model prediction - good fit to the data with different weight initialization.

Fig 17.15: Model trained to fit the sine data after replacing weight initialization.

Anim 17.15: https://nnfs.io/mno

Another previously stuck model has trained very well this time, achieving very good accuracy.

```
IJĴĻĨĬĬĨŅĦĹ ł OĴĻĨĬĬĨŅĦĹçAĤĞĬâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflĥd ĤĦĢĞÑłĝĦhġã


ńńń
ĦĴIJĢĴf fld ĞĢĢf flgflfġd ĬIJĹĹf flgġîħ âĤĞĻĞçĬIJĹĹf flgġîħd ĹĦĬçĬIJĹĹf flgflflfãd ĬĹf
flgflĥ
ĦĴIJĢĴf ĝflfld ĞĢĢf flgflĝħd ĬIJĹĹf flgflflí âĤĞĻĞçĬIJĹĹf flgflflíd ĹĦĬçĬIJĹĹf flgflflfãd ĬĹf
flgflġĥġîĥîflĥġħíĥĝħíġ
ggg
ĦĴIJĢĴf îflflfld ĞĢĢf flgíflğd ĬIJĹĹf flgflflf âĤĞĻĞçĬIJĹĹf flgflflfd ĹĦĬçĬIJĹĹf flgflflfãd
ĬĹf flgflflĥflflflĥflflflĥflflflĥflflĝ
ĦĴIJĢĴf îĝflfld ĞĢĢf flgğġġd ĬIJĹĹf flgflflf âĤĞĻĞçĬIJĹĹf flgflflfd ĹĦĬçĬIJĹĹf flgflflfãd
```

```
ĬĹf flgflflġîĥflîíĥğġħflħġîħì
ĦĴIJĢĴf îğflfld ĞĢĢf flgġġġd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfl̃ad
ĬĹf flgflflġîflğġġġġĥíğĥflíĝ
ĦĴIJĢĴf îġflfld ĞĢĢf flgíġíd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfl̃ad
ĬĹf flgflflġíĥġíġflğìĥìĥġîğíĥ
ĦĴIJĢĴf îġflfld ĞĢĢf flgġflîd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfl̃ad
ĬĹf flgflflġíflíĝĥġħġflğĥğîflî
ĦĴIJĢĴf îĥflfld ĞĢĢf flgğĥġd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfl̃ad
ĬĹf flgflflġìħğġĥíġĝîíġîîíĥ
ĦĴIJĢĴf îħflfld ĞĢĢf flgìîĥd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfl̃ad
ĬĹf flgflflġìĝìġğħĝìğğíflġflġ
ĦĴIJĢĴf îìflfld ĞĢĢf flgíflğd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfl̃ad
ĬĹf flgflflġħìġġġġîĥħġġġĥğí
ĦĴIJĢĴf îíflfld ĞĢĢf flgĝġĝd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfl̃ad
ĬĹf flgflflġħġflflĥíġġíìġĥflħî
ĦĴIJĢĴf îîflfld ĞĢĢf flgğğĝd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfl̃ad
ĬĹf flgflflġĥíìĥìħíġĝîĝğĝflĝ
ĦĴIJĢĴf ĝflflflfld ĞĢĢf flgħġĝd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfl̃ad
ĬĹf flgflflġĥġĥíħìíflħĝħġĝîħĥ
```

Fig 17.16: Model prediction - good fit to the data with different weight initialization.

Fig 17.17: Model trained to fit the sine data after replacing weight initialization.

Anim 17.17: https://nnfs.io/nop

The "jumping" accuracy in the case of this set of the optimizer settings shows that the learning rate is way too big, but even then, the model learned the shape of the sine function considerably well.

```
IJĴĻĨĬĬĨŅĦĹ ł OĴĻĨĬĬĨŅĦĹçAĤĞİâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflflĥd ĤĦĢĞÑłĝĦhġã


ńńń
ĦĴIJĢĴf fld ĞĢĢf flgflflġd ĬIJĹĹf flgǵîħ âĤĞĻĞçĬIJĹĹf flgǵîħd ĹĦĺçĬIJĹĹf flgflflflãd ĬĹf
flgflflĥ
ĦĴIJĢĴf ĝflfld ĞĢĢf flgflĝìd ĬIJĹĹf flgflǵí âĤĞĻĞçĬIJĹĹf flgflǵíd ĹĦĺçĬIJĹĹf flgflflflãd ĬĹf
flgflflǵĥǵîĥîflĥĝħíĥĝħíǵ
ĦĴIJĢĴf ğflfld ĞĢĢf flgğǵğd ĬIJĹĹf flgflflĝ âĤĞĻĞçĬIJĹĹf flgflflĝd ĹĦĺçĬIJĹĹf flgflflflãd ĬĹf
flgflflǵĝìflĝǵĝìíǵíğflħíǵ
```

```
ĦĴIJĢÎf ġflfld ĞĢĢf flgìíħd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflġíġîĝĝġìflġħĝíĝħí
ĦĴIJĢÎf ġflfld ĞĢĢf flgííĥd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflġĥìġîíĝġĝĥĞîħħġflġ
ggg
ĦĴIJĢÎf îîflfld ĞĢĢf flgîíğd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfãd
ĬĹf flgflflfġĥíìĥìħíġĝîĝĞĝflĝħ
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgîíĝd ĬIJĹĹf flgflflfl âĤĞĻĞçĬIJĹĹf flgflflfld ĹĦÍçĬIJĹĹf flgflflfãd
ĬĹf flgflflfġĥġĥíħìíflħĝħġĝîħġ
```

Fig 17.18: Model prediction - best fit to the data with different weight initialization.

Fig 17.19: Model trained to best fit the sine data after replacing weight initialization.

Anim 17.19: https://nnfs.io/opq

These hyperparameters yielded the best results again, but not by much.

As we can see, this time, our model learned in all cases, using different learning rates, and did not get stuck if any of them. That's how much changing weight initialization can impact the training process.

Full code up to this point:

```
ĨĬĴIJĹĻ ĮŁĬĴÑ ĞĽ ĮĴ
ĨĬĴIJĹĻ ĮĮÌĽ
ÌĹIJĬ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĨĬĴIJĹĻ ĽĨĮĦçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


ſi DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ſi LĞÑĦĹ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ
    ÎĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽd
                ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝłfld ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłfld
                ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝłfld ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłflãf
        ſi IĮĨĻĨĞĬĨŅĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍÎĻĽ ł flgĝ ấ ĮĴgĹĞĮĤIJĠgĹĞĮĤĮâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã
        ſi SĦĻ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĽĻĹĦĮÍĻÎ
        ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ł ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ
        ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ł ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ
        ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ł ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ
        ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ł ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ

    ſi FIJĹŃĞĹĤ ĴĞĽĽ
    ÎĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ſi RĦİĦİĠĦĹ ĨĮĴŁĻ ĽĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ſi CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ĽĞĬŁĦĽ ÌĹIJĬ ĨĮĴŁĻĽd ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĽĦĬÌgŃĦĨÍÎĻĽã ĸ ĽĦĬÌgĠĨĞĽĦĽ

    ſi BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÎĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĽĞĬŁĦĽãf
        ſi GĹĞĤĨĦĮĻĽ IJĮ ĴĞĹĞĬĦĻĦĹĽ
        ĽĦĬÌgĤŃĦĨÍÎĻĽ ł ĮĴgĤIJĻâĽĦĬÌgĨĮĴŁĻĽgTd ĤĽĞĬŁĦĽã
        ĽĦĬÌgĤĠĨĞĽĦĽ ł ĮĴgĽŁĬâĤĽĞĬŁĦĽd ĞŃĨĽłfld ĪĦĦĴĤĨĬĽłTĹŁĦã
```

```
        ńı GĹĞĤĨĦĮĻĽ ĲĮ ĹĦÍŁĬĞĹĨŅĞĻĨĲĮ
        ńı Lĝ ĲĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń ńf
            ĤLĝ ł ĮĴgĲĮĦĽçĬĨĨĦâĽĦĬÌgŃĦĨÍÎĻĽã
            ĤLĝäĽĦĬÌgŃĦĨÍÎĻĽ l· ńā ł hĝ
            ĽĦĬÌgĤŃĦĨÍÎĻĽ κł ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ś ĤLĝ
        ńı Lğ ĲĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ńf
            ĽĦĬÌgĤŃĦĨÍÎĻĽ κł ğ ś ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ś ċ
                                ĽĦĬÌgŃĦĨÍÎĻĽ
        ńı Lĝ ĲĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń ńf
            ĤLĝ ł ĮĴgĲĮĦĽçĬĨĨĦâĽĦĬÌgĠĨĞĽĦĽã
            ĤLĝäĽĦĬÌgĠĨĞĽĦĽ l· ńā ł hĝ
            ĽĦĬÌgĤĠĨĞĽĦĽ κł ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ś ĤLĝ
        ńı Lğ ĲĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ńf
            ĽĦĬÌgĤĠĨĞĽĦĽ κł ğ ś ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ś ċ
                                ĽĦĬÌgĠĨĞĽĦĽ

        ńı GĹĞĤĨĦĮĻ ĲĮ L·ĞĬŁĦĽ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĮĴgĤĲĻâĤL·ĞĬŁĦĽd ĽĦĬÌgŃĦĨÍÎĻĽgTã


ńı DĹĲĴĲŁĻ
ĢĬĞĽĽ LĞÑĦĹçDĹĲĴĲŁĻf

    ńı IĮĨĻ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĹĞĻĦãf
        ńı SĻĲĹĦ ĹĞĻĦd ŃĦ ĨĮL·ĦĹĻ ĨĻ ĞĽ ÌĲĹ ĦŃĞĬĴĬĦ ÌĲĹ ĤĹĲĴĲŁĻ
        ńı ĲÌ ńgĝ ŃĦ ĮĦĦĤ ĽŁĢĢĦĽĽ ĹĞĻĦ ĲÌ ńgî
        ĽĦĬÌgĹĞĻĦ ł ĝ h ĹĞĻĦ

    ńı FĲĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌĲĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        ńı SĞL·Ħ ĨĮĴŁĻ L·ĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ńı GĦĮĦĹĞĻĦ ĞĮĤ ĽĞL·Ħ ĽĢĞĬĦĤ ĬĞĽĨ
        ĽĦĬÌgĠĨĮĞĹÑçĬĞĽĨ ł ĮĴgĹĞĮĤĲĬgĠĨĮĲĬĨĞĬâĝd ĽĦĬÌgĹĞĻĦd
                            ĽĨĢĦłĨĮĴŁĻĽgĽĨĞĴĦã w ĽĦĬÌgĹĞĻĦ
        ńı AĴĴĬÑ ĬĞĽĨ ĻĲ ĲŁĻĴŁĻ L·ĞĬŁĦĽ
        ĽĦĬÌgĲŁĻĴŁĻ ł ĨĮĴŁĻĽ ś ĽĦĬÌgĠĨĮĞĹÑçĬĞĽĨ

    ńı BĞĢĨŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤL·ĞĬŁĦĽãf
        ńı GĹĞĤĨĦĮĻ ĲĮ L·ĞĬŁĦĽ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĤL·ĞĬŁĦĽ ś ĽĦĬÌgĠĨĮĞĹÑçĬĞĽĨ

ńı RĦLU ĞĢĻĨL·ĞĻĨĲĮ
```

```
ĢĬĞĽĹ AĢĻĨĿĞĻĨIJIçRĦLUf

    ħ FIJĹŃĞĹĤ ĴĞĽĹ
    ÂĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ĨĮĴŁĻĹãf
        ħ RĦİĦİĠĦĹ ĨĮĴŁĻ ĿĞĬŁĦĹ
        ĹĦĬÌgĨĮĴŁĻĹ ł ĨĮĴŁĻĹ
        ħ CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ĿĞĬŁĦĹ ÌĹIJİ ĨĮĴŁĻĹ
        ĹĦĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĨİŁİâfld ĨĮĴŁĻĹã

    ħ BĞĢĪŃĞĹĤ ĴĞĽĹ
    ÂĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ÂĿĞĬŁĦĹãf
        ħ SĨĮĢĦ ŃĦ ĮĦĦĤ ĻIJ İIJĤĨÌÌÑ IJĹĨĬĬĮĞĬ ĿĞĹĨĞĠĬĦd
        ħ ĬĦĻďĹ İĞĪĦ Ğ ĢIJĴÑ IJÌ ĿĞĬŁĦĹ ÌĨĹĹĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ÂĿĞĬŁĦĹgĢIJĴÑâã

        ħ ZĦĹIJ ĬĹĞĤĨĦĮĻ ŃÎĦĹĦ ĨĮĴŁĻ ĿĞĬŁĦĹ ŃĦĹĦ ĮĦĬĞĻĨĿĦ
        ĹĦĬÌgĤĨĮĴŁĻĹäĹĦĬÌgĨĮĴŁĻĹ l·ł flã ł fl


ħ SIJÌĻİĞŃ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĹ AĢĻĨĿĞĻĨIJIçSIJÌĻİĞŃf

    ħ FIJĹŃĞĹĤ ĴĞĽĹ
    ÂĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ĨĮĴŁĻĹãf
        ħ RĦİĦİĠĦĹ ĨĮĴŁĻ ĿĞĬŁĦĹ
        ĹĦĬÌgĨĮĴŁĻĹ ł ĨĮĴŁĻĹ

        ħ GĦĻ ŁĮĮIJĹİĞĬĨĨŃĦĤ ĴĹIJĠĞĠĨĬĬĻĨĦĹ
        ĦŃĴçĿĞĬŁĦĹ ł ĮĴgĦŃĴâĨĮĴŁĻĹ h ĮĴgİĞŃâĨĮĴŁĻĹd ĞŃĨĹłĝd
                                                ĪĦĦĴĤĨİĹłTĹŁĦãã
        ħ NIJĹİĞĬĨŃĦ ĻĨĦİ ÌIJĹ ĦĞĢĨ ĹĞİĴĬĦ
        ĴĹIJĠĞĠĨĬĬĻĨĦĹ ł ĦŃĴçĿĞĬŁĦĹ w ĮĴgĹŁİâĦŃĴçĿĞĬŁĦĹd ĞŃĨĹłĝd
                                                ĪĦĦĴĤĨİĹłTĹŁĦã

        ĹĦĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĨĬĬĻĨĦĹ

    ħ BĞĢĪŃĞĹĤ ĴĞĽĹ
    ÂĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ÂĿĞĬŁĦĹãf

        ħ CĹĦĞĻĦ ŁĮĨĮĨĻĨĞĬĨŃĦĤ ĞĹĹĞŃ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĮĴgĦİĴĻŃçĬĨĨĦâÂĿĞĬŁĦĹã

        ħ EĮŁİĦĹĞĻĦ IJŁĻĴŁĻĹ ĞĮĤ ĹĹĞĤĨĦĮĻĹ
        ÌIJĹ ĨĮĤĦŃd âĹĨĮĹĬĦçIJŁĻĴŁĻd ĹĨĮĹĬĦçÂĿĞĬŁĦĹã ĨĮ ċ
                ĦĮŁİĦĹĞĻĦâŃĨĴâĹĦĬÌgIJŁĻĴŁĻd ÂĿĞĬŁĦĹããf
            ħ FĬĞĻĻĦĮ IJŁĻĴŁĻ ĞĹĹĞŃ
            ĹĨĮĹĬĦçIJŁĻĴŁĻ ł ĹĨĮĹĬĦçIJŁĻĴŁĻgĹĦĹÎĞĴĦâhĝd ĝã
```

```
            # CĞĬĢŁĬĞĻĦ JĞĢIJĠĨĞĮ İĞĻĹĨŇ IJÌ ĻÎĦ IJŁĻĴŁĻ
            ÏĞĢIJĠĨĞĮçİĞĻĹĨŇ ł ĮĴgĤĬĞĹÌĬĞĻâĽĨĮĹĬĦçIJŁĻĴŁĻã h ċ
                              ĮĴgĤIJĻâĽĨĮĹĬĦçIJŁĻĴŁĻd ĽĨĮĹĬĦçIJŁĻĴŁĻgTã
            # CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦhŃĨĽĦ ĹĹĞĤĨĦĮĻ
            # ĞĮĤ ĞĤĤ ĨĻ ĻIJ ĻÎĦ ĞĹĹĞŇ IJÌ ĽĞİĴĬĦ ĹĹĞĤĨĦĮĻĽ
            ĽĦĬÌgĤĨĮĴŁĻĽäĨĮĤĦŇã ł ĮĴgĤIJĻâÏĞĢIJĠĨĞĮçİĞĻĹĨŇd
                                          ĽĨĮĹĬĦçĤŁĞĬŁĦĽã


# SĨĹİIJĨĤ ĞĢĻĨŁĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨŁĞĻĨIJĮçSĨĹİIJĨĤf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        # SĞŁĦ ĨĮĴŁĻ ĞĮĤ ĢĞĬĢŁĬĞĻĦwĽĞŁĦ IJŁĻĴŁĻ
        # IJÌ ĻÎĦ ĽĨĹİIJĨĤ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĝ w âĝ κ ĮĴgĦŃĴâhĨĮĴŁĻĽãã

    # BĞĢĨŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤŁĞĬŁĦĽãf
        # DĦĹĨŁĞĻĨŁĦ h ĢĞĬĢŁĬĞĻĦĽ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĻÎĦ ĽĨĹİIJĨĤ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĤŁĞĬŁĦĽ ő âĝ h ĽĦĬÌgIJŁĻĴŁĻã ő ĽĦĬÌgIJŁĻĴŁĻ


# LĨĮĦĞĹ ĞĢĻĨŁĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨŁĞĻĨIJĮçLĨĮĦĞĹf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽãf
        # JŁĽĻ ĹĦİĦİĠĦĹ ŁĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽ

    # BĞĢĨŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤŁĞĬŁĦĽãf
        # ĤĦĹĨŁĞĻĨŁĦ ĨĽ ĝd ĝ ő ĤŁĞĬŁĦĽ ł ĤŁĞĬŁĦĽ h ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĤŁĞĬŁĦĽgĢIJĴÑãã


# SGD IJĴĻĨİĨŅĦĹ
ĢĬĞĽĽ OĴĻĨİĨŅĦĹçSGDf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĹĽd
    # ĬĦĞĹĮĨĮĹ ĹĞĻĦ IJÌ ĝg ĨĽ ĤĦÌĞŁĬĻ ÌIJĹ ĻÎĨĽ IJĴĻĨİĨŅĦĹ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮĹçĹĞĻĦłĝgd ĤĦĢĞÑłflgd İIJİĦĮĻŁİłflgãf
        ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
```

```
    ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
    ĽĦĬÌgİIJİĦĮĻŁİ ł ÌIJİĦĮĻŁİ

ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
ÂĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãƒ
    ĨÌ ĽĦĬÌgĤĦĢĞÑƒ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ƒ ċ
            âĝg w âĝg κ ĽĦĬÌgĤĦĢĞÑ ƒ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

ƒ UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
ÂĦÌ ŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãƒ

    ƒ IÌ ŃĦ ŁĽĦ ÌIJİĦĮĻŁİ
    ĨÌ ĽĦĬÌgİIJİĦĮĻŁİƒ

        ƒ IÌ ĬĞÑĦĹ ÂIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ÌIJİĦĮĻŁİ ĞĹĹĞÑĽd ĢĹĦĞĻĦ ĻĨĦİ
        ƒ ÌĨĬĬĦÂ ŃĨĻĨ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ĨĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĬĨĨĻçİIJİĦĮĻŁİĽďãƒ
            ĬĞÑĦĹgŃĦĨĬĨĨĻçİIJİĦĮĻŁİĽ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgŃĦĨĬĨĨĻĽã
            ƒ IÌ ĻĨĦĹĦ ĨĽ ĮIJ İIJİĦĮĻŁİ ĞĹĹĞÑ ÌIJĹ ŃĦĨĬĨĨĻĽ
            ƒ TĨĦ ĞĹĹĞÑ ÂIJĦĽĮďĻ ĦŃĨĽĻ ÌIJĹ ĠĨĞĽĦĽ ÑĦĻ ĦĨĻĨĦĹg
            ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻŁİĽ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgĠĨĞĽĦĽã

        ƒ BŁĨĬĬĤ ŃĦĨĬĨĨĻ ŁĴĤĞĻĦĽ ŃĨĻĨ İIJİĦĮĻŁİ h ĻĞĨĦ ĴĹĦŁĨIJŁĽ
        ƒ ŁĴĤĞĻĦĽ İŁĬĻĨĴĬĨĦĤ ĠÑ ĹĦĻĞĨĮ ÌĞĢĻIJĹ ĞĮĤ ŁĴĤĞĻĦ ŃĨĻĨ
        ƒ ĢŁĹĹĦĮĻ ĹĹĞĤĨĦĮĻĽ
        ŃĦĨĬĨĨĻçŁĴĤĞĻĦĽ ł ċ
            ĽĦĬÌgİIJİĦĮĻŁİ ƒ ĬĞÑĦĹgŃĦĨĬĨĨĻçİIJİĦĮĻŁİĽ h ċ
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ƒ ĬĞÑĦĹgĤŃĦĨĬĨĨĻĽ
        ĬĞÑĦĹgŃĦĨĬĨĨĻçİIJİĦĮĻŁİĽ ł ŃĦĨĬĨĨĻçŁĴĤĞĻĦĽ

        ƒ BŁĨĬĬĤ ĠĨĞĽ ŁĴĤĞĻĦĽ
        ĠĨĞĽçŁĴĤĞĻĦĽ ł ċ
            ĽĦĬÌgİIJİĦĮĻŁİ ƒ ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻŁİĽ h ċ
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ƒ ĬĞÑĦĹgĤĠĨĞĽĦĽ
        ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻŁİĽ ł ĠĨĞĽçŁĴĤĞĻĦĽ

    ƒ VĞĮĨĬĬĞ SGD ŁĴĤĞĻĦĽ âĞĽ ĠĦÌIJĹĦ İIJİĦĮĻŁİ ŁĴĤĞĻĦã
    ĦĬĽĦƒ
        ŃĦĨĬĨĨĻçŁĴĤĞĻĦĽ ł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ƒ ċ
                            ĬĞÑĦĹgĤŃĦĨĬĨĨĻĽ
        ĠĨĞĽçŁĴĤĞĻĦĽ ł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ƒ ċ
                           ĬĞÑĦĹgĤĠĨĞĽĦĽ

    ƒ UĴĤĞĻĦ ŃĦĨĬĨĨĻĽ ĞĮĤ ĠĨĞĽĦĽ ŁĽĨĮĺ ĦĨĻĨĦĹ
    ƒ ŁĞĮĨĬĬĞ IJĹ İIJİĦĮĻŁİ ŁĴĤĞĻĦĽ
    ĬĞÑĦĹgŃĦĨĬĨĨĻĽ κł ŃĦĨĬĨĨĻçŁĴĤĞĻĦĽ
    ĬĞÑĦĹgĠĨĞĽĦĽ κł ĠĨĞĽçŁĴĤĞĻĦĽ
```

```
    fi CĞĬĬ IJĮĢH ĞÌĻHĹ ĞĮÑ ĴĞĹĞİHĻHĹ ŁĴĤĞĻHĽ
    ĤHÌ ĴIJĽĻçŁĴĤĞĻHçĴĞĹĞİĽâĽHĬÌãf
        ĽHĬÌgĨĻHĹĞĻĨIJĮĽ κł ĝ


fi AĤĞÍĹĞĤ IJĴĻĨİĨŅHĹ
ĢĬĞĽĽ OĴĻĨİĨŅHĹçAĤĞÍĹĞĤf

    fi IĮĨĻĨĞĬĨŅH IJĴĻĨİĨŅHĹ h ĽHĻ ĽHĻĻĨĮÍĽ
    ĤHÌ ççĨĮĨĻççâĽHĬÌd ĬHĞĹĮĨĮÍçĹĞĻHłĝgd ĤHĢĞÑłflgd HĴĽĨĬIJĮłĝHhìãf
        ĽHĬÌgĬHĞĹĮĨĮÍçĹĞĻH ł ĬHĞĹĮĨĮÍçĹĞĻH
        ĽHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ł ĬHĞĹĮĨĮÍçĹĞĻH
        ĽHĬÌgĤHĢĞÑ ł ĤHĢĞÑ
        ĽHĬÌgĨĻHĹĞĻĨIJĮĽ ł fl
        ĽHĬÌgHĴĽĨĬIJĮ ł HĴĽĨĬIJĮ

    fi CĞĬĬ IJĮĢH ĠHÌIJĹH ĞĮÑ ĴĞĹĞİHĻHĹ ŁĴĤĞĻHĽ
    ĤHÌ ĴĹHçŁĴĤĞĻHçĴĞĹĞİĽâĽHĬÌãf
        ĨÌ ĽHĬÌgĤHĢĞÑf
            ĽHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ł ĽHĬÌgĬHĞĹĮĨĮÍçĹĞĻH ắ ċ
                âĝg w âĝg κ ĽHĬÌgĤHĢĞÑ ắ ĽHĬÌgĨĻHĹĞĻĨIJĮĽãã

    fi UĴĤĞĻH ĴĞĹĞİHĻHĹĽ
    ĤHÌ ŁĴĤĞĻHçĴĞĹĞİĽâĽHĬÌd ĬĞÑHĹãf

        fi IÌ ĬĞÑHĹ ĤIJHĽ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĤH ĞĹĹĞÑĽd
        fi ĢĹHĞĻH ĻĤHİ ÌĨĬĬHĤ ÑĨĻĤ ŅHĹIJĽ
        ĨÌ ĮIJĻ ĤĞĽĞĻĻĹâĬĞÑHĹd ďŃHĨÍĤĻçĢĞĢĤHďãf
            ĬĞÑHĹgŃHĨÍĤĻçĢĞĢĤH ł ĮĴgŅHĹIJĽçĬĨĬHâĬĞÑHĹgŃHĨÍĤĻĽã
            ĬĞÑHĹgĠĨĞĽçĢĞĢĤH ł ĮĴgŅHĹIJĽçĬĨĬHâĬĞÑHĹgĠĨĞĽHĽã

        fi UĴĤĞĻH ĢĞĢĤH ŃĨĻĤ ĽĶŁĞĹHĤ ĢŁĹĹHĮĻ ÍĹĞĤĨHĮĻĽ
        ĬĞÑHĹgŃHĨÍĤĻçĢĞĢĤH κł ĬĞÑHĹgĤŃHĨÍĤĻĽááğ
        ĬĞÑHĹgĠĨĞĽçĢĞĢĤH κł ĬĞÑHĹgĤĠĨĞĽHĽááğ

        fi VĞĮĨĬĬĞ SGD ĴĞĹĞİHĻHĹ ŁĴĤĞĻH κ IJIJĹİĞĬĨŅĞĻĨIJĮ
        fi ŃĨĻĤ ĽĶŁĞĹH ĹIJIJĻHĤ ĢĞĢĤH
        ĬĞÑHĹgŃHĨÍĤĻĽ κł hĽHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ắ ċ
                         ĬĞÑHĹgĤŃHĨÍĤĻĽ w ċ
                         âĮĴgĽĶĹĻâĬĞÑHĹgŃHĨÍĤĻçĢĞĢĤHã κ ĽHĬÌgHĴĽĨĬIJĮã
        ĬĞÑHĹgĠĨĞĽHĽ κł hĽHĬÌgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻH ắ ċ
                        ĬĞÑHĹgĤĠĨĞĽHĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑHĹgĠĨĞĽçĢĞĢĤHã κ ĽHĬÌgHĴĽĨĬIJĮã

    fi CĞĬĬ IJĮĢH ĞÌĻHĹ ĞĮÑ ĴĞĹĞİHĻHĹ ŁĴĤĞĻHĽ
    ĤHÌ ĴIJĽĻçŁĴĤĞĻHçĴĞĹĞİĽâĽHĬÌãf
        ĽHĬÌgĨĻHĹĞĻĨIJĮĽ κł ĝ
```

```
ƒ RMSĴĹIJĴ IJĴĻĨĬĨŅĦĹ
ĢĬĞĽĹ OĴĻĨĬĨŅĦĹçRMSĴĹIJĴf

    ƒ IĮĨĻĨĞĬĨŅĦ IJĴĻĨĬĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĹĹ
    ÎĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮĹçĹĞĻĦłflgflflĝd ÎĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìd
                ĹÎIJłflgîãf
        ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgÎĦĢĞÑ ł ÎĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬÌgĹÎIJ ł ĹÎIJ

    ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴÎĞĻĦĽ
    ÎĦÌ ĴĹĦçłĴÎĞĻĦçĴĞĹĞĬĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgÎĦĢĞÑf
            ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĽĦĬÌgÎĦĢĞÑ ắ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    ƒ UĴÎĞĻĦ ĴĞĹĞĬĦĻĦĹĽ
    ÎĦÌ łĴÎĞĻĦçĴĞĹĞĬĽâĽĦĬÌd ĬĞÑĦĹãf

        ƒ IÌ ĬĞÑĦĹ ÎIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢÎĦ ĞĹĹĞÑĽd
        ƒ ĢĹĦĞĻĦ ĻÎĦĬ ÌĨĬĬĦÎ ŃĨĻÎ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ÎĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĬÎĻçĢĞĢÎĦďãf
            ĬĞÑĦĹgŃĦĨĬÎĻçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨĬÎĻĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

        ƒ UĴÎĞĻĦ ĢĞĢÎĦ ŃĨĻÎ ĽĶŁĞĹĦÎ ĢŁĹĹĦĮĻ ĬĹĞÎĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨĬÎĻçĢĞĢÎĦ ł ĽĦĬÌgĹÎIJ ắ ĬĞÑĦĹgŃĦĨĬÎĻçĢĞĢÎĦ κ ċ
            âĝ h ĽĦĬÌgĹÎIJã ắ ĬĞÑĦĹgÎŃĦĨĬÎĻĽắắĝ
        ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĽĦĬÌgĹÎIJ ắ ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ κ ċ
            âĝ h ĽĦĬÌgĹÎIJã ắ ĬĞÑĦĹgÎĠĨĞĽĦĽắắĝ

        ƒ VĞĮĨĬĬĞ SGD ĴĞĹĞĬĦĻĦĹ łĴÎĞĻĦ κ ĮIJĹĬĞĬĨŅĞĻĨIJĮ
        ƒ ŃĨĻÎ ĽĶŁĞĹĦ ĹIJIJĻĦÎ ĢĞĢÎĦ
        ĬĞÑĦĹgŃĦĨĬÎĻĽ κł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ắ ċ
                        ĬĞÑĦĹgÎŃĦĨĬÎĻĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨĬÎĻçĢĞĢÎĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ắ ċ
                       ĬĞÑĦĹgÎĠĨĞĽĦĽ w ċ
                       âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã

    ƒ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴÎĞĻĦĽ
    ÎĦÌ ĴIJĽĻçłĴÎĞĻĦçĴĞĹĞĬĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ
```

```
# AĤĞİ IJĴĻĨİĨŅĦĹ
ĢĬĞĽĹ OĴĻĨİĨŅĦĹçAĤĞİf

    # IĮĬĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĹĹ
    ĤĦĬ ççĨĮĬĻççâĽĦĬĬd ĬĦĞĹĮĨĮĹçĹĞĻĦłflgflflĝd ĤĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìd
                ĠĦĻĞçĝłflgîd ĠĦĻĞçğłflgîîîãf
        ĽĦĬĬgĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬĬgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬĬgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĽĦĬĬgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬĬgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬĬgĠĦĻĞçĝ ł ĠĦĻĞçĝ
        ĽĦĬĬgĠĦĻĞçğ ł ĠĦĻĞçğ

    # CĞĬĬ IJĮĢĦ ĠĦĬIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ łĴĤĞĻĦĽ
    ĤĦĬ ĴĹĦçłĴĤĞĻĦçĴĞĹĞĬĽâĽĦĬĬãf
        ĨĬ ĽĦĬĬgĤĦĢĞÑf
            ĽĦĬĬgĢłĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĽĦĬĬgĬĦĞĹĮĨĮĹçĹĞĻĦ ấ ċ
                âĝg w âĝg κ ĽĦĬĬgĤĦĢĞÑ ấ ĽĦĬĬgĨĻĦĹĞĻĨIJĮĽãã

    # UĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĽ
    ĤĦĬ łĴĤĞĻĦçĴĞĹĞĬĽâĽĦĬĬd ĬĞÑĦĹãf

        # IĬ ĬĞÑĦĹ ĤIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĬĦ ĞĹĹĞÑĽd
        # ĢĹĦĞĻĦ ĻĬĦĬ ĬĨĬĬĦĤ ŃĨĻĬ ŅĦĹIJĽ
        ĨĬ ĮIJĻ ĬĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĬĬĻçĢĞĢĬĦďãf
            ĬĞÑĦĹgŃĦĨĬĬĻçĬIJĬĦĮĻłĬĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨĬĬĻĽã
            ĬĞÑĦĹgŃĦĨĬĬĻçĢĞĢĬĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨĬĬĻĽã
            ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻłĬĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢĬĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

        # UĴĤĞĻĦ ĬIJĬĦĮĻłĬ  ŃĨĻĬ ĢłĹĹĦĮĻ ĬĹĞĤĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨĬĬĻçĬIJĬĦĮĻłĬĽ ł ĽĦĬĬgĠĦĻĞçĝ ấ ċ
                                  ĬĞÑĦĹgŃĦĨĬĬĻçĬIJĬĦĮĻłĬĽ κ ċ
                                  âĝ h ĽĦĬĬgĠĦĻĞçĝã ấ ĬĞÑĦĹgĤŃĦĨĬĬĻĽ
        ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻłĬĽ ł ĽĦĬĬgĠĦĻĞçĝ ấ ċ
                                ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻłĬĽ κ ċ
                                âĝ h ĽĦĬĬgĠĦĻĞçĝã ấ ĬĞÑĦĹgĤĠĨĞĽĦĽ
        # GĦĻ ĢIJĹĹĦĢĻĦĤ ĬIJĬĦĮĻłĬ
        # ĽĦĬĬgĨĻĦĹĞĻĨIJĮ ĨĽ fl ĞĻ ĬĨĹĹĻ ĴĞĽĽ
        # ĞĮĤ ŃĦ ĮĦĦĤ ĻIJ ĽĻĞĹĻ ŃĨĻĬ ĝ ĬĦĹĦ
        ŃĦĨĬĬĻçĬIJĬĦĮĻłĬĽçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨĬĬĻçĬIJĬĦĮĻłĬĽ w ċ
            âĝ h ĽĦĬĬgĠĦĻĞçĝ ấấ âĽĦĬĬgĨĻĦĹĞĻĨIJĮĽ κ ĝãã
        ĠĨĞĽçĬIJĬĦĮĻłĬĽçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻłĬĽ w ċ
            âĝ h ĽĦĬĬgĠĦĻĞçĝ ấấ âĽĦĬĬgĨĻĦĹĞĻĨIJĮĽ κ ĝãã
        # UĴĤĞĻĦ ĢĞĢĬĦ ŃĨĻĬ ĽĶłĞĹĦĤ ĢłĹĹĦĮĻ ĬĹĞĤĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨĬĬĻçĢĞĢĬĦ ł ĽĦĬĬgĠĦĻĞçğ ấ ĬĞÑĦĹgŃĦĨĬĬĻçĢĞĢĬĦ κ ċ
            âĝ h ĽĦĬĬgĠĦĻĞçğã ấ ĬĞÑĦĹgĤŃĦĨĬĬĻĽấấğ
```

```
        ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ ł ĽĦĬÌgĠĦĻĞçğ ǻ ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ κ ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğã ǻ ĬĞÑĦĹgĤĠĨĞĽĦĽǻǻğ
        ſ GĦĻ ĢIJĹĹĦĢĻĦĤ ĢĞĢĨĦ
        ŃĦĨÍÎĻçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢĨĦ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğ ǻǻ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã
        ĠĨĞĽçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ w ċ
            âĝ h ĽĦĬÌgĠĦĻĞçğ ǻǻ âĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κ ĝãã

        ſ VĞĮĨĬĬĞ SGD ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹĬĞĬĨŅĞĻĨIJĮ
        ſ ŃĨĻÎ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĨĦ
        ĬĞÑĦĹgŃĦĨÍÎĻĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ǻ ċ
                        ŃĦĨÍÎĻçĬIJĬĦĮĻŁĬĽçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĽĶĹĻâŃĦĨÍÎĻçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤã κ
                            ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ǻ ċ
                        ĠĨĞĽçĬIJĬĦĮĻŁĬĽçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĽĶĹĻâĠĨĞĽçĢĞĢĨĦçĢIJĹĹĦĢĻĦĤã κ
                            ĽĦĬÌgĦĴĽĨĬIJĮã

    ſ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞĬĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ


ſ CIJĬĬIJĮ ĬIJĽĽ ĢĬĞĽĽ
ĢĬĞĽĽ LIJĽĽf

    ſ RĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽ ĢĞĬĢŁĬĞĻĨIJĮ
    ĤĦÌ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽâĽĦĬÌd ĬĞÑĦĹãf

        ſ ſl ĠÑ ĤĦÌĞŁĬĻ
        ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ ł ſl

        ſ Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ŃĦĨÍÎĻĽ
        ſ ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ ſl
        ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń ſlf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ǻ ċ
                                    ĮĴgĽŁĬâĮĴgĞĢĽâĬĞÑĦĹgŃĦĨÍÎĻĽãã

        ſ Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ŃĦĨÍÎĻĽ
        ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ſlf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ǻ ċ
                                    ĮĴgĽŁĬâĬĞÑĦĹgŃĦĨÍÎĻĽ ǻ ċ
                                            ĬĞÑĦĹgŃĦĨÍÎĻĽã
```

```
        # Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
        # ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ #
        ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń #f
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ĸł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ấ ċ
                                    ĮĴgĽŁĬâĮĴgĞĠĽâĬĞÑĦĹgĠĨĞĽĦĽãã

        # Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
        ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń #f
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ĸł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ấ ċ
                                    ĮĴgĽŁĬâĬĞÑĦĹgĠĨĞĽĦĽ ấ ċ
                                           ĬĞÑĦĹgĠĨĞĽĦĽã

        ĹĦĻŁĹĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ

    # CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽĦĽ
    # ÍĨĹĦĮ ÌIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ĹĞĬŁĦĽ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñãf

        # CĞĬĢŁĬĞĻĦ ĽĞÌĴĬĦ ĬIJĹĽĦĽ
        ĽĞÌĴĬĦçĬIJĹĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        # CĞĬĢŁĬĞĻĦ ÌĦĞĮ ĬIJĹĽ
        ÂĞĻĞçĬIJĹĽ ł ĮĴgÌĦĞĮâĽĞÌĴĬĦçĬIJĹĽĦĽã

        # RĦĻŁĹĮ ĬIJĹĽ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĽ


# CĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ
ĢĬĞĽĽ LIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâLIJĹĽãf

    # FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ÑçĴĹĦÂd ÑçĻĹŁĦãf

        # NŁÌĠĦĹ IJÌ ĽĞÌĴĬĦĽ ĨĮ Ğ ĠĞĻĢÎ
        ĽĞÌĴĬĦĽ ł ĬĦĮâÑçĴĹĦÂã

        # CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦĹĦĮĻ ÂĨĹĨĹĨIJĮ ĠÑ #
        # CĬĨĴ ĠIJĻÎ ĽĨÂĦĽ ĻIJ ĮIJĻ ÂĹĞÍ ÌĦĞĮ ĻIJŃĞĹÂĽ ĞĮÑ ĹĞĬŁĦ
        ÑçĴĹĦÂçĢĬĨĴĴĦÂ ł ĮĴgĢĬĨĴâÑçĴĹĦÂd ĝĦhìd ĝ h ĝĦhìã

        # PĹIJĠĞĠĨĬĨĻĨĦĽ ÌIJĹ ĻĞĹÍĦĻ ĹĞĬŁĦĽ h
        # IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĽ ł ÑçĴĹĦÂçĢĬĨĴĴĦÂä
                ĹĞĮÍĦâĽĞÌĴĬĦĽãd
                ÑçĻĹŁĦ
            ã
```

```
        # MĞĽĨ L·ĞĬŁĦĽ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJÂĦĤ ĬĞĠĦĬĽ
        ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÎĦĮĢĦĽ ł ĮĴgĽŁİâ
                ÑçĴĹĦÎçĢĬĨĴĴĦĤ á ÑçĻĹŁĦd
                ĞÑĨĽłĝ
            ã

        # LIJĹĽĦĽ
        ĮĦÍĞĻĨL·ĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĽ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨÎĦĮĢĦĽã
        ĹĦĻŁĹĮ ĮĦÍĞĻĨL·ĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĽ

    # BĞĢĨŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ĠĞĢĨŃĞĹÂâĽĦĬÌd ÂL·ĞĬŁĦĽd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâÂL·ĞĬŁĦĽã
        # NŁİĠĦĹ IJÌ ĬĞĠĦĬĽ ĨĮ ĦL·ĦĹÑ ĽĞİĴĬĦ
        # WĦďĬĬ ŁĽĦ ĻÎĦ ÌĨĹĽĻ ĽĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻÎĦİ
        ĬĞĠĦĬĽ ł ĬĦĮâÂL·ĞĬŁĦĽäflãã

        # IÌ ĬĞĠĦĬĽ ĞĹĦ ĽĴĞĹĽĦd ĻŁĹĮ ĻÎĦİ ĨĮĻIJ IJĮĦhÎIJĻ L·ĦĢĻIJĹ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ÑçĻĹŁĦ ł ĮĴgĦÑĦâĬĞĠĦĬĽãäÑçĻĹŁĦã

        # CĞĬĢŁĬĞĻĦ ÍĹĞÂĨĦĮĻ
        ĽĦĬÌgÂĨĮĴŁĻĽ ł hÑçĻĹŁĦ w ÂL·ĞĬŁĦĽ
        # NIJĹĬĞĬĨŅĦ ÍĹĞÂĨĦĮĻ
        ĽĦĬÌgÂĨĮĴŁĻĽ ł ĽĦĬÌgÂĨĮĴŁĻĽ w ĽĞİĴĬĦĽ


# SIJÌĻĬĞŃ ĢĬĞĽĽĨÌÌĨĦĹ h ĢIJĬĠĨĮĦÂ SIJÌĻĬĞŃ ĞĢĻĨL·ĞĻĨIJĮ
# ĞĮÂ ĢĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ ÌIJĹ ÌĞĽĻĦĹ ĠĞĢĨŃĞĹÂ ĽĻĦĴ
ĢĬĞĽĽ AĢĻĨL·ĞĻĨIJĮçSIJÌĻĬĞŃçLIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâãf

    # CĹĦĞĻĦĽ ĞĢĻĨL·ĞĻĨIJĮ ĞĮÂ ĬIJĹĽ ÌŁĮĢĻĨIJĮ IJĠÏĦĢĻĽ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ĽĦĬÌgĞĢĻĨL·ĞĻĨIJĮ ł AĢĻĨL·ĞĻĨIJĮçSIJÌĻĬĞŃâã
        ĽĦĬÌgĬIJĹĽ ł LIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâã

    # FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ĨĮĴŁĻĽd ÑçĻĹŁĦãf
        # OŁĻĴŁĻ ĬĞÑĦĹďĽ ĞĢĻĨL·ĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĞĢĻĨL·ĞĻĨIJĮgÌIJĹŃĞĹÂâĨĮĴŁĻĽã
        # SĦĻ ĻÎĦ IJŁĻĴŁĻ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĽĦĬÌgĞĢĻĨL·ĞĻĨIJĮgIJŁĻĴŁĻ
        # CĞĬĢŁĬĞĻĦ ĞĮÂ ĹĦĻŁĹĮ ĬIJĹĽ L·ĞĬŁĦ
        ĹĦĻŁĹĮ ĽĦĬÌgĬIJĹĽgĢĞĬĢŁĬĞĻĦâĽĦĬÌgIJŁĻĴŁĻd ÑçĻĹŁĦã
```

```
    fi BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎHÌ ĠĞĢĪŃĞĹÎâĽHĬÌd ÎĹĞĬŁHĽd ÑçĻĹŁHãf

        fi NŁİĠHĹ IJÌ ĽĞİĴĬHĽ
        ĽĞİĴĬHĽ ł ĬHĮâÎĹĞĬŁHĽã

        fi IÌ ĬĞĠHĬĽ ĞĹH IJĮHhÎIJĻ HĮĢIJÎHHÎd
        fi ĻŁĹĮ ĻÎHİ ĨĮĻIJ ÎÑĽĢĹHĻH ĹĞĬŁHĽ
        ÑÌ ĬHĮâÑçĻĹŁHgĽÎĞĴHã łł ğf
            ÑçĻĹŁH ł ĮĴgĞĹÍİĞÑâÑçĻĹŁHd ĞÑĨĽłĝã

        fi CIJĴÑ ĽIJ ŃH ĢĞĮ ĽĞÌHĬÑ İIJÎÑÌÌÑ
        ĽHĬÌgÎÑĮĴŁĻĽ ł ÎĹĞĬŁHĽgĢIJĴÑâã
        fi CĞĬĢŁĬĞĻH ÍĹĞÎÑHĮĻ
        ĽHĬÌgÎÑĮĴŁĻĽäĹĞĮÍHâĽĞİĴĬHĽãd ÑçĻĹŁHã hł ĝ
        fi NIJĹİĞĬÑĢH ÍĹĞÎÑHĮĻ
        ĽHĬÌgÎÑĮĴŁĻĽ ł ĽHĬÌgÎÑĮĴŁĻĽ w ĽĞİĴĬHĽ


fi BÑĮĞĹÑ ĢĹIJĽĽhHĮĻĹIJĴÑ ĬIJĽĽ
ĢĬĞĽĽ LIJĽĽçBÑĮĞĹÑCĹIJĽĽHĮĻĹIJĴÑâLIJĽĽãf

    fi FIJĹŃĞĹÎ ĴĞĽĽ
    ÎHÌ ÌIJĹŃĞĹÎâĽHĬÌd ÑçĴĹHÎd ÑçĻĹŁHãf

        fi CĬÑĴ ÎĞĻĞ ĻIJ ĴĹHĹHĮĻ ÎÑĹĹÑĬIJĮ ĠÑ fl
        fi CĬÑĴ ĠIJĻÎ ĽÑÎHĽ ĻIJ ĮIJĻ ÎĹĞÍ İHĞĮ ĻIJŃĞĹÎĽ ĞĮÑ ĹĞĬŁH
        ÑçĴĹHÎçĢĬÑĴĴHÎ ł ĮĴgĢĬÑĴâÑçĴĹHÎd ĝHhìd ĝ h ĝHhìã

        fi CĞĬĢŁĬĞĻH ĽĞİĴĬHhŃÑĽH ĬIJĽĽ
        ĽĞİĴĬHçĬIJĽĽHĽ ł hâÑçĻĹŁH ǻ ĮĴgĬIJÍâÑçĴĹHÎçĢĬÑĴĴHÎã κ
                          âĝ h ÑçĻĹŁHã ǻ ĮĴgĬIJÍâĝ h ÑçĴĹHÎçĢĬÑĴĴHÎãã
        ĽĞİĴĬHçĬIJĽĽHĽ ł ĮĴgİHĞĮâĽĞİĴĬHçĬIJĽĽHĽd ĞÑĨĽłhĝã

        fi RHĻŁĹĮ ĬIJĽĽHĽ
        ĹHĻŁĹĮ ĽĞİĴĬHçĬIJĽĽHĽ

    fi BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎHÌ ĠĞĢĪŃĞĹÎâĽHĬÌd ÎĹĞĬŁHĽd ÑçĻĹŁHãf

        fi NŁİĠHĹ IJÌ ĽĞİĴĬHĽ
        ĽĞİĴĬHĽ ł ĬHĮâÎĹĞĬŁHĽã
        fi NŁİĠHĹ IJÌ IJŁĻĴŁĻĽ ÑĮ HĹHĹÑ ĽĞİĴĬH
        fi WHďĬĬ ŁĽH ĻÎH ÌÑĹĽĻ ĽĞİĴĬH ĻIJ ĢIJŁĮĻ ĻÎHİ
        IJŁĻĴŁĻĽ ł ĬHĮâÎĹĞĬŁHĽäflãã

        fi CĬÑĴ ÎĞĻĞ ĻIJ ĴĹHĹHĮĻ ÎÑĹĹÑĬIJĮ ĠÑ fl
        fi CĬÑĴ ĠIJĻÎ ĽÑÎHĽ ĻIJ ĮIJĻ ÎĹĞÍ İHĞĮ ĻIJŃĞĹÎĽ ĞĮÑ ĹĞĬŁH
        ĢĬÑĴĴHÎçÎĹĞĬŁHĽ ł ĮĴgĢĬÑĴâÎĹĞĬŁHĽd ĝHhìd ĝ h ĝHhìã
```

```
        # CǦĬĢŁĬǦĻĦ ÍĹǦĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł hâÑçĻĹŁĦ w ĢĬĨĴĴĦĤçĤĿǦĬŁĦĽ h
                        âĝ h ÑçĻĹŁĦã w âĝ h ĢĬĨĴĴĦĤçĤĿǦĬŁĦĽãã w IJŁĻĴŁĻĽ
        # NIJĹĬǦĬĨĴĦ ÍĹǦĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽǦĬĴĬĦĽ


# MĦǦĮ SĶŁǦĹĦĤ EĹĹIJĹ ĬIJĹĽ
ĢĬǦĽĽ LIJĽĽçMĦǦĮSĶŁǦĹĦĤEĹĹIJĹâLIJĽĽãf  # Lǧ ĬIJĹĽ

    # FIJĹŃǦĹĤ ĴǦĽĽ
    ĤĦÌ ÌIJĹŃǦĹĤâĽĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        # CǦĬĢŁĬǦĻĦ ĬIJĹĽ
        ĽǦĬĴĬĦçĬIJĽĽĦĽ ł ĮĴgĬĦǦĮââÑçĻĹŁĦ h ÑçĴĹĦĤãåǎǧd ǦŃĨĽłhĝã

        # RĦĻŁĹĮ ĬIJĽĽĦĽ
        ĹĦĻŁĹĮ ĽǦĬĴĬĦçĬIJĽĽĦĽ

    # BǦĢĪŃǦĹĤ ĴǦĽĽ
    ĤĦÌ ĠǦĢĪŃǦĹĤâĽĦĬÌd ĤĿǦĬŁĦĽd ÑçĻĹŁĦãf

        # NŁĬĠĦĹ IJÌ ĽǦĬĴĬĦĽ
        ĽǦĬĴĬĦĽ ł ĬĦĮâĤĿǦĬŁĦĽã
        # NŁĬĠĦĹ IJÌ IJŁĻĴŁĻĽ ĨĮ ĦĿĦĹÑ ĽǦĬĴĬĦ
        # WĦďĬĬ ŁĽĦ ĻĤĦ ÌĨĹĽĻ ĽǦĬĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦĬ
        IJŁĻĴŁĻĽ ł ĬĦĮâĤĿǦĬŁĦĽäflãã

        # GĹǦĤĨĦĮĻ IJĮ ĿǦĬŁĦĽ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł hǧ ǻ âÑçĻĹŁĦ h ĤĿǦĬŁĦĽã w IJŁĻĴŁĻĽ
        # NIJĹĬǦĬĨĴĦ ÍĹǦĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽǦĬĴĬĦĽ


# MĦǦĮ AĠĽIJĬŁĻĦ EĹĹIJĹ ĬIJĹĽ
ĢĬǦĽĽ LIJĽĽçMĦǦĮAĠĽIJĬŁĻĦEĹĹIJĹâLIJĽĽãf  # Lĝ ĬIJĹĽ

    ĤĦÌ ÌIJĹŃǦĹĤâĽĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        # CǦĬĢŁĬǦĻĦ ĬIJĹĽ
        ĽǦĬĴĬĦçĬIJĽĽĦĽ ł ĮĴgĬĦǦĮâĮĴgǦĠĽâÑçĻĹŁĦ h ÑçĴĹĦĤãd ǦŃĨĽłhĝã

        # RĦĻŁĹĮ ĬIJĽĽĦĽ
        ĹĦĻŁĹĮ ĽǦĬĴĬĦçĬIJĽĽĦĽ
```

```
    ſi BĞĢĪŃĞĹĤ ĴĞĹĹ
    ÎHÌ ĠĞĢĪŃĞĹĤâĹHĬÌd ÎĹĞĬŁHĹd ÑçĻĹŁHãf

        ſi NŁİĠHĹ IJÌ ĹĞİĴĬHĹ
        ĹĞİĴĬHĹ ł ĬHĮâĤĹĞĬŁHĹã
        ſi NŁİĠHĹ IJÌ IJŁĻĴŁĻĹ ĨĮ HĹHĹÑ ĹĞİĴĬH
        ſi WHďĬĬ ŁĹH ĻĤH ÌĨĹĹĻ ĹĞİĴĬH ĻIJ ĢIJŁĮĻ ĻĤHİ
        IJŁĻĴŁĻĹ ł ĬHĮâĤĹĞĬŁHĹäflãã

        ſi CĞĬĢŁĬĞĻH ÍĹĞĤĨHĮĻ
        ĹHĬÌgĤĨĮĴŁĻĹ ł ĮĴgĹĨÍĮâÑçĻĹŁH h ĤĹĞĬŁHĹã w IJŁĻĴŁĻĹ
        ſi NIJĹİĞĬĨŅH ÍĹĞĤĨHĮĻ
        ĹHĬÌgĤĨĮĴŁĻĹ ł ĹHĬÌgĤĨĮĴŁĻĹ w ĹĞİĴĬHĹ


ſi CĹHĞĻH ĤĞĻĞĹHĻ
Xd Ñ ł ĹĨĮHçĤĞĻĞâã

ſi CĹHĞĻH DHĮĹH ĬĞÑHĹ ŃĨĻĤ ĝ ĨĮĴŁĻ ÌHĞĻŁĹH ĞĮĤ ħǵ IJŁĻĴŁĻ ĹĞĬŁHĹ
ĤHĮĹHĝ ł LĞÑHĹçDHĮĹHâĝd ħǵã

ſi CĹHĞĻH RHLU ĞĢĻĨĹĞĻĨIJĮ âĻIJ ĠH ŁĹHĤ ŃĨĻĤ DHĮĹH ĬĞÑHĹãf
ĞĢĻĨĹĞĻĨIJĮĝ ł AĢĻĨĹĞĻĨIJĮçRHLUâã

ſi CĹHĞĻH ĹHĢIJĮĤ DHĮĹH ĬĞÑHĹ ŃĨĻĤ ħǵ ĨĮĴŁĻ ÌHĞĻŁĹHĹ âĞĹ ŃH ĻĞĬH IJŁĻĴŁĻ
ſi IJÌ ĴĹHĹĨIJŁĹ ĬĞÑHĹ ĤHĹHã ĞĮĤ ħǵ IJŁĻĴŁĻ ĹĞĬŁHĹ
ĤHĮĹHĝ ł LĞÑHĹçDHĮĹHâħǵd ħǵã

ſi CĹHĞĻH RHLU ĞĢĻĨĹĞĻĨIJĮ âĻIJ ĠH ŁĹHĤ ŃĨĻĤ DHĮĹH ĬĞÑHĹãf
ĞĢĻĨĹĞĻĨIJĮğ ł AĢĻĨĹĞĻĨIJĮçRHLUâã

ſi CĹHĞĻH ĻĤĨĹĤ DHĮĹH ĬĞÑHĹ ŃĨĻĤ ħǵ ĨĮĴŁĻ ÌHĞĻŁĹHĹ âĞĹ ŃH ĻĞĬH IJŁĻĴŁĻ
ſi IJÌ ĴĹHĹĨIJŁĹ ĬĞÑHĹ ĤHĹHã ĞĮĤ ĝ IJŁĻĴŁĻ ĹĞĬŁH
ĤHĮĹHġ ł LĞÑHĹçDHĮĹHâħǵd ĝã

ſi CĹHĞĻH LĨĮHĞĹ ĞĢĻĨĹĞĻĨIJĮf
ĞĢĻĨĹĞĻĨIJĮġ ł AĢĻĨĹĞĻĨIJĮçLĨĮHĞĹâã

ſi CĹHĞĻH ĬIJĹĹ ÌŁĮĢĻĨIJĮ
ĬIJĹĹçÌŁĮĢĻĨIJĮ ł LIJĹĹçMHĞĮSĶŁĞĹHĤEĹĹIJĹâã

ſi CĹHĞĻH IJĴĻĨİĨŅHĹ
IJĴĻĨİĨŅHĹ ł OĴĻĨİĨŅHĹçAĤĞİâĬHĞĹĮĨĮĬçĹĞĻHłflgflflĥd ĤHĢĞÑłĝHhġã
```

```
ﬁ AĢĢŁĹĞĢÑ ĴĹĦĢĨĽĨIJĮ ÌIJĹ ĞĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ
ﬁ TÎĦĹĦ ĞĹĦ ĮIJ ĹĦĞĬĬÑ ĞĢĢŁĹĞĢÑ ÌĞĢĻIJĹ ÌIJĹ ĹĦÍĹĦĽĽĨIJĮ ĴĹIJĠĬĦİd
ﬁ ĠŁĻ ŃĦ ĢĞĮ ĽĨİŁĬĞĻĦwĞĴĴĹIJŃĨİĞĻĦ ĨĻg WĦďĬĬ ĢĞĬĢŁĬĞĻĦ ĨĻ ĠÑ ĢÎĦĢĨĨĮÍ
ﬁ ÎIJŃ İĞĮÑ ĻĞĬŁĦĽ ÎĞĻĦ Ğ ÂĨÌÌĦĹĦĮĢĦ ĻIJ ĻÎĦĨĹ ÍĹIJŁĮÂ ĻĹŁĻÎ ĦĶŁĨĻĞĬĦĮĻ
ﬁ ĬĦĽĽ ĻÎĞĮ ÍĨĻĦĮ ĴĹĦĢĨĽĨIJĮ
ﬁ WĦďĬĬ ĢĞĬĢŁĬĞĻĦ ĻÎĨĽ ĴĹĦĢĨĽĨIJĮ ĞĽ Ğ ÌĹĞĢĻĨIJĮ IJÌ ĽĻĞĮÂĞĹÂ ÂĦĻĬĞĻĨIJĮ
ﬁ IJÌ ĞĬĬ ĻÎĦ ÍĹIJŁĮÂ ĻĹŁĻÎ ĻĞĬŁĦĽ
ĞĢĢŁĹĞĢÑçĴĹĦĢĨĽĨIJĮ ł ĮĴgĽĻÂâÑã w ğĥfl

ﬁ TĹĞĨĮ ĨĮ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝflflflĝãf

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ IJÌ IJŁĹ ĻĹĞĨĮĨĮÍ ÂĞĻĞ ĻÎĹIJŁÍÎ ĻÎĨĽ ĬĞÑĦĹ
    ÂĦĮĽĦĝgÌIJĹŃĞĹÂâXã

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĻĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    ﬁ ĻĞĪĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ÌĨĹĽĻ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨĻĞĻĨIJĮĝgÌIJĹŃĞĹÂâÂĦĮĽĦĝgIJŁĻĴŁĻã

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĽĦĢIJĮÂ DĦĮĽĦ ĬĞÑĦĹ
    ﬁ ĻĞĪĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĻĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    ﬁ IJÌ ÌĨĹĽĻ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
    ÂĦĮĽĦğgÌIJĹŃĞĹÂâĞĢĻĨĻĞĻĨIJĮĝgIJŁĻĴŁĻã

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĻĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    ﬁ ĻĞĪĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ĽĦĢIJĮÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨĻĞĻĨIJĮğgÌIJĹŃĞĹÂâÂĦĮĽĦğgIJŁĻĴŁĻã

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĻÎĨĹÂ DĦĮĽĦ ĬĞÑĦĹ
    ﬁ ĻĞĪĦĽ IJŁĻĴŁĻĽ IJÌ ĞĢĻĨĻĞĻĨIJĮ ÌŁĮĢĻĨIJĮ IJÌ ĽĦĢIJĮÂ ĬĞÑĦĹ ĞĽ ĨĮĴŁĻĽ
    ÂĦĮĽĦġgÌIJĹŃĞĹÂâĞĢĻĨĻĞĻĨIJĮğgIJŁĻĴŁĻã

    ﬁ PĦĹÌIJĹİ Ğ ÌIJĹŃĞĹÂ ĴĞĽĽ ĻÎĹIJŁÍÎ ĞĢĻĨĻĞĻĨIJĮ ÌŁĮĢĻĨIJĮ
    ﬁ ĻĞĪĦĽ ĻÎĦ IJŁĻĴŁĻ IJÌ ĻÎĨĹÂ ÂĦĮĽĦ ĬĞÑĦĹ ÎĦĹĦ
    ĞĢĻĨĻĞĻĨIJĮġgÌIJĹŃĞĹÂâÂĦĮĽĦġgIJŁĻĴŁĻã

    ﬁ CĞĬĢŁĬĞĻĦ ĻÎĦ ÂĞĻĞ ĬIJĽĽ
    ÂĞĻĞçĬIJĽĽ ł ĬIJĽĽçÌŁĮĢĻĨIJĮgĢĞĬĢŁĬĞĻĦâĞĢĻĨĻĞĻĨIJĮġgIJŁĻĴŁĻd Ñã

    ﬁ CĞĬĢŁĬĞĻĦ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮ ĴĦĮĞĬĻÑ
    ĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ ł ċ
        ĬIJĽĽçÌŁĮĢĻĨIJĮgĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽâÂĦĮĽĦĝã κ ċ
        ĬIJĽĽçÌŁĮĢĻĨIJĮgĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽâÂĦĮĽĦğã κ ċ
        ĬIJĽĽçÌŁĮĢĻĨIJĮgĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽâÂĦĮĽĦġã

    ﬁ CĞĬĢŁĬĞĻĦ IJĻĦĹĞĬĬ ĬIJĽĽ
    ĬIJĽĽ ł ÂĞĻĞçĬIJĽĽ κ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ
```

```
ﬁ CĞĬĢŁĬĞĻĦ ĞĢĢŁĹĞĢÑ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĞĢĻĨĿĞĻĨIJĮğ ĞĮÊ ĻĞĹÍĦĻĹ
ﬁ TIJ ĢĞĬĢŁĬĞĻĦ ĨĻ ŃĦďĹĦ ĻĞĬĨĨĮÍ ĞĠĹIJĬŁĻĦ ÊĨÌÌĦĹĦĮĢĦ ĠĦĻŃĦĦĮ
ﬁ ĴĹĦÊĨĢĻĨIJĮĹ ĞĮÊ ÍĹIJŁĮÊ ĻĹŁĻÎ ĿĞĬŁĦĹ ĞĮÊ ĢIJİĴĞĹĦ ĨÌ ÊĨÌÌĦĹĦĮĢĦĹ
ﬁ ĞĹĦ ĬIJŃĦĹ ĻÎĞĮ ÍĨĿĦĮ ĴĹĦĢĨĹĨIJĮ ĿĞĬŁĦ
ĴĹĦÊĨĢĻĨIJĮĹ ł ĞĢĻĨĿĞĻĨIJĮġgIJŁĻĴŁĻ
ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĮĴgĞĠĹIJĬŁĻĦâĴĹĦÊĨĢĻĨIJĮĹ h Ñã l·
                    ĞĢĢŁĹĞĢÑçĴĹĦĢĨĹĨIJĮã

ĨÌ ĮIJĻ ĦĴIJĢÎ ķ ĝﬂﬂf
    ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
          ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
          ÌďĬIJĹĹf ăĬIJĹĹfgġÌå âď κ
          ÌďÊĞĻĞçĬIJĹĹf ăÊĞĻĞçĬIJĹĹfgġÌåd ď κ
          ÌďĹĦÍçĬIJĹĹf ăĹĦÍŁĬĞĹĨŃĞĻĨIJĮçĬIJĹĹfgġÌåãd ď κ
          ÌďĬĹf ăIJĴĻĨÌĨŃĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

ﬁ BĞĢĨŃĞĹÊ ĴĞĹĹ
ĬIJĹĹçÌŁĮĢĻĨIJĮġĞĢĢĨŃĞĹÊâĞĢĻĨĿĞĻĨIJĮġgIJŁĻĴŁĻd Ñã
ĞĢĻĨĿĞĻĨIJĮġgĞĢĢĨŃĞĹÊâĬIJĹĹçÌŁĮĢĻĨIJĮgÊĨĮĴŁĻĹã
ÊĦĮĹĦġgĞĢĢĨŃĞĹÊâĞĢĻĨĿĞĻĨIJĮġgÊĨĮĴŁĻĹã
ĞĢĻĨĿĞĻĨIJĮğgĞĢĢĨŃĞĹÊâÊĦĮĹĦġgÊĨĮĴŁĻĹã
ÊĦĮĹĦğgĞĢĢĨŃĞĹÊâĞĢĻĨĿĞĻĨIJĮğgÊĨĮĴŁĻĹã
ĞĢĻĨĿĞĻĨIJĮĝgĞĢĢĨŃĞĹÊâÊĦĮĹĦğgÊĨĮĴŁĻĹã
ÊĦĮĹĦĝgĞĢĢĨŃĞĹÊâĞĢĻĨĿĞĻĨIJĮĝgÊĨĮĴŁĻĹã

ﬁ UĴÊĞĻĦ ŃĦĨÍÎĻĹ ĞĮÊ ĠĨĞĹĦĹ
IJĴĻĨÌĨŃĦĹgĴĹĦçŁĴÊĞĻĦçĴĞĹĞİĹâã
IJĴĻĨÌĨŃĦĹgŁĴÊĞĻĦçĴĞĹĞİĹâÊĦĮĹĦĝã
IJĴĻĨÌĨŃĦĹgŁĴÊĞĻĦçĴĞĹĞİĹâÊĦĮĹĦğã
IJĴĻĨÌĨŃĦĹgŁĴÊĞĻĦçĴĞĹĞİĹâÊĦĮĹĦġã
IJĴĻĨÌĨŃĦĹgĴIJĹĻçŁĴÊĞĻĦçĴĞĹĞİĹâã


ĨÌĴIJĹĻ İĞĻĴĬIJĻĬĨĞgĴÑĴĬIJĻ ĞĹ ĴĬĻ

XçĻĦĹĻd ÑçĻĦĹĻ ł ĹĨĮĦçÊĞĻĞâã

ÊĦĮĹĦĝgÌIJĹŃĞĹÊâXçĻĦĹĻã
ĞĢĻĨĿĞĻĨIJĮĝgÌIJĹŃĞĹÊâÊĦĮĹĦĝgIJŁĻĴŁĻã
ÊĦĮĹĦğgÌIJĹŃĞĹÊâĞĢĻĨĿĞĻĨIJĮĝgIJŁĻĴŁĻã
ĞĢĻĨĿĞĻĨIJĮğgÌIJĹŃĞĹÊâÊĦĮĹĦğgIJŁĻĴŁĻã
ÊĦĮĹĦġgÌIJĹŃĞĹÊâĞĢĻĨĿĞĻĨIJĮğgIJŁĻĴŁĻã
ĞĢĻĨĿĞĻĨIJĮġgÌIJĹŃĞĹÊâÊĦĮĹĦġgIJŁĻĴŁĻã

ĴĬĻgĴĬIJĻâXçĻĦĹĻd ÑçĻĦĹĻã
ĴĬĻgĴĬIJĻâXçĻĦĹĻd ĞĢĻĨĿĞĻĨIJĮġgIJŁĻĴŁĻã
ĴĬĻgĹÎIJŃâã
```

Supplementary Material: https://nnfs.io/ch17

Chapter code, further resources, and errata for this chapter

Chapter 18

Model Object

We built a model that can perform the forward pass, backward pass, and ancillary tasks like measuring accuracy. We have built all this by writing a fair bit of code and making modifications in some decently-sized blocks of code. It's beginning to make more sense to make our model an object itself, especially since we will want to do things like save and load this object to use for future prediction tasks. We will also use this object to cut down on some of the more common lines of code, making it easier to work with our current code base and build new models. To do this model object conversion, we'll use the last model we were working on, the regression model with sine data:

```
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ÎĬĴIJĹĻ ĽĨĮĦçĤĞĻĞ
Xd Ñ ł ĽĨĮĦçĤĞĻĞâã
```

Once we have the data, our first step for the model class is to add in the various layers we want. Thus, we can begin our model class by doing:

```
ƒ MIJÂHĬ ĢĬĞĹĹ
ĢĬĞĹĹ MIJÂHĬf

    ÂHÌ ççĨĮĨĻççâĹHĬÌãf
        ƒ CĹHĞĻH Ğ ĬĨĹĻ IJÌ ĮHĻŃIJĹĨ IJĠÏHĢĻĹ
        ĹHĬÌgĬĞÑHĹĹ ł äã

    ƒ AÂÂ IJĠÏHĢĻĹ ĻIJ ĻÎH ÌIJÂHĬ
    ÂHÌ ĞÂÂâĹHĬÌd ĬĞÑHĹãf
        ĹHĬÌgĬĞÑHĹĹgĞĴĴHĮÂâĬĞÑHĹã
```

This allows us to use the ĞÂÂ method of the model object to add layers. This alone will help with legibility considerably. Let's add some layers:

```
ƒ IĮĹĻĞĮĻĨĞĻH ĻÎH ÌIJÂHĬ
ÌIJÂHĬ ł MIJÂHĬâã

ƒ AÂÂ ĬĞÑHĹĹ
ÌIJÂHĬgĞÂÂâLĞÑHĹçDHĮĹHâĝd ħǵãã
ÌIJÂHĬgĞÂÂâAĢĻĨĻĞĻĨIJĮçRHLUâãã
ÌIJÂHĬgĞÂÂâLĞÑHĹçDHĮĹHâħǵd ħǵãã
ÌIJÂHĬgĞÂÂâAĢĻĨĻĞĻĨIJĮçRHLUâãã
ÌIJÂHĬgĞÂÂâLĞÑHĹçDHĮĹHâħǵd ĝãã
ÌIJÂHĬgĞÂÂâAĢĻĨĻĞĻĨIJĮçLĨĮHĞĹâãã
```

We can also query this model now:

```
ĴĹĨĮĻâÌIJÂHĬgĬĞÑHĹĹã


ńńń
äl·ççÌĞĨĮççgLĞÑHĹçDHĮĹH IJĠÏHĢĻ ĞĻ flŇflflflflflĝĥEîDĥflǵBCíńd
l·ççÌĞĨĮççgAĢĻĨĻĞĻĨIJĮçRHLU IJĠÏHĢĻ ĞĻ flŇflflflflflĝĥEîDĥflǵCǵíńd
l·ççÌĞĨĮççgLĞÑHĹçDHĮĹH IJĠÏHĢĻ ĞĻ flŇflflflflflĝĥEîDĥflǵCííńd
l·ççÌĞĨĮççgAĢĻĨĻĞĻĨIJĮçRHLU IJĠÏHĢĻ ĞĻ flŇflflflflflĝĥEîDĥflǵCCíńd
l·ççÌĞĨĮççgLĞÑHĹçDHĮĹH IJĠÏHĢĻ ĞĻ flŇflflflflflĝĥEîDĥflǵDflíńd
l·ççÌĞĨĮççgAĢĻĨĻĞĻĨIJĮçLĨĮHĞĹ IJĠÏHĢĻ ĞĻ flŇflflflflflĝĥEîDĥflǵDííńã
```

Besides adding layers, we also want to set a loss function and optimizer for the model. To do this, we'll create a method called ĹHĻ:

```
    ƒ SHĻ ĬIJĹĹ ĞĮÂ IJĴĻĨÌĨĨĨĮHĹ
     ÂHÌ ĹHĻâĹHĬÌd ǻd ĬIJĹĹd IJĴĻĨÌĨĨĨĮHĹãf
         ĹHĬÌgĬIJĹĹ ł ĬIJĹĹ
         ĹHĬÌgIJĴĻĨÌĨĨĨĮHĹ ł IJĴĻĨÌĨĨĨĮHĹ
```

The use of the asterisk in the parameter definitions notes that the subsequent parameters (*ĬIJĽĽ* and *IJĴĻĨĬĨŅĦĹ* in this case) are keyword arguments. Since they have no default value assigned, they are required keyword arguments, which means that they have to be passed by names and values, making code more legible.

Now we can add a call to this method into our newly-created model object, and pass the loss and optimizer objects:

```
ſi CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĨĮĦçĤĞĻĞâã

ſi IĮĽĻĞĮĻĨĞĻĦ ĻÎĦ ĬIJĤĦĬ
ĬIJĤĦĬ ł MIJĤĦĬâã

ſi AĤĤ ĬĞÑĦĹĽ
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝd ħǵãã
ĬIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçRĦLUâãã
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâħǵd ħǵãã
ĬIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçRĦLUâãã
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâħǵd ĝãã
ĬIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçLĨĮĦĞĹâãã

ſi SĦĻ ĬIJĽĽ ĞĮĤ IJĴĻĨĬĨŅĦĹ IJĠĬĦĢĻĽ
ĬIJĤĦĬgĽĦĻâ
    ĬIJĽĽłLIJĽĽçMĦĞĮSĶŁĞĹĦĤEĹĹIJĹâãd
    IJĴĻĨĬĨŅĦĹłOĴĻĨĬĨŅĦĹçAĤĞĬâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflĥd ĤĦĢĞÑłĝĦhġãd
ã
```

After we've set our model's layers, loss function, and optimizer, the next step is to train, so we'll add a train method. For now, we'll make it a placeholder and fill it in soon:

```
    ſi TĹĞĨĮ ĻÎĦ ĬIJĤĦĬ
    ĤĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ǻd ĦĴIJĢÎĽłĝd ĴĹĨĮçĦŁĦĹÑłĝãf

        ſi MĞĨĮ ĻĹĞĨĮĨĮĬ ĬIJIJĴ
        ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮĬĦâĝd ĦĴIJĢÎĽĸĝãf

            ſi TĦĬĴIJĹĞĹÑ
            ĴĞĽĽ
```

We can then add a call to the train method in the model definition. We'll pass the training data, the number of epochs (10000, as we've used so far), and an indicator of how often to print a training summary. We do not need or want to print it every step, so we'll make it configurable:

```
# CĹĦĞĻĦ ÂĞĻĞĹĦĻ
Xd Ñ ł ĹĨĨĦçÂĞĻĞâã

# IĮĹĻĞĮĻĨĞĻĦ ĻÎĦ ĬIJÂĦĬ
ĬIJÂĦĬ ł MIJÂĦĬâã

# AÂÂ ĬĞÑĦĹĹ
ĬIJÂĦĬgĞÂÂâLĞÑĦĹçDĦĮĹĦâĝd ħǵãã
ĬIJÂĦĬgĞÂÂâAĢĻĨĹĞĻĨIJĮçRĦLUâãã
ĬIJÂĦĬgĞÂÂâLĞÑĦĹçDĦĮĹĦâħǵd ħǵãã
ĬIJÂĦĬgĞÂÂâAĢĻĨĹĞĻĨIJĮçRĦLUâãã
ĬIJÂĦĬgĞÂÂâLĞÑĦĹçDĦĮĹĦâħǵd ĝãã
ĬIJÂĦĬgĞÂÂâAĢĻĨĹĞĻĨIJĮçLĨĮĦĞĹâãã

# SĦĻ ĬIJĹĹ ĞĮÂ IJĴĻĨĬĨŅĦĹ IJĠĬĦĢĻĹ
ĬIJÂĦĬgĹĦĻâ
    ĬIJĹĹłLIJĹĹçMĦĞĮSĶŁĞĹĦÂEĹĹIJĹâãd
    IJĴĻĨĬĨŅĦĹłOĴĻĨĬĨŅĦĹçAÂĞĬâĬĦĞĹĮĨĮĺçĹĞĻĦłflgflflĥd ÂĦĢĞÑłĝĦhġãd
ã

ĬIJÂĦĬgĻĹĞĨĮâXd Ñd ĦĴIJĢÎĹłĝflflflfld ĴĹĨĮĻçĦĹĦĹÑłĝflflã
```

To actually train, we need to perform a forward pass. Performing this forward pass in the object is slightly more complicated because we want to do this in a loop over the layers, and we need to know the previous layer's output to pass data properly. One issue with querying the previous layer is that the first layer doesn't have a "previous" layer. The first layer that we're defining is the first **hidden** layer. One option we have then is to create an "input layer." This is considered a layer in a neural network but doesn't have weights and biases associated with it. The input layer only contains the training data, and we'll only use it as a "previous" layer to the first layer during the iteration of the layers in a loop. We'll create a new class and call it in similarly as `LĞÑĦĹçDĦĮĹĦ` class — `LĞÑĦĹçIĮĴłĻ`:

```
# IĮĴłĻ ðĬĞÑĦĹð
ĢĬĞĹĹ LĞÑĦĹçIĮĴłĻf

    # FIJĹŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ÌIJĹŃĞĹÂâĹĦĬÌd ĨĮĴłĻĹãf
        ĹĦĬÌgIJłĻĴłĻ ł ĨĮĴłĻĹ
```

The ÌIJĹŃĞĹÎ method sets training samples as ĽĦĬÌgIJŁĻĴŁĻ. This property is common with other layers. There's no need for a backward method here since we'll never use it. It might seem silly right now to even have this class, but it should hopefully become clear how we're going to use this shortly. The next thing we're going to do is set the previous and next layer properties for each of the model's layers. We'll create a method called ÌĨĮĞĬĨŅĦ in the MIJÎĦĬ class:

```
ƒ FĨĮĞĬĨŅĦ ĻÎĦ ÌIJÎĦĬ
ÎĦÌ ÌĨĮĞĬĨŅĦâĽĦĬÌãf

    ƒ CĹĦĞĻĦ ĞĮÎ ĽĦĻ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
    ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ ł LĞÑĦĹçIĮĴŁĻâã

    ƒ CIJŁĮĻ ĞĬĬ ĻÎĦ IJĠÏĦĢĻĽ
    ĬĞÑĦĹçĢIJŁĮĻ ł ĬĦĮâĽĦĬÌgĬĞÑĦĹĽã

    ƒ IĻĦĹĞĻĦ ĻÎĦ IJĠÏĦĢĻĽ
    ÌIJĹ Ĩ ĨĮ ĹĞĮÍĦâĬĞÑĦĹçĢIJŁĮĻãf

        ƒ IÌ ĨĻďĽ ĻÎĦ ÌĨĹĽĻ ĬĞÑĦĹd
        ƒ ĻÎĦ ĴĹĦ·ĨIJŁĽ ĬĞÑĦĹ IJĠÏĦĢĻ ĨĽ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
        ĨÌ Ĩ łł ƒf
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĴĹĦ· ł ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĮĦŇĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝã

        ƒ AĬĬ ĬĞÑĦĹĽ ĦŇĢĦĴĻ ÌIJĹ ĻÎĦ ÌĨĹĽĻ ĞĮÎ ĻÎĦ ĬĞĽĻ
        ĦĬĨÌ Ĩ l· ĬĞÑĦĹçĢIJŁĮĻ h ĝf
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĴĹĦ· ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝã
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĮĦŇĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝã

        ƒ TÎĦ ĬĞĽĻ ĬĞÑĦĹ h ĻÎĦ ĮĦŇĻ IJĠÏĦĢĻ ĨĽ ĻÎĦ ĬIJĹĽ
        ĦĬĽĦf
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĴĹĦ· ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝã
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĮĦŇĻ ł ĽĦĬÌgĬIJĹĽ
```

This code creates an input layer and sets ĮĦŇĻ and ĴĹĦ· references for each layer contained within the ĽĦĬÌgĬĞÑĦĹĽ list of a model object. We wanted to create the LĞÑĦĹçIĮĴŁĻ class to set the ĴĹĦ· property of the first hidden layer in a loop since we are going to call all of the layers in a uniform way. The ĮĦŇĻ layer for the final layer will be the loss, which we already have created.

Now that we have the necessary layer information for our model object to perform a forward pass, let's add a forward method. We will use this forward method both for when we train and later when we just want to predict, which is also called **model inference**. Continuing the code within the MIJÎĦĬ class:

```
# FIJĹŃĞĹÂ ĴĞĽĽ
ĢĬĞĽĽ MIJÂĦĬf
    ggg
    # PĦĹÌIJĹİĽ ÌIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd Xãf

        # CĞĬĬ ÌIJĹŃĞĹÂ İĦĻÎIJÂ IJĮ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
        # ĻÎĨĽ ŃĨĬĬ ĽĦĻ ĻÎĦ IJŁĻĴŁĻ ĴĹIJĴĦĹĻÑ ĻÎĞĻ
        # ĻÎĦ ÌĨĹĽĻ ĬĞÑĦĹ ĨĮ ðĴĹĦĻð IJĠÏĦĢĻ ĨĽ ĦŇĴĦĢĻĨĮÍ
        ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹgÌIJĹŃĞĹÂâXã

        # CĞĬĬ ÌIJĹŃĞĹÂ İĦĻÎIJÂ IJÌ ĦĻĦĹÑ IJĠÏĦĢĻ ĨĮ Ğ ĢÎĞĨĮ
        # PĞĽĽ IJŁĻĴŁĻ IJÌ ĻÎĦ ĴĹĦĻĨIJŁĽ IJĠÏĦĢĻ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
        ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĬĞÑĦĹĽf
            ĬĞÑĦĹgÌIJĹŃĞĹÂâĬĞÑĦĹgĴĹĦĻgIJŁĻĴŁĻã

        # ðĬĞÑĦĹð ĨĽ ĮIJŃ ĻÎĦ ĬĞĽĻ IJĠÏĦĢĻ ÌĹIJİ ĻÎĦ ĬĨĽĻd
        # ĹĦĻŁĹĮ ĨĻĽ IJŁĻĴŁĻ
        ĹĦĻŁĹĮ ĬĞÑĦĹgIJŁĻĴŁĻ
```

In this case, we take in *X* (input data), then simply pass this data through the `ĨĮĴŁĻçĬĞÑĦĹ` in the `MIJÂĦĬ` object, which creates an `IJŁĻĴŁĻ` attribute in this object. From here, we begin iterating over the `ĽĦĬÌgĬĞÑĦĹĽ`, the layers starting with the first hidden layer. We perform a forward pass on the `ĬĞÑĦĹgĴĹĦĻgIJŁĻĴŁĻ`, the output data of the previous layer, for each layer. For the first hidden layer, the `ĬĞÑĦĹgĴĹĦĻ` is `ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ`. The `IJŁĻĴŁĻ` attribute is created for each layer when we call the `ÌIJĹŃĞĹÂ` method, which is then used as input to a `ÌIJĹŃĞĹÂ` method call on the next layer. Once we've iterated over all of the layers, we return the final layer's output. That's a forward pass, and now let's go ahead and add this forward pass method call to the `ĻĹĞĨĮ` method in the `MIJÂĦĬ` class:

```
# FIJĹŃĞĹÂ ĴĞĽĽ
ĢĬĞĽĽ MIJÂĦĬf
    ggg
    # TĹĞĨĮ ĻÎĦ İIJÂĦĬ
    ÂĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ǻd ĦĴIJĢÎĽłĝd ĴĹĨĮçĦĹĦĹÑłĝãf

        # MĞĨĮ ĻĹĞĨĮĨĮÍ ĬIJIJĴ
        ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝd ĦĴIJĢÎĽĸĝãf

            # PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÂ ĴĞĽĽ
            IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÂâXã

            # TĦİĴIJĹĞĹÑ
            ĴĹĨĮĻâIJŁĻĴŁĻã
            ĦŇĨĻâã
```

Full MIJÂHĬ class up to this point:

```
ƒ MIJÂHĬ ĢĬĞĹĹ
ĢĬĞĹĹ MIJÂHĬf

    ÂHÌ ççĨĮĨĻççâĹĤĬÌãf
        ƒ CĹĤĞĻĤ Ğ ĬĨĹĻ IJÌ ĮĤĻŃIJĹĨ IJĠÏĤĢĻĹ
        ĹĤĬÌgĬĞÑĤĹĹ ł äã

    ƒ AÂÂ IJĠÏĤĢĻĹ ĻIJ ĻÎĤ ÌIJÂHĬ
    ÂHÌ ĞÂÂâĹĤĬÌd ĬĞÑĤĹãf
        ĹĤĬÌgĬĞÑĤĹĹgĞĴĴĤĮÂâĬĞÑĤĹã

    ƒ SĤĻ ĬIJĹĹ ĞĮÂ IJĴĻĨÌĨĨÑĤĹ
    ÂHÌ ĹĤĻâĹĤĬÌd ǻd ĬIJĹĹd IJĴĻĨÌĨĨÑĤĹãf
        ĹĤĬÌgĬIJĹĹ ł ĬIJĹĹ
        ĹĤĬÌgIJĴĻĨÌĨĨÑĤĹ ł IJĴĻĨÌĨĨÑĤĹ

    ƒ FĨĮĞĬĨÑĤ ĻÎĤ ÌIJÂHĬ
    ÂHÌ ÌĨĮĞĬĨÑĤâĹĤĬÌãf

        ƒ CĹĤĞĻĤ ĞĮÂ ĹĤĻ ĻÎĤ ĨĮĴŁĻ ĬĞÑĤĹ
        ĹĤĬÌgĨĮĴŁĻçĬĞÑĤĹ ł LĞÑĤĹçIĮĴŁĻâã

        ƒ CIJŁĮĻ ĞĬĬ ĻÎĤ IJĠÏĤĢĻĹ
        ĬĞÑĤĹçĢIJŁĮĻ ł ĬĤĮâĹĤĬÌgĬĞÑĤĹĹã

        ƒ IĻĤĹĞĻĤ ĻÎĤ IJĠÏĤĢĻĹ
        ÌIJĹ Ĩ ĨĮ ĹĞĮÍĤâĬĞÑĤĹçĢIJŁĮĻãf

            ƒ IÌ ĨĻďĹ ĻÎĤ ÌĨĹĹĻ ĬĞÑĤĹd
            ƒ ĻÎĤ ĴĹĤL·ĨIJŁĹ ĬĞÑĤĹ IJĠÏĤĢĻ ĨĹ ĻÎĤ ĨĮĴŁĻ ĬĞÑĤĹ
            ĨÌ Ĩ łł flf
                ĹĤĬÌgĬĞÑĤĹĹäĨāgĴĹĤL· ł ĹĤĬÌgĨĮĴŁĻçĬĞÑĤĹ
                ĹĤĬÌgĬĞÑĤĹĹäĨāgĮĤŃĻ ł ĹĤĬÌgĬĞÑĤĹĹäĨĸĝā

            ƒ AĬĬ ĬĞÑĤĹĹ ĤŃĢĤĴĻ ÌIJĹ ĻÎĤ ÌĨĹĹĻ ĞĮÂ ĻÎĤ ĬĞĹĻ
            ĤĬĨÌ Ĩ l· ĬĞÑĤĹçĢIJŁĮĻ h ĝf
                ĹĤĬÌgĬĞÑĤĹĹäĨāgĴĹĤL· ł ĹĤĬÌgĬĞÑĤĹĹäĨhĝā
                ĹĤĬÌgĬĞÑĤĹĹäĨāgĮĤŃĻ ł ĹĤĬÌgĬĞÑĤĹĹäĨĸĝā

            ƒ TÎĤ ĬĞĹĻ ĬĞÑĤĹ h ĻÎĤ ĮĤŃĻ IJĠÏĤĢĻ ĨĹ ĻÎĤ ĬIJĹĹ
            ĤĬĹĤf
                ĹĤĬÌgĬĞÑĤĹĹäĨāgĴĹĤL· ł ĹĤĬÌgĬĞÑĤĹĹäĨhĝā
                ĹĤĬÌgĬĞÑĤĹĹäĨāgĮĤŃĻ ł ĹĤĬÌgĬIJĹĹ
```

```
ƒ TĹĞĨĮ ĻÎĦ İIJĤĦĬ
ÂĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ǻd ĦĴIJĢÎĽłĝd ĴĹĨĮĻçĦŁĦĹÑłĝãf

    ƒ MĞĨĮ ĻĹĞĨĮĨĮÍ ĬIJIJĴ
    ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝd ĦĴIJĢÎĽĸĝãf

        ƒ PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÂ ĴĞĽĽ
        IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÂâXã

        ƒ TĦİĴIJĹĞĹÑ
        ĴĹĨĮĻâIJŁĻĴŁĻã
        ĦŇĨĻâã

ƒ PĦĹÌIJĹİĽ ÌIJĹŃĞĹÂ ĴĞĽĽ
ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd Xãf

    ƒ CĞĬĬ ÌIJĹŃĞĹÂ İĦĻÎIJÂ IJĮ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
    ƒ ĻÎĨĽ ŃĨĬĬ ĽĦĻ ĻÎĦ IJŁĻĴŁĻ ĴĹIJĴĦĹĻÑ ĻÎĞĻ
    ƒ ĻÎĦ ÌĨĹĽĻ ĬĞÑĦĹ ĨĮ ðĴĹĦŁð IJĠÏĦĢĻ ĨĽ ĦŇĴĦĢĻĨĮÍ
    ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹgÌIJĹŃĞĹÂâXã

    ƒ CĞĬĬ ÌIJĹŃĞĹÂ İĦĻÎIJÂ IJÌ ĦŁĦĹÑ IJĠÏĦĢĻ ĨĮ Ğ ĢÎĞĨĮ
    ƒ PĞĽĽ IJŁĻĴŁĻ IJÌ ĻÎĦ ĴĹĦŁĨIJŁĽ IJĠÏĦĢĻ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
    ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĬĞÑĦĹĽƒ
        ĬĞÑĦĹgÌIJĹŃĞĹÂâĬĞÑĦĹgĴĹĦŁglJŁĻĴŁĻã

    ƒ ðĬĞÑĦĹð ĨĽ ĮIJŃ ĻÎĦ ĬĞĽĻ IJĠÏĦĢĻ ÌĹIJİ ĻÎĦ ĬĨĽĻd
    ƒ ĹĦĻŁĹĮ ĨĻĽ IJŁĻĴŁĻ
    ĹĦĻŁĹĮ ĬĞÑĦĹglJŁĻĴŁĻ
```

Finally, we can add in the `ÌĨĮĞĬĨĢĦ` method call to the main code (recall this method makes, among other things, the model's layers aware of their previous and next layers).

```
ƒ CĹĦĞĻĦ ÂĞĻĞĽĦĻ
Xd Ñ ł ĽĨĮĦçÂĞĻĞâã

ƒ IĮĽĻĞĮĻĨĞĻĦ ĻÎĦ İIJĤĦĬ
İIJĤĦĬ ł MIJĤĦĬâã

ƒ AÂÂ ĬĞÑĦĹĽ
İIJĤĦĬgĞÂÂâLĞÑĦĹçDĦĮĽĦâĝd ħǵãã
İIJĤĦĬgĞÂÂâAĢĻĨĽĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞÂÂâLĞÑĦĹçDĦĮĽĦâħǵd ħǵãã
İIJĤĦĬgĞÂÂâAĢĻĨĽĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞÂÂâLĞÑĦĹçDĦĮĽĦâħǵd ĝãã
İIJĤĦĬgĞÂÂâAĢĻĨĽĞĻĨIJĮçLĨĮĦĞĹâãã
```

```
ń SĦĻ ĬIJĽĽ ĞĮĤ IJĴĻĨĬĨŅĦĹ IJĠĬĦĢĻĽ
ĬIJĤĦĬgĽĦĻâ
    ĬIJĽĽłLIJĽĽçMĦĞĮSĶŁĞĹĦĤEĹĹIJĹâãd
    IJĴĻĨĬĨŅĦĹłOĴĻĨĬĨŅĦĹçAĤĞĬâĬĦĞĹĮĨĮĬçĹĞĻĦłflgflflĥd ĤĦĢĞÑłĝĦhġãd
ã

ń FĨĮĞĬĨŅĦ ĻĨĦ ĬIJĤĦĬ
ĬIJĤĦĬgĬĨĮĞĬĨŅĦâã

ń TĹĞĨĮ ĻĨĦ ĬIJĤĦĬ
ĬIJĤĦĬgĻĹĞĨĮâXd Ñd ĦĴIJĢĬĽłĝflflflfld ĴĹĨĮĻçĦĽĦĹÑłĝflflã
```

Running this:

```
ńńń
ää flgflflflflflflflflĦĸflflā
 ähĝggġgğflîĝgîĦhflíā
 ähğggĥgĝíğîìĦhflíā
ggg
 ähĝggğíĥîĥĝgĦhflĥā
 ähĝggğîíğìğĥĦhflĥā
 ähĝggġflîĥîġflĦhflĥāā
```

At this point, we've covered the forward pass of our model in the `MIJĤĦĬ` class. We still need to calculate loss and accuracy along with doing backpropagation. Before doing this, we need to know which layers are "trainable," which means layers with weights and biases that we can tweak. To do this, we need to check if the layer has either a `ŃĦĨÍÎĻĽ` or `ĠĨĞĽĦĽ` attribute. We can check this with the following code:

```
ń IÌ ĬĞÑĦĹ ĢIJĮĻĞĨĮĽ ĞĮ ĞĻĻĹĨĠŁĻĦ ĢĞĬĬĦĤ ðŃĦĨÍÎĻĽdð
ń ĨĻdĽ Ğ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹ h
ń ĞĤĤ ĨĻ ĻIJ ĻĨĦ ĬĨĽĻ IJÌ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
ń WĦ ĤIJĮdĻ ĮĦĦĤ ĻIJ ĢĨĦĢĨ ÌIJĹ ĠĨĞĽĦĽ h
ń ĢĨĦĢĨĨĮÍ ÌIJĹ ŃĦĨÍÎĻĽ ĨĽ ĦĮIJŁÍÎ
ĨÌ ÎĞĽĞĻĻĹâĽĦĬÌgĬĞÑĦĹĽäĨād dŃĦĨÍÎĻĽdãf
    ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽgĞĴĴĦĮĤâĽĦĬÌgĬĞÑĦĹĽäĨāã
```

Where Ĩ is the index for the layer in the list of layers. We'll put this code into the ÌĨĮĞĬĨŅĦ method. The full code for that method so far:

```
ſı FĨĮĞĬĨŅĦ ĻÎĦ İIJĤĦĬ
ÎĦÌ ÌĨĮĞĬĨŅĦâĽĦĬÌãf

    ſı CĹĦĞĻĦ ĞĮĤ ĽĦĻ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
    ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ ł LĞÑĦĹçIĮĴŁĻâã

    ſı CIJŁĮĻ ĞĬĬ ĻÎĦ IJĠÏĦĢĻĽ
    ĬĞÑĦĹçĢIJŁĮĻ ł ĬĦĮâĽĦĬÌgĬĞÑĦĹĽã

    ſı IĮĨĻĨĞĬĨŅĦ Ğ ĬĨĽĻ ĢIJĮĻĞĨĮĨĮĺ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽf
    ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ ł äã

    ſı IĻĦĹĞĻĦ ĻÎĦ IJĠÏĦĢĻĽ
    ÌIJĹ Ĩ ĨĮ ĹĞĮÍĦâĬĞÑĦĹçĢIJŁĮĻãf

        ſı IÌ ĨĻďĽ ĻÎĦ ÌĨĹĽĻ ĬĞÑĦĹd
        ſı ĻÎĦ ĴĹĦŁĬIJŁĽ ĬĞÑĦĹ IJĠÏĦĢĻ ĨĽ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
        ĨÌ Ĩ łł ſlf
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦŁ ł ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦŇĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝã

        ſı AĬĬ ĬĞÑĦĹĽ ĦŇĢĦĴĻ ÌIJĹ ĻÎĦ ÌĨĹĽĻ ĞĮĤ ĻÎĦ ĬĞĽĻ
        ĦĬĨÌ Ĩ l· ĬĞÑĦĹçĢIJŁĮĻ h ĝf
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦŁ ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝã
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦŇĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝã

        ſı TÎĦ ĬĞĽĻ ĬĞÑĦĹ h ĻÎĦ ĮĦŇĻ IJĠÏĦĢĻ ĨĽ ĻÎĦ ĬIJĹĽ
        ſı AĬĹIJ ĬĦĻďĽ ĽĞŁĦ ĞĽĨÎĦ ĻÎĦ ĹĦÌĦĹĦĮĢĦ ĻIJ ĻÎĦ ĬĞĽĻ IJĠÏĦĢĻ
        ſı ŃÎIJĹĦ IJŁĻĴŁĻ ĨĽ ĻÎĦ İIJĤĦĬďĽ IJŁĻĴŁĻ
        ĦĬĽĦf
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦŁ ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝã
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦŇĻ ł ĽĦĬÌgĬIJĹĽ
            ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨŁĞĻĨIJĮ ł ĽĦĬÌgĬĞÑĦĹĽäĨā

        ſı IÌ ĬĞÑĦĹ ĢIJĮĻĞĨĮĽ ĞĮ ĞĻĻĹĨĠŁĻĦ ĢĞĬĬĦĤ ðŃĦĨÍÎĻĽðd
        ſı ĨĻďĽ Ğ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹ h
        ſı ĞĤĤ ĨĻ ĻIJ ĻÎĦ ĬĨĽĻ IJÌ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
        ſı WĦ ĤIJĮďĻ ĮĦĦĤ ĻIJ ĢÎĦĢĨ ÌIJĹ ĠĨĞĽĦĽ h
        ſı ĢÎĦĢĨĨĮÍ ÌIJĹ ŃĦĨÍÎĻĽ ĨĽ ĦĮIJŁÍÎ
        ĨÌ ÎĞĽĞĻĻĹâĽĦĬÌgĬĞÑĦĹĽäĨād dŃĦĨÍÎĻĽďãf
            ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽgĞĴĴĦĮĤâĽĦĬÌgĬĞÑĦĹĽäĨāã
```

Next, we'll modify the common LIJĹĽ class to contain the following:

```
# CIJİİIJĮ ĬIJĹĽ ĢĬĞĹĽ
ĢĬĞĹĽ LIJĹĽf
    ggg
    # SHĻwĹHİHİĠHĹ ĻĹĞĨĮĞĠĬH ĬĞÑHĹĽ
    ÂHÌ ĹHİHİĠHĹçĻĹĞĨĮĞĠĬHçĬĞÑHĹĽâĹHĬÌd ĻĹĞĨĮĞĠĬHçĬĞÑHĹĽãf
        ĽHĬÌgĻĹĞĨĮĞĠĬHçĬĞÑHĹĽ ł ĻĹĞĨĮĞĠĬHçĬĞÑHĹĽ

    # CĞĬĢŁĬĞĻHĽ ĻÎH ÂĞĻĞ ĞĮÂ ĹHÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽHĽ
    # ÍĨĿHĮ ĬIJÂHĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ĿĞĬŁHĽ
    ÂHÌ ĢĞĬĢŁĬĞĻHâĹHĬÌd IJŁĻĴŁĻd Ñãf

        # CĞĬĢŁĬĞĻH ĽĞİĴĬH ĬIJĹĽHĽ
        ĽĞİĴĬHçĬIJĹĽHĽ ł ĽHĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        # CĞĬĢŁĬĞĻH İHĞĮ ĬIJĹĽ
        ÂĞĻĞçĬIJĹĽ ł ĮĴgİHĞĮâĽĞİĴĬHçĬIJĹĽHĽã

        # RHĻŁĹĮ ĻÎH ÂĞĻĞ ĞĮÂ ĹHÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽHĽ
        ĹHĻŁĹĮ ÂĞĻĞçĬIJĹĽd ĽHĬÌgĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽâã
```

The ĹHİHİĠHĹçĻĹĞĨĮĞĠĬHçĬĞÑHĹĽ method in the common LIJĹĽ class "tells" which layers in the MIJÂHĬ object are trainable. The ĢĞĬĢŁĬĞĻH method was modified to also return the ĽHĬÌgĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽâã during a single call. The ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ method currently requires a layer object, but with the ĽHĬÌgĻĹĞĨĮĞĠĬHçĬĞÑHĹĽ property set in ĹHİHİĠHĹçĻĹĞĨĮĞĠĬHçĬĞÑHĹĽ, method we can now iterate over the trainable layers to compute regularization loss for the entire model, rather than one layer at a time:

```
ĢĬĞĹĽ LIJĹĽf
    ggg
    # RHÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽ ĢĞĬĢŁĬĞĻĨIJĮ
    ÂHÌ ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽâĹHĬÌã f

        # fl ĠÑ ÂHÌĞŁĬĻ
        ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ł fl

        # CĞĬĢŁĬĞĻH ĹHÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽ
        # ĨĻHĹĞĻH ĞĬĬ ĻĹĞĨĮĞĠĬH ĬĞÑHĹĽ
        ÌIJĹ ĬĞÑHĹ ĨĮ ĽHĬÌgĻĹĞĨĮĞĠĬHçĬĞÑHĹĽf

            # Lĝ ĹHÍŁĬĞĹĨŅĞĻĨIJĮ h ŃHĨÍÎĻĽ
            # ĢĞĬĢŁĬĞĻH IJĮĬÑ ŃÎHĮ ÌĞĢĻIJĹ ÍĹHĞĻHĹ ĻÎĞĮ fl
            ĨÌ ĬĞÑHĹgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬĝ ń flf
                ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ κł ĬĞÑHĹgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬĝ ắ ċ
                                        ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑHĹgŃHĨÍÎĻĽãã
```

```
        ƒ Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ŃĦĨÍÎĻĽ
        ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ƒf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ĸł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                    ĮĴgĽŁİâĬĞÑĦĹgŃĦĨÍÎĻĽ ő ċ
                                            ĬĞÑĦĹgŃĦĨÍÎĻĽã

        ƒ Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
        ƒ ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ ƒl
        ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń ƒf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ĸł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ő ċ
                                    ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgĠĨĞĽĦĽãã

        ƒ Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
        ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ƒf
            ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ĸł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                    ĮĴgĽŁİâĬĞÑĦĹgĠĨĞĽĦĽ ő ċ
                                            ĬĞÑĦĹgĠĨĞĽĦĽã

    ĹĦĻŁĹĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ
```

For calculating accuracy, we need predictions. So far, predicting has required different code depending on the type of model. For a softmax classifier, we do a `ĮĴgĞĹÍİĞŃâã`, but for regression, the prediction is the direct output because of the linear activation function being used in an output layer. Ideally, we'd have a prediction method that would choose the appropriate method for our model. To do this, we'll add a `ĴĹĦĤĨĢĻĨIJĮĽ` method to each activation function class:

```
ƒ SIJÌĻİĞŃ ĞĢĻĨĹĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĹĞĻĨIJĮçSIJÌĻİĞŃƒ
    ggg
    ƒ CĞĬĢŁĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ĤĦÌ ĴĹĦĤĨĢĻĨIJĮĽâĽĦĬÌd IJŁĻĴŁĻĽãƒ
        ĹĦĻŁĹĮ ĮĴgĞĹÍİĞŃâIJŁĻĴŁĻĽd ĞŃĨĽłĝã


ƒ SĨÍİIJĨĤ ĞĢĻĨĹĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĹĞĻĨIJĮçSĨÍİIJĨĤƒ
    ggg
    ƒ CĞĬĢŁĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ĤĦÌ ĴĹĦĤĨĢĻĨIJĮĽâĽĦĬÌd IJŁĻĴŁĻĽãƒ
        ĹĦĻŁĹĮ âIJŁĻĴŁĻĽ ń flghãã ő ĝ
```

```
# LĨĮĦĞĹ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĹĹ AĢĻĨĿĞĻĨIJĮçLĨĮĦĞĹf
    ggg
    # CĞĬĢŁĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĹ ÌIJĹ IJŁĻĴŁĻĹ
    ĤĦÌ ĴĹĦĤĨĢĻĨIJĮĹâĹĦĬÌd IJŁĻĴŁĻĹãf
        ĹĦĻŁĹĮ IJŁĻĴŁĻĹ
```

All the computations made inside the `ĴĹĦĤĨĢĻĨIJĮĹ` functions are the same as those performed with appropriate models in previous chapters. While we have no plans for using the ReLU activation function for an output layer's activation function, we'll include it here for completeness:

```
# RĦLU ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĹĹ AĢĻĨĿĞĻĨIJĮçRĦLUf
    ggg
    # CĞĬĢŁĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĹ ÌIJĹ IJŁĻĴŁĻĹ
    ĤĦÌ ĴĹĦĤĨĢĻĨIJĮĹâĹĦĬÌd IJŁĻĴŁĻĹãf
        ĹĦĻŁĹĮ IJŁĻĴŁĻĹ
```

We still need to set a reference to the activation function for the final layer in the `MIJĤĦĬ` object. We can later call the `ĴĹĦĤĨĢĻĨIJĮĹ` method, which will return predictions calculated from the outputs. We'll set this in the `MIJĤĦĬ` class' `ÌĨĮĞĬĨĢĦ` method:

```
# MIJĤĦĬ ĢĬĞĹĹ
ĢĬĞĹĹ MIJĤĦĬf
    ggg
    ĤĦÌ ÌĨĮĞĬĨĢĦâĹĦĬÌãf
        ggg
            # TĨĦ ĬĞĹĻ ĬĞÑĦĹ h ĻĨĦ ĮĦŇĻ IJĠĬĦĢĻ ĨĹ ĻĨĦ ĬIJĹĹ
            # AĬĹIJ ĬĦĻďĹ ĹĞĿĦ ĞĹĨĤĦ ĻĨĦ ĹĦÌĦĹĦĮĢĦ ĻIJ ĻĨĦ ĬĞĹĻ IJĠĬĦĢĻ
            # ŃĨIJĹĦ IJŁĻĴŁĻ ĨĹ ĻĨĦ ĬIJĤĦĬďĹ IJŁĻĴŁĻ
            ĦĬĹĦf
                ĹĦĬÌgĬĞÑĦĹĹäĨāgĴĹĦĿ ł ĹĦĬÌgĬĞÑĦĹĹäĨhĝā
                ĹĦĬÌgĬĞÑĦĹĹäĨāgĮĦŇĻ ł ĹĦĬÌgĬIJĹĹ
                ĹĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĿĞĻĨIJĮ ł ĹĦĬÌgĬĞÑĦĹĹäĨā
```

Just like the different prediction methods, we also calculate accuracy in different ways. We're going to implement this in a way similar to the specific loss class' objects implementation — we'll create specific accuracy classes and their objects, which we'll associate with models.

First, we'll write a common `AĢĢŁĹĞĢÑ` class containing (for now) just a single method, `ĢĞĬĢŁĬĞĻĦ`, returning an accuracy calculated from comparison results. We've already added a call to the `ĹĦĬÌgĢIJĬĴĞĹĦ` method that does not exist yet, but we'll create it soon in other classes that will inherit from this `AĢĢŁĹĞĢÑ` class. For now, it's enough to know that it will return a list of *True* and *False* values, indicating if a prediction matches the ground-truth value. Next, we calculate the mean value (which treats *True* as *1* and *False* as *0*) and return it as an accuracy. The

code:

```
ƒ CIJİİIJĮ ĞĢĢŁĹĞĢÑ ĢĬĞĽĽ
ĢĬĞĽĽ AĢĢŁĹĞĢÑf

    ƒ CĞĬĢŁĬĞĻĦĽ ĞĮ ĞĢĢŁĹĞĢÑ
    ƒ ÍĨĿĦĮ ĴĹĦĤĨĢĻĨIJĮĽ ĞĮĤ ÍĹIJŁĮĤ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd ĴĹĦĤĨĢĻĨIJĮĽd Ñãf

        ƒ GĦĻ ĢIJİĴĞĹĨĽIJĮ ĹĦĽŁĬĻĽ
        ĢIJİĴĞĹĨĽIJĮĽ ł ĽĦĬÌgĢIJİĴĞĹĦâĴĹĦĤĨĢĻĨIJĮĽd Ñã

        ƒ CĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĢIJİĴĞĹĨĽIJĮĽã

        ƒ RĦĻŁĹĮ ĞĢĢŁĹĞĢÑ
        ĹĦĻŁĹĮ ĞĢĢŁĹĞĢÑ
```

Next, we can work with this common AĢĢŁĹĞĢÑ class, inheriting from it, then building further for specific types of models. In general, each of these classes will contain two methods: ĨĮĨĻ (not to be confused with a Python class' ççĨĮĨĻçç method) for initialization from inside the model object and ĢIJİĴĞĹĦ for performing comparison calculations. For regression, the ĨĮĨĻ method will calculate an accuracy precision, the same as we have written previously for the regression model, and have been running before the training loop. The ĢIJİĴĞĹĦ method will contain the actual comparison code we have implemented in the training loop itself, which uses ĽĦĬÌgĴĹĦĢĨĽĨIJĮ. Note that initialization won't recalculate precision unless forced to do so by setting the ĹĦĨĮĨĻ parameter to TĹŁĦ. This allows for multiple use-cases, including setting ĽĦĬÌgĴĹĦĢĨĽĨIJĮ independently, calling ĨĮĨĻ whenever needed (e.g., from outside of the model during its creation), and even calling it multiple times (which will become handy soon):

```
ƒ AĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ ÌIJĹ ĹĦÍĹĦĽĽĨIJĮ İIJĤĦĬ
ĢĬĞĽĽ AĢĢŁĹĞĢÑçRĦÍĹĦĽĽĨIJĮâAĢĢŁĹĞĢÑãf

    ÂĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ƒ CĹĦĞĻĦ ĴĹĦĢĨĽĨIJĮ ĴĹIJĴĦĹĻÑ
        ĽĦĬÌgĴĹĦĢĨĽĨIJĮ ł NIJĮĦ

    ƒ CĞĬĢŁĬĞĻĦĽ ĴĹĦĢĨĽĨIJĮ ĿĞĬŁĦ
    ƒ ĠĞĽĦĤ IJĮ ĴĞĽĽĦĤ ĨĮ ÍĹIJŁĮĤ ĻĹŁĻÎ
    ÂĦÌ ĨĮĨĻâĽĦĬÌd Ñd ĹĦĨĮĨĻłFĞĬĽĦãf
        ĨÌ ĽĦĬÌgĴĹĦĢĨĽĨIJĮ ĨĽ NIJĮĦ IJĹ ĹĦĨĮĨĻf
            ĽĦĬÌgĴĹĦĢĨĽĨIJĮ ł ĮĴgĽĻĤâÑã w ğĥfl

    ƒ CIJİĴĞĹĦĽ ĴĹĦĤĨĢĻĨIJĮĽ ĻIJ ĻÎĦ ÍĹIJŁĮĤ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ÂĦÌ ĢIJİĴĞĹĦâĽĦĬÌd ĴĹĦĤĨĢĻĨIJĮĽd Ñãf
        ĹĦĻŁĹĮ ĮĴgĞĠĽIJĬŁĻĦâĴĹĦĤĨĢĻĨIJĮĽ h Ñã l· ĽĦĬÌgĴĹĦĢĨĽĨIJĮ
```

We can then set the accuracy object from within the ĽĦĻ method in our MIJÎĦĬ class the same way as the loss and optimizer currently:

```
# MIJÎĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJÎĦĬf
    ggg
    # SĦĻ ĬIJĽĽd IJĴĻĨĬĨĨŅĦĹ ĞĮÎ ĞĢĢŁĹĞĢÑ
    ÎĦÌ ĽĦĻâĽĦĬÌd ǻd ĬIJĽĽd IJĴĻĨĬĨŅĦĹd ĞĢĢŁĹĞĢÑãf
        ĽĦĬÌgĬIJĽĽ ł ĬIJĽĽ
        ĽĦĬÌgIJĴĻĨĬĨŅĦĹ ł IJĴĻĨĬĨŅĦĹ
        ĽĦĬÌgĞĢĢŁĹĞĢÑ ł ĞĢĢŁĹĞĢÑ
```

Then we can finally add the loss and accuracy calculations to our model right after the completed forward pass' code. Note that we also initialize the accuracy with ĽĦĬÌgĞĢĢŁĹĞĢÑgĨĮĨĻâÑã at the beginning of the ĻĹĞĨĮ method, which can be called multiple times — as noted earlier. In the case of regression accuracy, this will invoke a precision calculation a single time during the first call. The code of the ĻĹĞĨĮ method with implemented loss and accuracy calculations:

```
# MIJÎĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJÎĦĬf
    ggg
    # TĹĞĨĮ ĻÎĦ ÌIJÎĦĬ
    ÎĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ǻd ĦĴIJĢÎĽłĝd ĴĹĨĮçĦĽĦĹÑłĝãf

        # IĮĨĻĨĞĬĨŅĦ ĞĢĢŁĹĞĢÑ IJĠÏĦĢĻ
        ĽĦĬÌgĞĢĢŁĹĞĢÑgĨĮĨĻâÑã

        # MĞĨĮ ĻĹĞĨĮĨĮÍ ĬIJIJĴ
        ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝd ĦĴIJĢÎĽĸĝãf

            # PĦĹÌIJĹÌ ĻÎĦ ÌIJĹŃĞĹÎ ĴĞĽĽ
            IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÎâXã

            # CĞĬĢŁĬĞĻĦ ĬIJĽĽ
            ÎĞĻĞçĬIJĽĽd ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ ł ċ
                ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd Ñã
            ĬIJĽĽ ł ÎĞĻĞçĬIJĽĽ ĸ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ

            # GĦĻ ĴĹĦÎĨĢĻĨIJĮĽ ĞĮÎ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
            ĴĹĦÎĨĢĻĨIJĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĽĞĻĨIJĮgĴĹĦÎĨĢĻĨIJĮĽâ
                                IJŁĻĴŁĻã
            ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦÎĨĢĻĨIJĮĽd Ñã
```

Finally, we'll add a call to the previously created method `ĹĦİĦİĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ` with the `LIJĽĽ` class' object, which we'll do in the `ÌĨĨĮĞĬĨÑĦ` method (`ĽĦĬÌgĬIJĽĽgĹĦİĦİĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽâĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽã`). The full model class code so far:

```
ƒ MIJÂĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJÂĦĬf

    ÂĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ƒ CĹĦĞĻĦ Ğ ĬĨĽĻ IJÌ ĮĦĻŃIJĹĨ IJĠÏĦĢĻĽ
        ĽĦĬÌgĬĞÑĦĹĽ ł äā

    ƒ AÂÂ IJĠÏĦĢĻĽ ĻIJ ĻÎĦ ÌIJÂĦĬ
    ÂĦÌ ĞÂÂâĽĦĬÌd ĬĞÑĦĹãf
        ĽĦĬÌgĬĞÑĦĹĽgĞĴĴĦĮÂâĬĞÑĦĹã

    ƒ SĦĻ ĬIJĽĽd IJĴĻĨÌĨÑĦĹ ĞĮÂ ĞĢĢŁĹĞĢÑ
    ÂĦÌ ĽĦĻâĽĦĬÌd ǻd ĬIJĽĽd IJĴĻĨÌĨÑĦĹd ĞĢĢŁĹĞĢÑãf
        ĽĦĬÌgĬIJĽĽ ł ĬIJĽĽ
        ĽĦĬÌgIJĴĻĨÌĨÑĦĹ ł IJĴĻĨÌĨÑĦĹ
        ĽĦĬÌgĞĢĢŁĹĞĢÑ ł ĞĢĢŁĹĞĢÑ

    ƒ FĨĮĞĬĨÑĦ ĻÎĦ ÌIJÂĦĬ
    ÂĦÌ ÌĨĮĞĬĨÑĦâĽĦĬÌãf

        ƒ CĹĦĞĻĦ ĞĮÂ ĽĦĻ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
        ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ ł LĞÑĦĹçIĮĴŁĻâã

        ƒ CIJŁĮĻ ĞĬĬ ĻÎĦ IJĠÏĦĢĻĽ
        ĬĞÑĦĹçĢIJŁĮĻ ł ĬĦĮâĽĦĬÌgĬĞÑĦĹĽã

        ƒ IĮĨĻĨĞĬĨÑĦ Ğ ĬĨĽĻ ĢIJĮĻĞĨĮĨĮĺ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽf
        ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ ł äā

        ƒ IĻĦĹĞĻĦ ĻÎĦ IJĠÏĦĢĻĽ
        ÌIJĹ Ĩ ĨĮ ĹĞĮÍĦâĬĞÑĦĹçĢIJŁĮĻãf

            ƒ IÌ ĨĻďĽ ĻÎĦ ÌĨĹĽĻ ĬĞÑĦĹd
            ƒ ĻÎĦ ĴĹĦŁĨIJŁĽ ĬĞÑĦĹ IJĠÏĦĢĻ ĨĽ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
            ĨÌ Ĩ łł flf
                ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦŁ ł ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ
                ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦÑĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝā

            ƒ AĬĬ ĬĞÑĦĹĽ ĦÑĢĦĴĻ ÌIJĹ ĻÎĦ ÌĨĹĽĻ ĞĮÂ ĻÎĦ ĬĞĽĻ
            ĦĬĨÌ Ĩ l· ĬĞÑĦĹçĢIJŁĮĻ h ĝf
                ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦŁ ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝā
                ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦÑĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝā
```

```
        # TÎĦ ĬĞĽĻ ĬĞÑĦĹ h ĻÎĦ ĮĦŇĻ ĲĠĬĦĢĻ ĨĽ ĻÎĦ ĬĲĽĽ
        # AĬĽĲ ĬĦĻďĽ ĽĞŁĦ ĞĽĨÂĦ ĻÎĦ ĹĦÌĦĹĦĮĢĦ ĻĲ ĻÎĦ ĬĞĽĻ ĲĠĬĦĢĻ
        # ŃÎĲĽĦ ĲŁĻĴŁĻ ĨĽ ĻÎĦ ĬĲÂĦĬďĽ ĲŁĻĴŁĻ
        ĦĬĽĦf
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦŁ· ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝā
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦŇĻ ł ĽĦĬÌgĬĲĽĽ
            ĽĦĬÌgĲŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨŁ·ĞĻĨĲĮ ł ĽĦĬÌgĬĞÑĦĹĽäĨā

        # IÌ ĬĞÑĦĹ ĢĲĮĻĞĨĮĽ ĞĮ ĞĻĻĹĨĠŁĻĦ ĢĞĬĬĦÂ ðŃĦĬĹÎĻĽðd
        # ĨĻďĽ Ğ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹ h
        # ĞÂÂ ĨĻ ĻĲ ĻÎĦ ĬĨĽĻ ĲÌ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
        # WĦ ÂĲĮďĻ ĮĦĦÂ ĻĲ ĢÎĦĢĨ ÌĲĹ ĠĨĞĽĦĽ h
        # ĢÎĦĢĨĨĮÍ ÌĲĹ ŃĦĬĹÎĻĽ ĨĽ ĦĮĲŁĹÎ
        ĨÌ ÎĞĽĞĻĻĹâĽĦĬÌgĬĞÑĦĹĽäĨād dŃĦĬĹÎĻĽďãf
            ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽgĞĴĴĦĮÂâĽĦĬÌgĬĞÑĦĹĽäĨāã

    # UĴÂĞĻĦ ĬĲĽĽ ĲĠĬĦĢĻ ŃĨĻÎ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
    ĽĦĬÌgĬĲĽĽgĹĦĬĦĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽâ
        ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ
    ã

# TĹĞĨĮ ĻÎĦ ĬĲÂĦĬ
ÂĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ấd ĦĴĲĢÎĽłĝd ĴĹĨĮĻçĦŁĦĹÑłĝãf

    # IĮĨĻĨĞĬĨŅĦ ĞĢĢŁĹĞĢÑ ĲĠĬĦĢĻ
    ĽĦĬÌgĞĢĢŁĹĞĢÑgĨĮĨĻâÑã

    # MĞĨĮ ĻĹĞĨĮĨĮÍ ĬĲĲĴ
    ÌĲĹ ĦĴĲĢÎ ĨĮ ĹĞĮÍĦâĝd ĦĴĲĢÎĽĸĝãf

        # PĦĹÌĲĹÌ ĻÎĦ ÌĲĹŃĞĹÂ ĴĞĽĽ
        ĲŁĻĴŁĻ ł ĽĦĬÌgÌĲĹŃĞĹÂâXã

        # CĞĬĢŁĬĞĻĦ ĬĲĽĽ
        ÂĞĻĞçĬĲĽĽd ĹĦÍŁĬĞĹĨŅĞĻĨĲĮçĬĲĽĽ ł ċ
            ĽĦĬÌgĬĲĽĽgĢĞĬĢŁĬĞĻĦâĲŁĻĴŁĻd Ñã
        ĬĲĽĽ ł ÂĞĻĞçĬĲĽĽ ĸ ĹĦÍŁĬĞĹĨŅĞĻĨĲĮçĬĲĽĽ

        # GĦĻ ĴĹĦÂĨĢĻĨĲĮĽ ĞĮÂ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĴĹĦÂĨĢĻĨĲĮĽ ł ĽĦĬÌgĲŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨŁ·ĞĻĨĲĮgĴĹĦÂĨĢĻĨĲĮĽâ
                            ĲŁĻĴŁĻã
        ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦÂĨĢĻĨĲĮĽd Ñã
```

```
    fi PHĹÌIJĹİĹ ÌIJĹŃĞĹÎ ĴĞĹĹ
    ÎHÌ ÌIJĹŃĞĹÎâĹHĬÌd Xãf

        fi CĞĬĬ ÌIJĹŃĞĹÎ İHĻÎIJÎ IJĮ ĻÎH ĬĮĴŁĻ ĬĞÑHĹ
        fi ĻÎĬĹ ŃĬĬĬ ĹHĻ ĻÎH IJŁĻĴŁĻ ĴĹIJĴHĹĻÑ ĻÎĞĻ
        fi ĻÎH ÌĬĹĹĻ ĬĞÑHĹ ĬĮ ðĴĹHŁð IJĠÏHĢĻ ĬĹ HŃĴHĢĻĬĮÍ
        ĹHĬÌgĬĮĴŁĻçĬĞÑHĹgÌIJĹŃĞĹÎâXã

        fi CĞĬĬ ÌIJĹŃĞĹÎ İHĻÎIJÎ IJÌ HŁHĹÑ IJĠÏHĢĻ ĬĮ Ğ ĢÎĞĬĮ
        fi PĞĹĹ IJŁĻĴŁĻ IJÌ ĻÎH ĴĹHŁĬIJŁĹ IJĠÏHĢĻ ĞĹ Ğ ĴĞĹĞİHĻHĹ
        ÌIJĹ ĬĞÑHĹ ĬĮ ĹHĬÌgĬĞÑHĹĹf
            ĬĞÑHĹgÌIJĹŃĞĹÎâĬĞÑHĹgĴĹHŁgIJŁĻĴŁĻã

        fi ðĬĞÑHĹð ĬĹ ĮIJŃ ĻÎH ĬĞĹĻ IJĠÏHĢĻ ÌĹIJİ ĻÎH ĬĬĹĻd
        fi ĹHĻŁĹĮ ĬĻĹ IJŁĻĴŁĻ
        ĹHĻŁĹĮ ĬĞÑHĹgIJŁĻĴŁĻ
```

Full code for the LIJĹĹ class:

```
fi CIJİİIJĮ ĬIJĹĹ ĢĬĞĹĹ
ĢĬĞĹĹ LIJĹĹf

    fi RHĺŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĹ ĢĞĬĢŁĬĞĻĨIJĮ
    ÎHÌ ĹHĺŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹâĹHĬÌãf

        fi fl ĠÑ ÎHÌĞŁĬĻ
        ĹHĺŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ ł fl

        fi CĞĬĢŁĬĞĻH ĹHĺŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĹ
        fi ĨĻHĹĞĻH ĞĬĬ ĻĹĞĨĮĞĠĬH ĬĞÑHĹĹ
        ÌIJĹ ĬĞÑHĹ ĬĮ ĹHĬÌgĻĹĞĨĮĞĠĬHçĬĞÑHĹĹf

            fi Lĝ ĹHĺŁĬĞĹĨŅĞĻĨIJĮ h ŃHĨĺÎĻĹ
            fi ĢĞĬĢŁĬĞĻH IJĮĬÑ ŃÎHĮ ÌĞĢĻIJĹ ĺĹHĞĻHĹ ĻÎĞĮ fl
            ĨÌ ĬĞÑHĹgŃHĨĺÎĻçĹHĺŁĬĞĹĨŅHĹçĬĝ ń flf
                ĹHĺŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ κł ĬĞÑHĹgŃHĨĺÎĻçĹHĺŁĬĞĹĨŅHĹçĬĝ ő ċ
                                        ĮĴgĹŁİâĮĴgĞĠĹâĬĞÑHĹgŃHĨĺÎĻĹãã

            fi Lğ ĹHĺŁĬĞĹĨŅĞĻĨIJĮ h ŃHĨĺÎĻĹ
            ĨÌ ĬĞÑHĹgŃHĨĺÎĻçĹHĺŁĬĞĹĨŅHĹçĬğ ń flf
                ĹHĺŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ κł ĬĞÑHĹgŃHĨĺÎĻçĹHĺŁĬĞĹĨŅHĹçĬğ ő ċ
                                        ĮĴgĹŁİâĬĞÑHĹgŃHĨĺÎĻĹ ő ċ
                                                ĬĞÑHĹgŃHĨĺÎĻĹã
```

```
            fi Lĝ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
            fi IJĮĬÑ ĢĞĬĢŁĬĞĻĦ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ fl
            ĨÌ ĬĞÑĦĹgĠĬĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
                ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgĠĬĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ő ċ
                                          ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgĠĬĞĽĦĽãã

            fi Lğ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĽĦĽ
            ĨÌ ĬĞÑĦĹgĠĬĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
                ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgĠĬĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                          ĮĴgĽŁİâĬĞÑĦĹgĠĬĞĽĦĽ ő ċ
                                                 ĬĞÑĦĹgĠĬĞĽĦĽã

        ĹĦĻŁĹĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ

    fi SĦĻwĹĦİĦİĠĦĹ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
    ÎĦÌ ĹĦİĦİĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽâĽĦĬÌd ĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽãf
        ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ ł ĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ

    fi CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÎĞĻĞ ĞĮÎ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽĦĽ
    fi ÍĨĿĦĮ ÌIJÎĦĬ IJŁĻĴŁĻ ĞĮÎ ÍĹIJŁĮÎ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ÎĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñãf

        fi CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦ ĬIJĽĽĦĽ
        ĽĞİĴĬĦçĬIJĽĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÎâIJŁĻĴŁĻd Ñã

        fi CĞĬĢŁĬĞĻĦ İĦĞĮ ĬIJĽĽ
        ÎĞĻĞçĬIJĽĽ ł ĮĴgİĦĞĮâĽĞİĴĬĦçĬIJĽĽĦĽã

        fi RĦĻŁĹĮ ĻÎĦ ÎĞĻĞ ĞĮÎ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽĦĽ
        ĹĦĻŁĹĮ ÎĞĻĞçĬIJĽĽd ĽĦĬÌgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽâã
```

Now that we've done a full forward pass and have calculated loss and accuracy, we can begin the backward pass. The `ĠĞĢĪŃĞĹÎ` method in the `MIJÎĦĬ` class is structurally similar to the `ÌIJĹŃĞĹÎ` method, just in reverse and using different parameters. Following the backward pass in our previous training approach, we need to call the `ĠĞĢĪŃĞĹÎ` method of a loss object to create the `ÎĨĮĴŁĻĽ` property. Next, we'll loop through all the layers in reverse order, calling their `ĠĞĢĪŃĞĹÎ` methods with the `ÎĨĮĴŁĻĽ` property of the next layer (in normal order) as a parameter, effectively backpropagating the gradient returned by that next layer. Remember that we have set the loss object as a `ĮĦŃĻ` layer in the last, output layer.

```
# MIJÂHĬ ĢĬĞĹĹ
ĢĬĞĹĹ MIJÂHĬf
    ggg
    # PHĹÌIJĹİĹ ĠĞĢĪŃĞĹÂ ĴĞĹĹ
    ÂHÌ ĠĞĢĪŃĞĹÂâĹHĬÌd IJŁĻĴŁĻd Ñãf

        # FĨĹĹĻ ĢĞĬĬ ĠĞĢĪŃĞĹÂ ĮHĻÎIJÂ IJĮ ĻÎH ĬIJĹĹ
        # ĻÎĨĹ ŃĨĬĬ ĹHĻ ÂĨĮĴŁĻĹ ĴĹIJĴHĹĻÑ ĻÎĞĻ ĻÎH ĬĞĹĻ
        # ĬĞÑHĹ ŃĨĬĬ ĻĹÑ ĻIJ ĞĢĢHĹĹ ĹÎIJĹĻĬÑ
        ĹHĬÌgĬIJĹĹgĠĞĢĪŃĞĹÂâIJŁĻĴŁĻd Ñã

        # CĞĬĬ ĠĞĢĪŃĞĹÂ ĮHĻÎIJÂ ÍIJĨĮÍ ĻÎĹIJŁÍÎ ĞĬĬ ĻÎH IJĠÏHĢĻĹ
        # ĨĮ ĹHŁHĹĹHÂ IJĹÂHĹ ĴĞĹĹĨĮÍ ÂĨĮĴŁĻĹ ĞĹ Ğ ĴĞĹĞĮHĻHĹ
        ÌIJĹ ĬĞÑHĹ ĨĮ ĹHŁHĹĹHÂâĹHĬÌgĬĞÑHĹĹãf
            ĬĞÑHĹgĠĞĢĪŃĞĹÂâĬĞÑHĹgĮHÑĻgÂĨĮĴŁĻĹã
```

Next, we'll add a call of this ĠĞĢĪŃĞĹÂ method to the end of the ĻĹĞĨĮ method:

```
# PHĹÌIJĹİ ĠĞĢĪŃĞĹÂ ĴĞĹĹ
ĹHĬÌgĠĞĢĪŃĞĹÂâIJŁĻĴŁĻd Ñã
```

After this backward pass, the last action to perform is to optimize. We have previously been calling the optimizer object's ŁĴÂĞĻHçĴĞĹĞİĹ method as many times as we had trainable layers. We have to make this code universal as well by looping through the list of trainable layers and calling ŁĴÂĞĻHçĴĞĹĞİĹâã in this loop:

```
# OĴĻĨİĨÑH âŁĴÂĞĻH ĴĞĹĞİHĻHĹĹã
ĹHĬÌgIJĴĻĨİĨÑHĹgĴĹHçŁĴÂĞĻHçĴĞĹĞİĹâã
ÌIJĹ ĬĞÑHĹ ĨĮ ĹHĬÌgĻĹĞĨĮĞĠĬHçĬĞÑHĹĹf
    ĹHĬÌgIJĴĻĨİĨÑHĹgŁĴÂĞĻHçĴĞĹĞİĹâĬĞÑHĹã
ĹHĬÌgIJĴĻĨİĨÑHĹgĴIJĹĻçŁĴÂĞĻHçĴĞĹĞİĹâã
```

Then we can output useful information — here's where this last parameter to the ĻĹĞĨĮ method becomes handy:

```
# PĹĨĮĻ Ğ ĹŁİİĞĹÑ
ĨÌ ĮIJĻ HĴIJĢÎ ķ ĴĹĨĮĻçHŁHĹÑf
    ĴĹĨĮĻâÌďHĴIJĢÎf ăHĴIJĢÎåd ď κ
          ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
          ÌďĬIJĹĹf ăĬIJĹĹfgġÌå âď κ
          ÌďÂĞĻĞçĬIJĹĹf ăÂĞĻĞçĬIJĹĹfgġÌåd ď κ
          ÌďĹHÍçĬIJĹĹf ăĹHÍŁĬĞĹĨŃĞĻĨIJĮçĬIJĹĹfgġÌåãd ď κ
          ÌďĬĹf ăĹHĬÌgIJĴĻĨİĨÑHĹgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻHåďã
```

```
ƒ TĹĞĨĮ ĻÎĦ İIJĤĦĬ
ÎĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ǻd ĦĴIJĢÎĽłĝd ĴĹĨĮĻçĦĿĦĹÑłĝãf

    ƒ IĮĨĻĨĞĬĨŅĦ ĞĢĢŁĹĞĢÑ IJĠÏĦĢĻ
    ĽĦĬÌgĞĢĢŁĹĞĢÑgĨĮĨĻâÑã

    ƒ MĞĨĮ ĻĹĞĨĮĨĮĬ ĬIJIJĴ
    ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮĬĦâĝd ĦĴIJĢÎĽĸĝãf

        ƒ PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹĤ ĴĞĽĽ
        IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹĤâXã

        ƒ CĞĬĢŁĬĞĻĦ ĬIJĽĽ
        ĤĞĻĞçĬIJĽĽd ĹĦĬŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ ł ċ
            ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd Ñã
        ĬIJĽĽ ł ĤĞĻĞçĬIJĽĽ ĸ ĹĦĬŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ

        ƒ GĦĻ ĴĹĦĤĨĢĻĨIJĮĽ ĞĮĤ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĴĹĦĤĨĢĻĨIJĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĿĞĻĨIJĮgĴĹĦĤĨĢĻĨIJĮĽâ
                            IJŁĻĴŁĻã
        ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦĤĨĢĻĨIJĮĽd Ñã

        ƒ PĦĹÌIJĹİ ĠĞĢĪŃĞĹĤ ĴĞĽĽ
        ĽĦĬÌgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd Ñã

        ƒ OĴĻĨİĨŅĦ âŁĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽã
        ĽĦĬÌgIJĴĻĨİĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâã
        ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽf
            ĽĦĬÌgIJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĬĞÑĦĹã
        ĽĦĬÌgIJĴĻĨİĨŅĦĹgĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâã

        ƒ PĹĨĮĻ Ğ ĽŁİĬĞĹÑ
        ĨÌ IJIJĻ ĦĴIJĢÎ ķ ĴĹĨĮĻçĦĿĦĹÑf
            ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď ĸ
                  ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď ĸ
                  ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď ĸ
                  ÌďĤĞĻĞçĬIJĽĽf ăĤĞĻĞçĬIJĽĽfgġÌåd ď ĸ
                  ÌďĹĦĹçĬIJĽĽf ăĹĦĹŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽfgġÌåãd ď ĸ
                  ÌďĬĹf ăĽĦĬÌgIJĴĻĨİĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦåďã
```

We can now pass the accuracy class' object into the model and test our model's performance:

```
ſi CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĨĮĦçĤĞĻĞâã

ſi IĮĽĻĞĮĻĨĞĻĦ ĻĨĦ ĲĴĤĦĬ
ĲĴĤĦĬ ł MĴĤĦĬâã

ſi AĤĤ ĬĞÑĦĹĽ
ĲĴĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝd ħǵãã
ĲĴĤĦĬgĞĤĤâAĢĻĨĿĞĻĨĴĮçRĦLUâãã
ĲĴĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâħǵd ħǵãã
ĲĴĤĦĬgĞĤĤâAĢĻĨĿĞĻĨĴĮçRĦLUâãã
ĲĴĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâħǵd ĝãã
ĲĴĤĦĬgĞĤĤâAĢĻĨĿĞĻĨĴĮçLĨĮĦĞĹâãã

ſi SĦĻ ĬĴĽĽd ĴĴĻĨĲĨĢĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ ĴĠĲĦĢĻĽ
ĲĴĤĦĬgĽĦĻâ
    ĬĴĽĽłLĴĽĽçMĦĞĮSĶŁĞĹĦĤEĹĹĴĽâãd
    ĴĴĻĨĲĨĢĦĹłOĴĻĨĲĨĢĦĹçAĤĞĲâĬĦĞĹĮĨĮĺçĹĞĻĦłflgflflĥd ĤĦĢĞÑłĝĦhġãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçRĦĺĹĦĽĽĨĴĮâã
ã

ſi FĨĮĞĬĨĢĦ ĻĨĦ ĲĴĤĦĬ
ĲĴĤĦĬgÌĨĮĞĬĨĢĦâã

ſi TĹĞĨĮ ĻĨĦ ĲĴĤĦĬ
ĲĴĤĦĬgĻĹĞĨĮâXd Ñd ĦĴĴĢĤĽłĝflflflfld ĴĹĨĮçĦŁĦĹÑłĝflflã


ńńń
ĦĴĴĢĤf ĝflfld ĞĢĢf flgflflĥd ĬĴĽĽf flgflíĥ âĤĞĻĞçĬĴĴĽĽf flgflíĥd ĹĦĺçĬĴĴĽĽf flgflflfãd ĬĹf
flgflflǵĥǵîĥîflĥǵĥíĥĝĥíǵ
ĦĴĴĢĤf ĝflfld ĞĢĢf flgflǵĝd ĬĴĴĽĽf flgflǵĥ âĤĞĻĞçĬĴĴĽĽf flgflǵĥd ĹĦĺçĬĴĴĽĽf flgflflfãd ĬĹf
flgflflǵĝìflĝǵĝìíǵíĝflĥíǵ
ggg
ĦĴĴĢĤf îîflfld ĞĢĢf flgîǵǵd ĬĴĴĽĽf flgflflfl âĤĞĻĞçĬĴĴĽĽf flgflflfld ĹĦĺçĬĴĴĽĽf flgflflfãd
ĬĹf flgflflflǵĥíìĥìĥíǵĝîĝĝĝflĝĥ
ĦĴĴĢĤf ĝflflflfld ĞĢĢf flgîìfld ĬĴĴĽĽf flgflflfl âĤĞĻĞçĬĴĴĽĽf flgflflfld ĹĦĺçĬĴĴĽĽf flgflflfãd
ĬĹf flgflflflǵĥǵĥíĥìíflĥĝĥǵĝîĥǵ
```

Our new model is behaving well, and now we're able to make new models more easily with our `MĴĤĦĬ` class. We have to continue to modify these classes to handle for entirely new models. For example, we haven't yet handled for binary logistic regression. For this, we need to add two things. First, we need to calculate the categorical accuracy:

```
# AĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ ÌIJĹ ĢĬĞĹĹĨÌÌĨĢĞĻĨIJĮ ĬIJÂĦĬ
ĢĬĞĹĹ AĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâAĢĢŁĹĞĢÑãf

    # NIJ ĨĮĨĻĨĞĬĨĨĢĞĻĨIJĮ ĨĹ ĮĦĦÂĦĦÂ
    ÂĦÌ ĨĮĨĻâĹĦĬÌd Ñãf
        ĴĞĹĹ

    # CIJĬĴĞĹĦĹ ĴĹĦÂĨĢĻĨIJĮĹ ĻIJ ĻÎĦ ÍĹIJŁĮÂ ĻĹŁĻÎ ŁĞĬŁĦĹ
    ÂĦÌ ĢIJĬĴĞĹĦâĹĦĬÌd ĴĹĦÂĨĢĻĨIJĮĹd Ñãf
        ĨÌ ĬĦĮâÑgĹÎĞĴĦã łł ğf
            Ñ ł ĮĴgĞĹÍĬĞÑâÑd ĞÑĨĹłĝã
        ĹĦĻŁĹĮ ĴĹĦÂĨĢĻĨIJĮĹ łł Ñ
```

This is the same as the accuracy calculation for classification, just wrapped into a class and with an additional switch parameter. This switch disables one-hot to sparse label conversion while this class is used with the binary cross-entropy model, since this model always require the groundtrue values to be a 2D array and they're not one-hot encoded. Note that we do not perform any initialization here, but the method needs to exist since it’s going to be called from the `ĻĹĞĨĮ` method of the `MIJÂĦĬ` class. The next thing that we need to add is the ability to validate the model using validation data. Validation requires only a forward pass and calculation of loss (just data loss). We’ll modify the `ĢĞĬĢŁĬĞĻĦ` method of the `LIJĹĹ` class to let it calculate the validation loss as well:

```
# CIJĬÌIJĮ ĬIJĹĹ ĢĬĞĹĹ
ĢĬĞĹĹ LIJĹĹf
    ggg
    # CĞĬĢŁĬĞĻĦĹ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮ ĬIJĹĹĦĹ
    # ÍĨĹĦĮ ÌIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ŁĞĬŁĦĹ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĹĦĬÌd IJŁĻĴŁĻd Ñd ǻd ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹĨĢĞĻĨIJĮłFĞĬĹĦãf

        # CĞĬĢŁĬĞĻĦ ĹĞÌĴĬĦ ĬIJĹĹĦĹ
        ĹĞÌĴĬĦçĬIJĹĹĦĹ ł ĹĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        # CĞĬĢŁĬĞĻĦ ÌĦĞĮ ĬIJĹĹ
        ÂĞĻĞçĬIJĹĹ ł ĮĴgÌĦĞĮâĹĞÌĴĬĦçĬIJĹĹĦĹã

        # IÌ ÏŁĹĻ ÂĞĻĞ ĬIJĹĹ h ĹĦĻŁĹĮ ĨĻ
        ĨÌ ĮIJĻ ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹĨĢĞĻĨIJĮf
            ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĹ

        # RĦĻŁĹĮ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĹd ĹĦĬÌgĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĹĹâã
```

We've added a new parameter and condition to return just the data loss, as regularization loss is not being used in this case. To run it, we'll pass predictions and targets the same way as with the training data. We will not return regularization loss by default, which means we need to update the call to this method in the ĻĹĞĨĮ method to include regularization loss during training:

```
# CĞĬĢŁĬĞĻĦ ĬIJĹĽ
ÎĞĻĞçĬIJĹĽd ĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽ ł ċ
    ĽĦĬÌgĬIJĹĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd Ñd
                        ĨĮĢĬŁÎĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮłTĹŁĦã
```

Then we can add the validation code to the ĻĹĞĨĮ method in the MIJÎĦĬ class. We added the *ĽĞĬĨÎĞĻĨIJĮçÎĞĻĞ* parameter to the function, which takes a tuple of validation data (samples and targets), the ĨÌ statement to check if the validation data is present, and if it is — the code to perform a forward pass over these data, calculate loss and accuracy in the same way as during training and print the results:

```
# MIJÎĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJÎĦĬf
    ggg
    # TĹĞĨĮ ĻÎĦ ĬIJÎĦĬ
    ÎĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ắd ĦĴIJĢÎĹłĝd ĴĹĨĮçĦĽĦĹÑłĝd
            ĽĞĬĨÎĞĻĨIJĮçÎĞĻĞłNIJĮĦãf
        ggg
        # IÌ ĻÎĦĹĦ ĨĽ ĻÎĦ ĽĞĬĨÎĞĻĨIJĮ ÎĞĻĞ
        ĨÌ ĽĞĬĨÎĞĻĨIJĮçÎĞĻĞ ĨĽ ĮIJĻ NIJĮĦf

            # FIJĹ ĠĦĻĻĦĹ ĹĦĞÎĞĠĨĬĨĨĻÑ
            XçĽĞĬd ÑçĽĞĬ ł ĽĞĬĨÎĞĻĨIJĮçÎĞĻĞ

            # PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÎ ĴĞĽĽ
            IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÎâXçĽĞĬã

            # CĞĬĢŁĬĞĻĦ ĻÎĦ ĬIJĹĽ
            ĬIJĹĽ ł ĽĦĬÌgĬIJĹĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ÑçĽĞĬã

            # GĦĻ ĴĹĦÎĬĢĻĨIJĮĹ ĞĮÎ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
            ĴĹĦÎĬĢĻĨIJĮĹ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĽĞĻĨIJĮgĴĹĦÎĬĢĻĨIJĮĹâ
                                IJŁĻĴŁĻã
            ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦÎĬĢĻĨIJĮĹd ÑçĽĞĬã

            # PĹĨĮĻ Ğ ĽŁĬĬĞĹÑ
            ĴĹĨĮĻâÌďĽĞĬĨÎĞĻĨIJĮd ď κ
                  ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
                  ÌďĬIJĹĽf ăĬIJĹĽfgġÌåďã
```

The full ĻĹĞĨĮ method for the MIJĤĦĬ class:

```
# MIJĤĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJĤĦĬf
    ggg
    # TĹĞĨĮ ĻĨĦ ĬIJĤĦĬ
    ĤĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ấd ĦĴIJĢĨĽłĝd ĴĹĨĮĻçĦŁĦĹÑłĝd
            ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞłNIJĮĦãf

        # IĮĨĻĨĞĬĨĢĦ ĞĢĢłĹĞĢÑ IJĠÏĦĢĻ
        ĽĦĬÌgĞĢĢłĹĞĢÑgĨĮĨĻâÑã

        # MĞĨĮ ĻĹĞĨĮĨĮĬ ĬIJIJĴ
        ÌIJĹ ĦĴIJĢĨ ĨĮ ĹĞĮĬĦâĝd ĦĴIJĢĨĽĸĝãf

            # PĦĹÌIJĹĬ ĻĨĦ ÌIJĹŃĞĹĤ ĴĞĽĽ
            IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹĤâXã

            # CĞĬĢŁĬĞĻĦ ĬIJĽĽ
            ĤĞĻĞçĬIJĽĽd ĹĦĬŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ ł ċ
                ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd Ñd
                                    ĨĮĢĬŁĤĦçĹĦĬŁĬĞĹĨĢĞĻĨIJĮłTĹŁĦã
            ĬIJĽĽ ł ĤĞĻĞçĬIJĽĽ ĸ ĹĦĬŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ

            # GĦĻ ĴĹĦĤĨĢĻĨIJĮĽ ĞĮĤ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢłĹĞĢÑ
            ĴĹĦĤĨĢĻĨIJĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĻĞĻĨIJĮgĴĹĦĤĨĢĻĨIJĮĽâ
                              IJŁĻĴŁĻã
            ĞĢĢłĹĞĢÑ ł ĽĦĬÌgĞĢĢłĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦĤĨĢĻĨIJĮĽd Ñã

            # PĦĹÌIJĹĬ ĠĞĢĪŃĞĹĤ ĴĞĽĽ
            ĽĦĬÌgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd Ñã

            # OĴĻĨĬĨĨĦ âŁĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĽã
            ĽĦĬÌgIJĴĻĨĬĨĨĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĽâã
            ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽf
                ĽĦĬÌgIJĴĻĨĬĨĨĦĹgŁĴĤĞĻĦçĴĞĹĞĬĽâĬĞÑĦĹã
            ĽĦĬÌgIJĴĻĨĬĨĨĦĹgĴIJĹĻçŁĴĤĞĻĦçĴĞĹĞĬĽâã

            # PĹĨĮĻ Ğ ĽŁĬĬĞĹÑ
            ĨÌ ĮIJĻ ĦĴIJĢĨ ĸ ĴĹĨĮĻçĦŁĦĹÑf
                ĴĹĨĮĻâÌďĦĴIJĢĨf ăĦĴIJĢĨåd ď ĸ
                      ÌďĞĢĢf ăĞĢĢłĹĞĢÑfgġÌåd ď ĸ
                      ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď ĸ
                      ÌďĤĞĻĞçĬIJĽĽf ăĤĞĻĞçĬIJĽĽfgġÌåd ď ĸ
                      ÌďĹĦÍçĬIJĽĽf ăĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽfgġÌåãd ď ĸ
                      ÌďĬĹf ăĽĦĬÌgIJĴĻĨĬĨĨĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã
```

```
ƒ IÌ ĻÎĦĹĦ ĨĽ ĻÎĦ ĿĞĬĨĤĞĻĨIJĮ ĤĞĻĞ
ĨÌ ĿĞĬĨĤĞĻĨIJĮçĤĞĻĞ ĨĽ ĮIJĻ NIJĮĦf

    ƒ FIJĹ ĠĦĻĻĦĹ ĹĦĞĤĞĠĬĬĨĨĻÑ
    XçĿĞĬd ÑçĿĞĬ ł ĿĞĬĨĤĞĻĨIJĮçĤĞĻĞ

    ƒ PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹĤ ĴĞĽĽ
    IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹĤâXçĿĞĬã

    ƒ CĞĬĢŁĬĞĻĦ ĻÎĦ ĬIJĽĽ
    ĬIJĽĽ ł ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ÑçĿĞĬã

    ƒ GĦĻ ĴĹĦĤĬĢĻĨIJĮĽ ĞĮĤ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
    ĴĹĦĤĬĢĻĨIJĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĿĞĻĨIJĮgĴĹĦĤĬĢĻĨIJĮĽâ
                        IJŁĻĴŁĻã
    ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦĤĬĢĻĨIJĮĽd ÑçĿĞĬã

    ƒ PĹĨĮĻ Ğ ĽŁİĬĞĹÑ
    ĴĹĨĮĻâÌďĿĞĬĨĤĞĻĨIJĮd ď κ
          ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
          ÌďĬIJĽĽf ăĬIJĽĽfgġÌåďã
```

Now we can create the test data and test the binary logistic regression model with the following code:

```
ƒ CĹĦĞĻĦ ĻĹĞĨĮ ĞĮĤ ĻĦĽĻ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝƒƒd ĢĬĞĽĽĦĽłğã
XçĻĦĽĻd ÑçĻĦĽĻ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝƒƒd ĢĬĞĽĽĦĽłğã

ƒ RĦĽĤĞĴĦ ĬĞĠĦĬĽ ĻIJ ĠĦ Ğ ĬĨĽĻ IJÌ ĬĨĽĻĽ
ƒ IĮĮĦĹ ĬĨĽĻ ĢIJĮĻĞĨĮĽ IJĮĦ IJŁĻĴŁĻ âĦĨĻĤĦĹ ƒ IJĹ ĝã
ƒ ĴĦĹ ĦĞĢĤ IJŁĻĴŁĻ ĮĦŁĹIJĮd ĝ ĨĮ ĻĤĨĽ ĢĞĽĦ
Ñ ł ÑgĹĦĽĤĞĴĦâhĝd ĝã
ÑçĻĦĽĻ ł ÑçĻĦĽĻgĹĦĽĤĞĴĦâhĝd ĝã

ƒ IĮĽĻĞĮĻĨĞĻĦ ĻÎĦ ĬIJĤĦĬ
ĬIJĤĦĬ ł MIJĤĦĬâã

ƒ AĤĤ ĬĞÑĦĹĽ
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâğd ħǵd ŃĦĬĹĤĻçĹĦĹŁĬĞĹĨŅĦĹçĬğłĥĦhǵd
                              ĠĨĞĽçĹĦĹŁĬĞĹĨŅĦĹçĬğłĥĦhǵãã
ĬIJĤĦĬgĞĤĤâAĢĻĨĿĞĻĨIJĮçRĦLUâãã
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâħǵd ĝãã
ĬIJĤĦĬgĞĤĤâAĢĻĨĿĞĻĨIJĮçSĬĹĬIJĨĤâãã
```

```
# SHL ĬIJĹĽd IJĴĻĨĬĨŅHĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬHĢĻĽ
ĬIJĤHĬgĽHĻâ
    ĬIJĹĽłLIJĹĽçBĨĮĞĹÑCĹIJĹĽHĮĻĹIJĴÑâãd
    IJĴĻĨĬĨŅHĹłOĴĻĨĬĨŅHĹçAĤĞĬâĤHĢĞÑłĥHhìãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçCĞĻHÍIJĹĨĢĞĬâã
ã

# FĨĮĞĬĨŅH ĻĨH ĬIJĤHĬ
ĬIJĤHĬgÌĨĮĞĬĨŅHâã

# TĹĞĨĮ ĻĨH ĬIJĤHĬ
ĬIJĤHĬgĻĹĞĨĮâXd Ñd ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞłâXçĻHĽĻd ÑçĻHĽĻãd
            HĴIJĢĬĽłĝflflflfld ĴĹĨĮĻçHŁHĹÑłĝflflã


ńńń
HĴIJĢĬf ĝflfld ĞĢĢf flgħğĥd ĬIJĹĽf flgħìĥ âĤĞĻĞçĬIJĹĽf flgħìġd ĹHÍçĬIJĹĽf flgflflĝãd ĬĹf
flgflflflîîîîĥflĥflğġĥflĝğíì
HĴIJĢĬf ğflfld ĞĢĢf flgħġfld ĬIJĹĽf flgħħî âĤĞĻĞçĬIJĹĽf flgħħíd ĹHÍçĬIJĹĽf flgflflĝãd ĬĹf
flgflflflîîîîflflĥflîííîğħĥĝ
ggg
HĴIJĢĬf îîflfld ĞĢĢf flgîflĥd ĬIJĹĽf flgġĝğ âĤĞĻĞçĬIJĹĽf flgğìħd ĹHÍçĬIJĹĽf flgflġìãd
ĬĹf flgflflflîîĥflìġíìħíîħìîîġ
HĴIJĢĬf ĝflflflfld ĞĢĢf flgîflĥd ĬIJĹĽf flgġĝğ âĤĞĻĞçĬIJĹĽf flgğìĥd ĹHÍçĬIJĹĽf flgflġħãd
ĬĹf flgflflflîîĥflğĥġìflħĥîġííĥ
ŁĞĬĨĤĞĻĨIJĮd ĞĢĢf flgììĥd ĬIJĹĽf flgǵğġ
```

Now that we're streamlining the forward and backward pass code, including validation, this is a good time to reintroduce dropout. Recall that dropout is a method to disable, or filter out, certain neurons in an attempt to regularize and improve our model's ability to generalize. If dropout is employed in our model, we want to make sure to leave it out when performing validation and inference (predictions); in our previous code, we left it out by not calling its `ÌIJĹŃĞĹĤ` method during the forward pass during validation. Here we have a common method for performing a forward pass for both training and validation, so we need a different approach for turning off dropout — to inform the layers if we are during the training and let them "decide" on calculation to include. The first thing we'll do is include a *`ĻĹĞĨĮĨĮĺ`* boolean argument for the `ÌIJĹŃĞĹĤ` method in all the layer and activation classes, since we are calling them in a unified form:

```
    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴłĻĽd ĻĹĞĨĮĨĮĺãf
```

When we're not training, we can set the output to the input directly in the LǦÑĦĹçDĹIJĴIJŁĻ class and return from the method without changing outputs:

```
        # IÌ ĮIJĻ ĨĮ ĻÎĦ ĻĹǦĨĮĨĮÍ ÌIJÂĦ h ĹĦĻŁĹĮ ŁǦĬŁĦĹ
        ĨÌ ĮIJĻ ĻĹǦĨĮĨĮÍf
            ĹĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĹgǦIJĴÑâã
            ĹĦĻŁĹĮ
```

When we are training, we will engage the dropout:

```
# DĹIJĴIJŁĻ
ǦĬǦĹĹ LǦÑĦĹçDĹIJĴIJŁĻf
    ggg
    # FIJĹŃǦĹÂ ĴǦĹĹ
    ÂĦÌ ÌIJĹŃǦĹÂâĹĦĬÌd ĨĮĴŁĻĹd ĻĹǦĨĮĨĮÍãf
        # SǦŁĦ ĨĮĴŁĻ ŁǦĬŁĦĹ
        ĹĦĬÌgĨĮĴŁĻ ł ĨĮĴŁĻĹ

        # IÌ ĮIJĻ ĨĮ ĻÎĦ ĻĹǦĨĮĨĮÍ ÌIJÂĦ h ĹĦĻŁĹĮ ŁǦĬŁĦĹ
        ĨÌ ĮIJĻ ĻĹǦĨĮĨĮÍf
            ĹĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĹgǦIJĴÑâã
            ĹĦĻŁĹĮ

        # GĦĮĦĹǦĻĦ ǦĮÂ ĹǦŁĦ ĹǦǦĬĦÂ ÌǦĹĪ
        ĹĦĬÌgĠĨĮǦĹÑçÌǦĹĪ ł ĮĴgĹǦĮÂIJÌgĠĨĮIJÌĨǦĬâĝd ĹĦĬÌgĹǦĻĦd
                            ĹĨŅĦłĨĮĴŁĻĹgĹÎǦĴĦã w ĹĦĬÌgĹǦĻĦ
        # AĴĴĬÑ ÌǦĹĪ ĻIJ IJŁĻĴŁĻ ŁǦĬŁĦĹ
        ĹĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĹ ǻ ĹĦĬÌgĠĨĮǦĹÑçÌǦĹĪ
```

Next, we modify the ÌIJĹŃǦĹÂ method of our MIJÂĦĬ class to add the ĻĹǦĨĮĨĮÍ parameter and a call to the ÌIJĹŃǦĹÂ methods of the layers to take this parameter's value:

```
# MIJÂĦĬ ǦĬǦĹĹ
ǦĬǦĹĹ MIJÂĦĬf
    ggg
    # PĦĹÌIJĹÌĹ ÌIJĹŃǦĹÂ ĴǦĹĹ
    ÂĦÌ ÌIJĹŃǦĹÂâĹĦĬÌd Xd ĻĹǦĨĮĨĮÍãf

        # CǦĬĬ ÌIJĹŃǦĹÂ ÌĦĻÎIJÂ IJĮ ĻÎĦ ĨĮĴŁĻ ĬǦÑĦĹ
        # ĻÎĨĹ ŃĨĬĬ ĹĦĻ ĻÎĦ IJŁĻĴŁĻ ĴĹIJĴĦĹĻÑ ĻÎǦĻ
        # ĻÎĦ ÌĨĹĹĻ ĬǦÑĦĹ ĨĮ ðĴĹĦŁð IJǦÏĦǦĻ ĨĹ ĦŃĴĦǦĻĨĮÍ
        ĹĦĬÌgĨĮĴŁĻçĬǦÑĦĹgÌIJĹŃǦĹÂXd ĻĹǦĨĮĨĮÍã

        # CǦĬĬ ÌIJĹŃǦĹÂ ÌĦĻÎIJÂ IJÌ ĦŁĦĹÑ IJǦÏĦǦĻ ĨĮ Ǧ ǦÎǦĨĮ
        # PǦĹĹ IJŁĻĴŁĻ IJÌ ĻÎĦ ĴĹĦŁĨIJŁĹ IJǦÏĦǦĻ ǦĹ Ǧ ĴǦĹǦÌĦĻĦĹ
        ÌIJĹ ĬǦÑĦĹ ĨĮ ĹĦĬÌgĬǦÑĦĹĹf
            ĬǦÑĦĹgÌIJĹŃǦĹÂâĬǦÑĦĹgĴĹĦŁgIJŁĻĴŁĻd ĻĹǦĨĮĨĮÍã
```

```
        # ðĬĞÑĦĹð ĨĽ ĮIJŃ ĻÎĦ ĬĞĽĻ IJĠÏĦĢĻ ÌĹIJİ ĻÎĦ ĬĨĽĻd
        # ĹĦĻŁĹĮ ĨĻĽ IJŁĻĴŁĻ
        ĹĦĻŁĹĮ ĬĞÑĦĹgIJŁĻĴŁĻ
```

We also need to update the ĻĹĞĨĮ method in the MIJÂĦĬ class since the *ĻĹĞĨĮĨĮĺ* parameter in the forward method call will need to be set to TĹŁĦ:

```
            # PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÂ ĴĞĽĽ
            IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÂâXd ĻĹĞĨĮĨĮĺłTĹŁĦã
```

Then set to FĞĬĽĦ during validation:

```
            # PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÂ ĴĞĽĽ
            IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÂâXçĿĞĬd ĻĹĞĨĮĨĮĺłFĞĬĽĦã
```

Making the full ĻĹĞĨĮ method in the MIJÂĦĬ class:

```
    # MIJÂĦĬ ĢĬĞĽĽ
    ĢĬĞĽĽ MIJÂĦĬf
        ggg
        # TĹĞĨĮ ĻÎĦ ÌIJÂĦĬ
        ÂĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ấd ĦĴIJĢÎĽłĝd ĴĹĨĮĻçĦĿĦĹÑłĝd
                  ĿĞĬĨÂĞĻĨIJĮçÂĞĻĞłNIJĮĦãf

            # IĮĨĻĨĞĬĨŅĦ ĞĢĢŁĹĞĢÑ IJĠÏĦĢĻ
            ĽĦĬÌgĞĢĢŁĹĞĢÑgĨĮĨĻâÑã

            # MĞĨĮ ĻĹĞĨĮĨĮĺ ĬIJIJĴ
            ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮĺĦâĝd ĦĴIJĢÎĽĸĝãf

                # PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÂ ĴĞĽĽ
                IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÂâXd ĻĹĞĨĮĨĮĺłTĹŁĦã

                # CĞĬĢŁĬĞĻĦ ĬIJĽĽ
                ÂĞĻĞçĬIJĽĽd ĹĦĺŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ ł ċ
                    ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd Ñd
                                        ĨĮĢĬŁÂĦçĹĦĺŁĬĞĹĨŅĞĻĨIJĮłTĹŁĦã
                ĬIJĽĽ ł ÂĞĻĞçĬIJĽĽ ĸ ĹĦĺŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ

                # GĦĻ ĴĹĦÂĨĢĻĨIJĮĽ ĞĮÂ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
                ĴĹĦÂĨĢĻĨIJĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĿĞĻĨIJĮgĴĹĦÂĨĢĻĨIJĮĽâ
                                IJŁĻĴŁĻã
                ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦÂĨĢĻĨIJĮĽd Ñã

                # PĦĹÌIJĹİ ĠĞĢĪŃĞĹÂ ĴĞĽĽ
                ĽĦĬÌgĠĞĢĪŃĞĹÂâIJŁĻĴŁĻd Ñã
```

```
        # OJĻĨIĨŅH âŁJĤĞĻH JĞĹĞIHĻHĹĽã
        ĽHĬIgIJJĻĨIĨŅHĹgJĹHçŁJĤĞĻHçJĞĹĞIĽâã
        ÌIJĹ ĬĞÑHĹ ĨĮ ĽHĬIgĻĹĞĨĮĞĠĬHçĬĞÑHĹĽf
            ĽHĬIgIJJĻĨIĨŅHĹgŁJĤĞĻHçJĞĹĞIĽâĬĞÑHĹã
        ĽHĬIgIJJĻĨIĨŅHĹgJIJĽĻçŁJĤĞĻHçJĞĹĞIĽâã

        # PĹĨĮĻ Ğ ĽŁIIĞĹÑ
        ĨÌ ĮIJĻ HJIJĢÎ ķ JĹĨĮĻçHŁHĹÑf
            JĹĨĮĻâÌďHJIJĢÎf ăHJIJĢÎåd ď κ
                  ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
                  ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď κ
                  ÌďĤĞĻĞçĬIJĽĽf ăĤĞĻĞçĬIJĽĽfgġÌåd ď κ
                  ÌďĹHÍçĬIJĽĽf ăĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽfgġÌåãd ď κ
                  ÌďĬĹf ăĽHĬIgIJJĻĨIĨŅHĹgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻHåďã

    # IÌ ĻÎHĹH ĨĽ ĻÎH ŁĞĬĨĤĞĻĨIJĮ ĤĞĻĞ
    ĨÌ ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞ ĨĽ ĮIJĻ NIJĮHf

        # FIJĹ ĠHĻĻHĹ ĹHĞĤĞĠĨĬĨĻÑ
        XçŁĞĬd ÑçŁĞĬ ł ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞ

        # PHĹÌIJĹİ ĻÎH ÌIJĹŃĞĹĤ JĞĽĽ
        IJŁĻJŁĻ ł ĽHĬÌgÌIJĹŃĞĹĤâXçŁĞĬd ĻĹĞĨĮĨĮÍłFĞĬĽHã

        # CĞĬĢŁĬĞĻH ĻÎH ĬIJĽĽ
        ĬIJĽĽ ł ĽHĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻHâIJŁĻJŁĻd ÑçŁĞĬã

        # GHĻ JĹHĤĨĢĻĨIJĮĽ ĞĮĤ ĢĞĬĢŁĬĞĻH ĞĮ ĞĢĢŁĹĞĢÑ
        JĹHĤĨĢĻĨIJĮĽ ł ĽHĬÌgIJŁĻJŁĻçĬĞÑHĹçĞĢĻĨŁĞĻĨIJĮgJĹHĤĨĢĻĨIJĮĽâ
                          IJŁĻJŁĻã
        ĞĢĢŁĹĞĢÑ ł ĽHĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻHâJĹHĤĨĢĻĨIJĮĽd ÑçŁĞĬã

        # PĹĨĮĻ Ğ ĽŁIIĞĹÑ
        JĹĨĮĻâÌďŁĞĬĨĤĞĻĨIJĮd ď κ
              ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĽĽf ăĬIJĽĽfgġÌåďã
```

The last thing that we have to take care of, with the `MIJĤHĬ` class, is the combined Softmax activation and CrossEntropy loss class that we created for faster gradient calculation. The challenge here is that previously we have been defining forward and backward passes by hand for every model separately. Now, however, we have loops over layers in both directions of calculations, a unified way of calculating outputs and gradients, and other improvements. We cannot just simply remove the Softmax activation and CrossEntropy loss and replace them with an object combining both. It won't work with the code that we have so far, since we are handling the output activation function and loss in a specific way. Since the combined object contains just a backward pass optimization, let's leave the forward pass as is, using separate Softmax activation and Categorical Cross-Entropy loss objects, and handle just for the backward pass.

To start, we want to automatically decide if the current model is a classifier and if it uses the

Softmax activation and Categorical Cross-Entropy loss. This can be achieved by checking the class name of the last layer's object, which is an activation function's object, and by checking the class name of the loss function's object. We'll add this check to the end of the ÌĨĮĞĬĨŅĦ method:

```
# IÌ IJŁĻĴŁĻ ĞĢĻĨĿĞĻĨIJĮ ĨĽ SIJÌĻİĞŇ ĞĮĤ
# ĬIJĹĹ ÌŁĮĢĻĨIJĮ ĨĽ CĞĻĦÍIJĹĨĢĞĬ CĹIJĹĹhEĮĻĹIJĴÑ
# ĢĹĦĞĻĦ ĞĮ IJĠÏĦĢĻ IJÌ ĢIJİĠĨĮĦĤ ĞĢĻĨĿĞĻĨIJĮ
# ĞĮĤ ĬIJĹĹ ÌŁĮĢĻĨIJĮ ĢIJĮĻĞĨĮĨĮÍ
# ÌĞĹĻĦĹ ÍĹĞĤĨĦĮĻ ĢĞĬĢŁĬĞĻĨIJĮ
ĨÌ ĨĹĨĮĹĻĞĮĢĦâĹĦĬÌgĬĞŇĦĹĹähĝād AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇã ĞĮĤ ċ
   ĨĹĨĮĹĻĞĮĢĦâĹĦĬÌgĬIJĹĹd LIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑãf
    # CĹĦĞĻĦ ĞĮ IJĠÏĦĢĻ IJÌ ĢIJİĠĨĮĦĤ ĞĢĻĨĿĞĻĨIJĮ
    # ĞĮĤ ĬIJĹĹ ÌŁĮĢĻĨIJĮĹ
    ĹĦĬÌgĹIJÌĻİĞŇçĢĬĞĹĹĨÌÌĨĦĹçIJŁĻĴŁĻ ł ċ
        AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇçLIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑâã
```

To make this check, we are using Python's ĨĹĨĮĹĻĞĮĢĦ function, which returns *True* if a given object is an instance of a given class. If both of the tests return *True*, we are setting a new property containing an object of the AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇçLIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑ class.
We also want to initialize this property with a value of NIJĮĦ in the MIJĤĦĬ class' constructor:

```
# SIJÌĻİĞŇ ĢĬĞĹĹĨÌÌĨĦĹďĹ IJŁĻĴŁĻ IJĠÏĦĢĻ
ĹĦĬÌgĹIJÌĻİĞŇçĢĬĞĹĹĨÌÌĨĦĹçIJŁĻĴŁĻ ł NIJĮĦ
```

The last step is, during the backward pass, to check if this object is set and, if it is, to use it. To do so, we need to handle this case separately with a slightly modified version of the current code of the backward pass. First, we call the ĠĞĢĪŃĞĹĤ method of the combined object, then, since we won't call the ĠĞĢĪŃĞĹĤ method of the activation function (the last object on a list of layers), set the ĤĨĮĴŁĻĹ of the object of this function with the gradient calculated within the activation/loss object. At the end, we can iterate all of the layers except for the last one and perform the backward pass on them:

```
# IÌ ĹIJÌĻİĞŇ ĢĬĞĹĹĨÌÌĨĦĹ
ĨÌ ĹĦĬÌgĹIJÌĻİĞŇçĢĬĞĹĹĨÌÌĨĦĹçIJŁĻĴŁĻ ĨĹ ĮIJĻ NIJĮĦf
    # FĨĹĹĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻĨIJĤ
    # IJĮ ĻĨĦ ĢIJİĠĨĮĦĤ ĞĢĻĨĿĞĻĨIJĮwĬIJĹĹ
    # ĻĨĨĹ ŃĨĬĬ ĹĦĻ ĤĨĮĴŁĻĹ ĴĹIJĴĦĹĻŃ
    ĹĦĬÌgĹIJÌĻİĞŇçĢĬĞĹĹĨÌÌĨĦĹçIJŁĻĴŁĻgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd Ñã

    # SĨĮĢĦ ŃĦďĬĬ ĮIJĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻĨIJĤ IJÌ ĻĨĦ ĬĞĹĻ ĬĞŇĦĹ
    # ŃĨĨĢĨ ĨĹ SIJÌĻİĞŇ ĞĢĻĨĿĞĻĨIJĮ
    # ĞĹ ŃĦ ŁĹĦĤ ĢIJİĠĨĮĦĤ ĞĢĻĨĿĞĻĨIJĮwĬIJĹĹ
    # IJĠÏĦĢĻd ĬĦĻďĹ ĹĦĻ ĤĨĮĴŁĻĹ ĨĮ ĻĨĨĹ IJĠÏĦĢĻ
```

```
            ĽĦĬÌgĬĞÑĦĹĽähĝāgĤĨĮĴŁĻĽ ł ċ
                ĽĦĬÌgĽIJÌĻİĞŇçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻĴŁĻgĤĨĮĴŁĻĽ
            ſı CĞĬĬ ĠĞĢĬŃĞĹĤ İĦĻĨIJĤ ÍIJĨĮÍ ĻĨĹIJŁÍĨĨ
            ſı ĞĬĬ ĻĨĦ IJĠÏĦĢĻĽ ĠŁĻ ĬĞĽĻ
            ſı ĨĮ ĹĦŁĦĹĽĦĤ IJĹĤĦĹ ĴĞĽĽĨĮÍ ĤĨĮĴŁĻĽ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
            ÌIJĹ ĬĞÑĦĹ ĨĮ ĹĦŁĦĹĽĦĤâĽĦĬÌgĬĞÑĦĹĽäfhĝāãf
                ĬĞÑĦĹgĠĞĢĬŃĞĹĤâĬĞÑĦĹgĮĦŇĻgĤĨĮĴŁĻĽã

            ĹĦĻŁĹĮ
```

Full MIJĤĦĬ class up to this point:

```
ſı MIJĤĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJĤĦĬf

    ĤĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ſı CĹĦĞĻĦ Ğ ĬĨĽĻ IJÌ ĮĦĻŃIJĹĨ IJĠÏĦĢĻĽ
        ĽĦĬÌgĬĞÑĦĹĽ ł äā
        ſı SIJÌĻİĞŇ ĢĬĞĽĽĨÌÌĨĦĹďĽ IJŁĻĴŁĻ IJĠÏĦĢĻ
        ĽĦĬÌgĽIJÌĻİĞŇçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻĴŁĻ ł NIJĮĦ

    ſı AĤĤ IJĠÏĦĢĻĽ ĻIJ ĻĨĦ İIJĤĦĬ
    ĤĦÌ ĞĤĤâĽĦĬÌd ĬĞÑĦĹãf
        ĽĦĬÌgĬĞÑĦĹĽgĞĴĴĦĮĤâĬĞÑĦĹã

    ſı SĦĻ ĬIJĽĽd IJĴĻĨÌĨÌÑĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ
    ĤĦÌ ĽĦĻâĽĦĬÌd ǻd ĬIJĽĽd IJĴĻĨÌĨÌÑĦĹd ĞĢĢŁĹĞĢÑãf
        ĽĦĬÌgĬIJĽĽ ł ĬIJĽĽ
        ĽĦĬÌgIJĴĻĨÌĨÌÑĦĹ ł IJĴĻĨÌĨÌÑĦĹ
        ĽĦĬÌgĞĢĢŁĹĞĢÑ ł ĞĢĢŁĹĞĢÑ

    ſı FĨĮĞĬĨÌÑĦ ĻĨĦ İIJĤĦĬ
    ĤĦÌ ÌĨĮĞĬĨÌÑĦâĽĦĬÌãf

        ſı CĹĦĞĻĦ ĞĮĤ ĽĦĻ ĻĨĦ ĨĮĴŁĻ ĬĞÑĦĹ
        ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ ł LĞÑĦĹçIĮĴŁĻâã

        ſı CIJŁĮĻ ĞĬĬ ĻĨĦ IJĠÏĦĢĻĽ
        ĬĞÑĦĹçĢIJŁĮĻ ł ĬĦĮâĽĦĬÌgĬĞÑĦĹĽã

        ſı IĮĨĻĨĞĬĨÌÑĦ Ğ ĬĨĽĻ ĢIJĮĻĞĨĮĨĮÍ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽf
        ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ ł äā

        ſı IĻĦĹĞĻĦ ĻĨĦ IJĠÏĦĢĻĽ
        ÌIJĹ Ĩ ĨĮ ĹĞĮÍĦâĬĞÑĦĹçĢIJŁĮĻãf

            ſı IÌ ĨĻďĽ ĻĨĦ ÌĨĹĽĻ ĬĞÑĦĹd
            ſı ĻĨĦ ĴĹĦĻĨIJŁĽ ĬĞÑĦĹ IJĠÏĦĢĻ ĨĽ ĻĨĦ ĨĮĴŁĻ ĬĞÑĦĹ
            ĨÌ Ĩ łł flf
```

```
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦĻ ł ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦŃĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝā
        fi AĬĬ ĬĞÑĦĹĽ ĦŃĢĦĴĻ ÌIJĹ ĻĨĦ ÌĨĹĽĻ ĞĮĤ ĻĨĦ ĬĞĽĻ
        ĦĬĨÌ Ĩ l· ĬĞÑĦĹçĢIJŁĮĻ h ĝf
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝā
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦŃĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝā

        fi TĨĦ ĬĞĽĻ ĬĞÑĦĹ h ĻĨĦ ĮĦŃĻ IJĠĬĦĢĻ ĨĽ ĻĨĦ ĬIJĽĽ
        fi AĬĽIJ ĬĦĻďĽ ĽĞŁĦ ĞĽĨĤĦ ĻĨĦ ĹĦÌĦĹĦĮĢĦ ĻIJ ĻĨĦ ĬĞĽĻ IJĠĬĦĢĻ
        fi ŃĨIJĽĦ IJŁĻĴŁĻ ĨĽ ĻĨĦ ĬIJĤĦĬďĽ IJŁĻĴŁĻ
        ĦĬĽĦf
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝā
            ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦŃĻ ł ĽĦĬÌgĬIJĽĽ
            ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨŁĞĻĨIJĮ ł ĽĦĬÌgĬĞÑĦĹĽäĨā

        fi IÌ ĬĞÑĦĹ ĢIJĮĻĞĨĮĽ ĞĮ ĞĻĻĹĨĠŁĻĦ ĢĞĬĬĦĤ ðŃĦĨĹĨĻĽðd
        fi ĨĻďĽ Ğ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹ h
        fi ĞĤĤ ĨĻ ĻIJ ĻĨĦ ĬĨĽĻ IJÌ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
        fi WĦ ĤIJĮďĻ ĮĦĦĤ ĻIJ ĢĨĦĢĨ ÌIJĹ ĠĨĞĽĦĹ h
        fi ĢĨĦĢĨĨĮĹ ÌIJĹ ŃĦĨĹĨĻĽ ĨĽ ĦĮIJŁĹĨ
        ĨÌ ĨĞĽĞĻĻĹâĽĦĬÌgĬĞÑĦĹĽäĨād dŃĦĨĹĨĻĽďãf
            ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽgĞĴĴĦĮĤâĽĦĬÌgĬĞÑĦĹĽäĨāã

        fi UĴĤĞĻĦ ĬIJĽĽ IJĠĬĦĢĻ ŃĨĻĨ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
        ĽĦĬÌgĬIJĽĽgĹĦĬĦĬĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽâ
            ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ
        ã

    fi IÌ IJŁĻĴŁĻ ĞĢĻĨŁĞĻĨIJĮ ĨĽ SIJÌĻĬĞÑ ĞĮĤ
    fi ĬIJĽĽ ÌŁĮĢĻĨIJĮ ĨĽ CĞĻĦĬIJĹĨĢĞĬ CĹIJĽĽhEĮĻĹIJĴÑ
    fi ĢĹĦĞĻĦ ĞĮ IJĠĬĦĢĻ IJÌ ĢIJĬĠĨĮĦĤ ĞĢĻĨŁĞĻĨIJĮ
    fi ĞĮĤ ĬIJĽĽ ÌŁĮĢĻĨIJĮ ĢIJĮĻĞĨĮĨĮĬ
    fi ÌĞĽĻĦĹ ĬĹĞĤĨĦĮĻ ĢĞĬĢŁĬĞĻĨIJĮ
    ĨÌ ĨĽĨĮĽĻĞĮĢĦâĽĦĬÌgĬĞÑĦĹĽähĝād AĢĻĨŁĞĻĨIJĮçSIJÌĻĬĞÑã ĞĮĤ ċ
       ĨĽĨĮĽĻĞĮĢĦâĽĦĬÌgĬIJĽĽd LIJĽĽçCĞĻĦĬIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑãf
        fi CĹĦĞĻĦ ĞĮ IJĠĬĦĢĻ IJÌ ĢIJĬĠĨĮĦĤ ĞĢĻĨŁĞĻĨIJĮ
        fi ĞĮĤ ĬIJĽĽ ÌŁĮĢĻĨIJĮĽ
        ĽĦĬÌgĽIJÌĻĬĞÑçĢĬĞĽĽĨÌĨĦĹçIJŁĻĴŁĻ ł ċ
            AĢĻĨŁĞĻĨIJĮçSIJÌĻĬĞÑçLIJĽĽçCĞĻĦĬIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâã

fi TĹĞĨĮ ĻĨĦ ĬIJĤĦĬ
ĤĦÌ ĻĹĞĨĮâĽĦĬÌd Xd Ñd ấd ĦĴIJĢĨĽłĝd ĴĹĨĮçĦŁĦĹÑłĝd
          ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞłNIJĮĦãf

    fi IĮĨĻĨĞĬĨŅĦ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻ
    ĽĦĬÌgĞĢĢŁĹĞĢÑgĨĮĨĻâÑã

    fi MĞĨĮ ĻĹĞĨĮĨĮĬ ĬIJIJĴ
    ÌIJĹ ĦĴIJĢĨ ĨĮ ĹĞĮĬĦâĝd ĦĴIJĢĨĽкĝãf
```

```
    # PHĹÌIJĹİ ĻÎH ÌIJĹŃĞĹÂ ĴĞĽĽ
    IJŁĻĴŁĻ ł ĽHĬÌgÌIJĹŃĞĹÂâXd ĻĹĞĨĮĨĮÍłTĹŁHã

    # CĞĬĢŁĬĞĻH ĬIJĽĽ
    ÂĞĻĞçĬIJĽĽd ĹHÍŁĬĞĹĨŅĞĻĨIJIçĬIJĽĽ ł ċ
        ĽHĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻHâIJŁĻĴŁĻd Ñd
                                ĨĮĢĬŁÂHçĹHÍŁĬĞĹĨŅĞĻĨIJIłTĹŁHã
    ĬIJĽĽ ł ÂĞĻĞçĬIJĽĽ κ ĹHÍŁĬĞĹĨŅĞĻĨIJIçĬIJĽĽ

    # GHĻ ĴĹHÂĨĢĻĨIJIĽ ĞIÂ ĢĞĬĢŁĬĞĻH ĞI ĞĢĢŁĹĞĢÑ
    ĴĹHÂĨĢĻĨIJIĽ ł ĽHĬÌgIJŁĻĴŁĻçĬĞÑHĹçĞĢĻĨŁĞĻĨIJIgĴĹHÂĨĢĻĨIJIĽâ
                        IJŁĻĴŁĻã
    ĞĢĢŁĹĞĢÑ ł ĽHĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻHâĴĹHÂĨĢĻĨIJIĽd Ñã

    # PHĹÌIJĹİ ĠĞĢĪŃĞĹÂ ĴĞĽĽ
    ĽHĬÌgĠĞĢĪŃĞĹÂâIJŁĻĴŁĻd Ñã

    # OĴĻĨİĨŅH âŁĴÂĞĻH ĴĞĹĞİHĻHĹĽã
    ĽHĬÌgIJĴĻĨİĨŅHĹgĴĹHçŁĴÂĞĻHçĴĞĹĞİĽâã
    ÌIJĹ ĬĞÑHĹ ĨĮ ĽHĬÌgĻĹĞĨĮĞĠĬHçĬĞÑHĹĽf
        ĽHĬÌgIJĴĻĨİĨŅHĹgŁĴÂĞĻHçĴĞĹĞİĽâĬĞÑHĹã
    ĽHĬÌgIJĴĻĨİĨŅHĹgĴIJĽĻçŁĴÂĞĻHçĴĞĹĞİĽâã

    # PĹĨĮĻ Ğ ĽŁİĬĞĹÑ
    ĨÌ ĮIJĻ HĴIJĢÎ ķ ĴĹĨĮĻçHŁHĹÑf
        ĴĹĨĮĻâÌďHĴIJĢÎf ăHĴIJĢÎåd ď κ
              ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď κ
              ÌďÂĞĻĞçĬIJĽĽf ăÂĞĻĞçĬIJĽĽfgġÌåd ď κ
              ÌďĹHÍçĬIJĽĽf ăĹHÍŁĬĞĹĨŅĞĻĨIJIçĬIJĽĽfgġÌåãd ď κ
              ÌďĬĹf ăĽHĬÌgIJĴĻĨİĨŅHĹgĢŁĹĹHĮĻçĬHĞĹĮĨĮÍçĹĞĻHåďã

# IÌ ĻÎHĹH ĨĽ ĻÎH ŁĞĬĨÂĞĻĨIJĮ ÂĞĻĞ
ĨÌ ŁĞĬĨÂĞĻĨIJIçÂĞĻĞ ĨĽ ĮIJĻ NIJĮH f

    # FIJĹ ĠHĻĻHĹ ĹHĞÂĞĠĨĬĨĻÑ
    XçŁĞĬd ÑçŁĞĬ ł ŁĞĬĨÂĞĻĨIJIçÂĞĻĞ

    # PHĹÌIJĹİ ĻÎH ÌIJĹŃĞĹÂ ĴĞĽĽ
    IJŁĻĴŁĻ ł ĽHĬÌgÌIJĹŃĞĹÂâXçŁĞĬd ĻĹĞĨĮĨĮÍłFĞĬĽHã

    # CĞĬĢŁĬĞĻH ĻÎH ĬIJĽĽ
    ĬIJĽĽ ł ĽHĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻHâIJŁĻĴŁĻd ÑçŁĞĬã

    # GHĻ ĴĹHÂĨĢĻĨIJIĽ ĞIÂ ĢĞĬĢŁĬĞĻH ĞI ĞĢĢŁĹĞĢÑ
    ĴĹHÂĨĢĻĨIJIĽ ł ĽHĬÌgIJŁĻĴŁĻçĬĞÑHĹçĞĢĻĨŁĞĻĨIJIgĴĹHÂĨĢĻĨIJIĽâ
                        IJŁĻĴŁĻã
    ĞĢĢŁĹĞĢÑ ł ĽHĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻHâĴĹHÂĨĢĻĨIJIĽd ÑçŁĞĬã
```

```
        # PĹĨĮĻ Ğ ĽŁİİĞĹÑ
        ĴĹĨĮĻâÌďĽĞĬĨĤĞĻĨIJĮd ď κ
              ÌďĞ̣Ğ̣f ăĞ̣Ğ̣ŁĹĞ̣Ñfgġìåd ď κ
              Ìďĭ̇IJĽĽf ăĬIJĽĽfgġìåďã

# PĦĹÌIJĹİĽ ÌIJĹŃĞĹĤ ĴĞĽĽ
ÎĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd Xd Ļ̣ĹĞĨĮĨĮÍãf

    # CĞĬĬ ÌIJĹŃĞĹĤ İĦĻÎIJĤ IJĮ Ļ̣ÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
    # Ļ̣ÎĨĽ ŃĨĬĬ ĽĦĻ Ļ̣ÎĦ IJŁĻ̣ĴŁĻ ĴĹIJĴĦĹĻÑ Ļ̣ÎĞĻ
    # Ļ̣ÎĦ ÌĨĹĽĻ ĬĞÑĦĹ ĨĮ ðĴĹĦĻð IJĠÏĦĢĻ ĨĽ ĦŃĴĦĢĻĨĮÍ
    ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹgÌIJĹŃĞĹĤâXd Ļ̣ĹĞĨĮĨĮÍã

    # CĞĬĬ ÌIJĹŃĞĹĤ İĦĻÎIJĤ IJÌ ĦĻĦĹÑ IJĠÏĦĢĻ ĨĮ Ğ ĢÎĞĨĮ
    # PĞĽĽ IJŁĻ̣ĴŁĻ IJÌ Ļ̣ÎĦ ĴĹĦĻĨIJŁĽ IJĠÏĦĢĻ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
    ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĬĞÑĦĹĽf
        ĬĞÑĦĹgÌIJĹŃĞĹĤâĬĞÑĦĹgĴĹĦĻgIJŁĻ̣ĴŁĻd Ļ̣ĹĞĨĮĨĮÍã

    # ðĬĞÑĦĹð ĨĽ ĮIJŃ Ļ̣ÎĦ ĬĞĽĻ IJĠÏĦĢĻ ÌĹIJİ Ļ̣ÎĦ ĬĨĽĻd
    # ĹĦĻŁĹĮ ĨĻĽ IJŁĻ̣ĴŁĻ
    ĹĦĻŁĹĮ ĬĞÑĦĹgIJŁĻ̣ĴŁĻ

# PĦĹÌIJĹİĽ ĠĞĢĪŃĞĹĤ ĴĞĽĽ
ÎĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd IJŁĻ̣ĴŁĻd Ñãf

    # IÌ ĽIJÌĻİĞÑ ĢĬĞĽĽĨÌÌĨĦĹ
    ĨÌ ĽĦĬÌgĽIJÌĻİĞÑçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻ̣ĴŁĻ ĨĽ ĮIJĻ NIJĮĦf
        # FĨĹĽĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻÎIJĤ
        # IJĮ Ļ̣ÎĦ ĢIJİĠĨĮĦĤ ĞĢĻĨĹĞĻĨIJĮwĬIJĽĽ
        # Ļ̣ÎĨĽ ŃĨĬĬ ĽĦĻ ĤĨĮĴŁĻĽ ĴĹIJĴĦĹĻÑ
        ĽĦĬÌgĽIJÌĻİĞÑçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻ̣ĴŁĻgĠĞĢĪŃĞĹĤâIJŁĻ̣ĴŁĻd Ñã

        # SĨĮĢĦ ŃĦďĬĬ ĮIJĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻÎIJĤ IJÌ Ļ̣ÎĦ ĬĞĽĻ ĬĞÑĦĹ
        # ŃÎĨĢÎ ĨĽ SIJÌĻİĞÑ ĞĢĻĨĹĞĻĨIJĮ
        # ĞĽ ŃĦ ŁĽĦĤ ĢIJİĠĨĮĦĤ ĞĢĻĨĹĞĻĨIJĮwĬIJĽĽ
        # IJĠÏĦĢĻd ĬĦĻďĽ ĽĦĻ ĤĨĮĴŁĻĽ ĨĮ Ļ̣ÎĨĽ IJĠÏĦĢĻ
        ĽĦĬÌgĬĞÑĦĹĽähĝāgĤĨĮĴŁĻĽ ł ċ
            ĽĦĬÌgĽIJÌĻİĞÑçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻ̣ĴŁĻgĤĨĮĴŁĻĽ

        # CĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻÎIJĤ ÍIJĨĮÍ Ļ̣ÎĹIJŁÍÎ
        # ĞĬĬ Ļ̣ÎĦ IJĠÏĦĢĻĽ ĠŁĻ ĬĞĽĻ
        # ĨĮ ĹĦĻĦĹĽĦĤ IJĹĤĦĹ ĴĞĽĽĨĮÍ ĤĨĮĴŁĻĽ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
        ÌIJĹ ĬĞÑĦĹ ĨĮ ĹĦĻĦĹĽĦĤâĽĦĬÌgĬĞÑĦĹĽäfhĝāãf
            ĬĞÑĦĹgĠĞĢĪŃĞĹĤâĬĞÑĦĹgĮĦŃĻgĤĨĮĴŁĻĽã

        ĹĦĻŁĹĮ
```

```
        # FĨĹĽĻ ĢĞĬĬ ĠĞĢĪŃĞĹÎ ĮĦĻÎIJÎ IJĮ ĻÎĦ ĬIJĽĽ
        # ĻÎĨĽ ŃĨĬĬ ĽĦĻ ÎĨĮĴŁĻĽ ĴĹIJĴĦĹĻÑ ĻÎĞĻ ĻÎĦ ĬĞĽĻ
        # ĬĞÑĦĹ ŃĨĬĬ ĻĹÑ ĻIJ ĞĢĢĦĽĽ ĽÎIJĹĻĬÑ
        ĽĦĬÌgĬIJĽĽgĠĞĢĪŃĞĹÎâIJŁĻĴŁĻd Ñã

        # CĞĬĬ ĠĞĢĪŃĞĹÎ ĮĦĻÎIJÎ ÍIJĨĮÍ ĻÎĹIJŁÍÎ ĞĬĬ ĻÎĦ IJĠÏĦĢĻĽ
        # ĨĮ ĹĦL·ĦĹĽĦÎ IJĹÎĦĹ ĴĞĽĽĨĮÍ ÎĨĮĴŁĻĽ ĞĽ Ğ ĴĞĹĞĮĦĻĦĹ
        ÌIJĹ ĬĞÑĦĹ ĨĮ ĹĦL·ĦĹĽĦÎâĽĦĬÌgĬĞÑĦĹĽãf
            ĬĞÑĦĹgĠĞĢĪŃĞĹÎâĬĞÑĦĹgĮĦŃĻgÎĨĮĴŁĻĽã
```

Also we won't need the initializer and the forward method of the AĢĻĨL·ĞĻĨIJĮçSIJÌĻĮĞÑçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑ class anymore so we can remove them leaving in just the backward pass:

```
# SIJÌĻĮĞÑ ĢĬĞĽĽĨÌÌĨĦĹ h ĢIJĮĞĨĮĦÎ SIJÌĻĮĞÑ ĞĢĻĨL·ĞĻĨIJĮ
# ĞĮÎ ĢĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ ÌIJĹ ÌĞĽĻĦĹ ĠĞĢĪŃĞĹÎ ĽĻĦĴ
ĢĬĞĽĽ AĢĻĨL·ĞĻĨIJĮçSIJÌĻĮĞÑçLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâãf

    # BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ĠĞĢĪŃĞĹÎâĽĦĬÌd ÎL·ĞĬŁĦĽd ÑçĻĹŁĦãf

        # NŁĮĠĦĹ IJÌ ĽĞĮĴĬĦĽ
        ĽĞĮĴĬĦĽ Ł ĬĦĮâÎL·ĞĬŁĦĽã

        # CIJĴÑ ĽIJ ŃĦ ĢĞĮ ĽĞÌĦĬÑ ĮIJÎĨÌÑ
        ĽĦĬÌgÎĨĮĴŁĻĽ Ł ÎL·ĞĬŁĦĽgĢIJĴÑâã
        # CĞĬĢŁĬĞĻĦ ĹĹĞÎĨĦĮĻ
        ĽĦĬÌgÎĨĮĴŁĻĽäĹĞĮĹĦâĽĞĮĴĬĦĽãd ÑçĻĹŁĦã hŁ ĝ
        # NIJĹĮĞĬĨÑĦ ĹĹĞÎĨĦĮĻ
        ĽĦĬÌgÎĨĮĴŁĻĽ Ł ĽĦĬÌgÎĨĮĴŁĻĽ w ĽĞĮĴĬĦĽ
```

Now we can test our updated MIJÎĦĬ object with dropout:

```
# CĹĦĞĻĦ ÎĞĻĞĽĦĻ
Xd Ñ Ł ĽĴĨĹĞĬçÎĞĻĞâĽĞĮĴĬĦĽŁĝflflfld ĢĬĞĽĽĦĽŁġã
XçĻĦĽĻd ÑçĻĦĽĻ Ł ĽĴĨĹĞĬçÎĞĻĞâĽĞĮĴĬĦĽŁĝflfld ĢĬĞĽĽĦĽŁġã

# IĮĽĻĞĮĻĨĞĻĦ ĻÎĦ ĮIJÎĦĬ
ĮIJÎĦĬ Ł MIJÎĦĬâã
```

```
ſı AĤĤ ĬĞÑĦĹĽ
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâğd ĥĝğd ŃĦĬÍĮçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦhǵd
                                ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłĥĦhǵãã
İIJĤĦĬgĞĤĤâAĢĻĨŁĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĹIJĴIJŁĻâflgĝãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĥĝğd ġãã
İIJĤĦĬgĞĤĤâAĢĻĨŁĞĻĨIJĮçSIJÌĻİĞŇâãã

ſı SĦĻ ĬIJĽĽd IJĴĻĨİĨŅĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĽ
İIJĤĦĬgĽĦĻâ
    ĬIJĽĽłLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâãd
    IJĴĻĨİĨŅĦĹłOĴĻĨİĨŅĦĹçAĤĞİâĬĦĞĹĮĨĮÍçĹĞĻĦłflgflĥd ĤĦĢĞÑłĥĦhĥãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâã
ã

ſı FĨĮĞĬĨŅĦ ĻÎĦ İIJĤĦĬ
İIJĤĦĬgÌĨĮĞĬĨŅĦâã

ſı TĹĞĨĮ ĻÎĦ İIJĤĦĬ
İIJĤĦĬgĻĹĞĨĮâXd Ñd ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞłâXçĻĦĽĻd ÑçĻĦĽĻãd
            ĦĴIJĢÎĽłĝflflflfld ĴĹĨĮçĦŁĦĹÑłĝflflã


ńńń
ĦĴIJĢÎf ĝflfld ĞĢĢf flgìĝħd ĬIJĽĽf flgìğħ âĤĞĻĞçĬIJĽĽf flgħħħd ĹĦÍçĬIJĽĽf flgflħflãd ĬĹf
flgflǵîìĥġìĝîflîflĥflğflğ
ĦĴIJĢÎf ğflfld ĞĢĢf flgìíìd ĬIJĽĽf flgħĝĥ âĤĞĻĞçĬIJĽĽf flgĥġíd ĹĦÍçĬIJĽĽf flgflìiãd ĬĹf
flgflǵîĥflìǵflĝǵĥħĥflğíflħ
ggg
ĦĴIJĢÎf îîflfld ĞĢĢf flgíħĝd ĬIJĽĽf flgǵǵħ âĤĞĻĞçĬIJĽĽf flgǵíîd ĹĦÍçĬIJĽĽf flgflǵħãd
ĬĹf flgflġǵǵǵĥîġǵħǵħħǵǵì
ĦĴIJĢÎf ĝflflflfld ĞĢĢf flgííflfld ĬIJĽĽf flgǵîǵ âĤĞĻĞçĬIJĽĽf flgǵǵìd ĹĦÍçĬIJĽĽf flgflǵìãd
ĬĹf flgflġǵǵǵǵǵǵǵǵíĝǵíğìĝ
ŁĞĬĨĤĞĻĨIJĮd ĞĢĢf flgíħìd ĬIJĽĽf flgǵìî
```

It seems like everything is working as intended. Now that we've got this MIJĤĦĬ class, we're able to define new models without writing large amounts of code repeatedly. Rewriting code is annoying and leaves more room to make small, hard-to-notice mistakes.

Full code up to this point:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ĨİĴIJĹĻ ĽĨĮĦçĤĞĻĞd ĽĴĨĹĞĬçĤĞĻĞ

ĮĮÌĽgĨĮĨĻâã


ſ DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ſ LĞÑĦĹ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽd
                ŃĦĨÍĴĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝłfld ŃĦĨÍĴĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłfld
                ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝłfld ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłflãf
        ſ IĮĨĻĨĞĬĨŅĦ ŃĦĨÍĴĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍĴĻĽ ł flgflĝ ǻ ĮĴgĹĞĮĤIJİgĹĞĮĤĮâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã
        ſ SĦĻ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĽĻĹĦĮÍĻĴ
        ĽĦĬÌgŃĦĨÍĴĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ł ŃĦĨÍĴĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ
        ĽĦĬÌgŃĦĨÍĴĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ł ŃĦĨÍĴĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ
        ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ł ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ
        ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ł ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ

    ſ FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ſ RĦİĦİĠĦĹ ĨĮĴŁĻ ĽĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ſ CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ĽĞĬŁĦĽ ÌĹIJİ ĨĮĴŁĻĽd ŃĦĨÍĴĻĽ ĞĮĤ ĠĨĞĽĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgĤIJĻâĨĮĴŁĻĽd ĽĦĬÌgŃĦĨÍĴĻĽã ĸ ĽĦĬÌgĠĨĞĽĦĽ

    ſ BĞĢĨŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤĽĞĬŁĦĽãf
        ſ GĹĞĤĨĦĮĻĽ IJĮ ĴĞĹĞİĦĻĦĹĽ
        ĽĦĬÌgĤŃĦĨÍĴĻĽ ł ĮĴgĤIJĻâĽĦĬÌgĨĮĴŁĻĽgTd ĤĽĞĬŁĦĽã
        ĽĦĬÌgĤĠĨĞĽĦĽ ł ĮĴgĽŁİâĤĽĞĬŁĦĽd ĞŃĨĽłfld ĪĦĦĴĤĨİĽłTĹŁĦã
```

```
        # GĹĞĤĨĦĮĻĽ IJĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ
        # Lĝ IJĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ĤLĝ ł ĮĴgIJĮĦĽçĬĨĨĦâĽĦĬÌgŃĦĨÍÎĻĽã
            ĤLĝäĽĦĬÌgŃĦĨÍÎĻĽ l· flā ł hĝ
            ĽĦĬÌgĤŃĦĨÍÎĻĽ κł ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ấ ĤLĝ
        # Lğ IJĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĽĦĬÌgĤŃĦĨÍÎĻĽ κł ğ ấ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ấ ċ
                                ĽĦĬÌgŃĦĨÍÎĻĽ
        # Lĝ IJĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ĤLĝ ł ĮĴgIJĮĦĽçĬĨĨĦâĽĦĬÌgĠĨĞĽĦĽã
            ĤLĝäĽĦĬÌgĠĨĞĽĦĽ l· flā ł hĝ
            ĽĦĬÌgĤĠĨĞĽĦĽ κł ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ấ ĤLĝ
        # Lğ IJĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĽĦĬÌgĤĠĨĞĽĦĽ κł ğ ấ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ấ ċ
                                ĽĦĬÌgĠĨĞĽĦĽ

        # GĹĞĤĨĦĮĻ IJĮ L·ĞĬŁĦĽ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĮĴgĤIJĻâĤL·ĞĬŁĦĽd ĽĦĬÌgŃĦĨÍÎĻĽgTã


# DĹIJĴIJŁĻ
ĢĬĞĽĽ LĞÑĦĹçDĹIJĴIJŁĻf

    # IĮĨĻ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĹĞĻĦãf
        # SĻIJĹĦ ĹĞĻĦd ŃĦ ĨĮL·ĦĹĻ ĨĻ ĞĽ ÌIJĹ ĦŃĞĬĴĬĦ ÌIJĹ ĤĹIJĴIJŁĻ
        # IJÌ flgĝ ŃĦ ĮĦĦĤ ĽŁĢĢĦĽĽ ĹĞĻĦ IJÌ flgî
        ĽĦĬÌgĹĞĻĦ ł ĝ h ĹĞĻĦ

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮĺãf
        # SĞL·Ħ ĨĮĴŁĻ L·ĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ

        # IÌ ĮIJĻ ĨĮ ĻĤĦ ĻĹĞĨĮĨĮĺ ÌIJĤĦ h ĹĦĻŁĹĮ L·ĞĬŁĦĽ
        ĨÌ ĮIJĻ ĻĹĞĨĮĨĮĺf
            ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽgĢIJĴÑâã
            ĹĦĻŁĹĮ

        # GĦĮĦĹĞĻĦ ĞĮĤ ĽĞL·Ħ ĽĢĞĬĦĤ ÌĞĽĨ
        ĽĦĬÌgĠĨĮĠĹÑçÌĞĽĨ ł ĮĴgĹĞĮĤIJÌgĠĨĮIJÌĨĞĬâĝd ĽĦĬÌgĹĞĻĦd
                            ĹĨŅĦłĨĮĴŁĻĽgĽĨĞĴĦã w ĽĦĬÌgĹĞĻĦ
        # AĴĴĬÑ ÌĞĽĨ ĻIJ IJŁĻĴŁĻ L·ĞĬŁĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽ ấ ĽĦĬÌgĠĨĮĠĹÑçÌĞĽĨ
```

```
    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ÂĽĞĬŁHĽãf
        # GĹĞĤĨHĮĻ IJĮ ĽĞĬŁHĽ
        ĽHĬÌgĤĨĮĴŁĻĽ ł ÂĽĞĬŁHĽ ǻ ĽHĬÌgĠĨĮĞĹÑçĬĞĽĨ


# IĮĴŁĻ ðĬĞÑHĹð
ĢĬĞĽĽ LĞÑHĹçIĮĴŁĻf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ĽHĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽ


# RHLU ĞĢĻĨĽĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĽĞĻĨIJĮçRHLUf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        # RHĬHĬĠHĹ ĨĮĴŁĻ ĽĞĬŁHĽ
        ĽHĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        # CĞĬĢŁĬĞĻH IJŁĻĴŁĻ ĽĞĬŁHĽ ÌĹIJĬ ĨĮĴŁĻĽ
        ĽHĬÌgIJŁĻĴŁĻ ł ĮĴgĬĞŃĨĬŁĬâfld ĨĮĴŁĻĽã

    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ÂĽĞĬŁHĽãf
        # SĨĮĢH ŃH ĮHHÂ ĻIJ ĬIJÂĨÌÑ IJĹĨÍĨĮĞĬ ĽĞĹĨĞĠĬHd
        # ĬHĻďĽ ĬĞĨH Ğ ĢIJĴÑ IJÌ ĽĞĬŁHĽ ÌĨĹĽĻ
        ĽHĬÌgÂĨĮĴŁĻĽ ł ÂĽĞĬŁHĽgĢIJĴÑâã

        # ZHĹIJ ÍĹĞÂĨHĮĻ ŃÎHĹH ĨĮĴŁĻ ĽĞĬŁHĽ ŃHĹH ĮHÍĞĻĨĽH
        ĽHĬÌgÂĨĮĴŁĻĽälHĬÌgĨĮĴŁĻĽ l·ł flã ł fl

    # CĞĬĢŁĬĞĻH ĴĹHÂĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ÂHÌ ĴĹHÂĨĢĻĨIJĮĽâĽHĬÌd IJŁĻĴŁĻĽãf
        ĹHĻŁĹĮ IJŁĻĴŁĻĽ


# SIJÌĻĬĞŃ ĞĢĻĨĽĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĽĞĻĨIJĮçSIJÌĻĬĞŃf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        # RHĬHĬĠHĹ ĨĮĴŁĻ ĽĞĬŁHĽ
        ĽHĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ

        # GHĻ ŁĮĮIJĹĬĞĬĨĨŃHĤ ĴĹIJĠĞĠĨĬĨĨĻĨHĽ
        HŃĴçĽĞĬŁHĽ ł ĮĴgHŃĴâĨĮĴŁĻĽ h ĮĴgĬĞŃâĨĮĴŁĻĽd ĞŃĨĽłĝd
                                                ĨHHĴĤĨĬĽłTĹŁHãã
```

```
        ſ NIJĹİĞĬĨŅĦ ĻÎĦİ ÌIJĹ ĦĞĢÎ ĽĞİĴĬĦ
        ĴĹIJĠĞĠĬĬĨĻĨĦĽ ł ĦŇĴçĿĞĬŁĦĽ w ĮĴgĽŁİâĦŇĴçĿĞĬŁĦĽd ĞŇĨĽłĝd
                                                ĪĦĦĴĤĨİĽłTĹŁĦã

        ĽĦĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĬĬĨĻĨĦĽ

    ſ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ÂĿĞĬŁĦĽãf

        ſ CĹĦĞĻĦ ŁĮĨĮĨĻĨĞĬĨŅĦĤ ĞĹĹĞÑ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĮĴgĦİĴĻÑçĬĨĨĦâĤĿĞĬŁĦĽã

        ſ EĮŁİĦĹĞĻĦ IJŁĻĴŁĻĽ ĞĮĤ ÍĹĞĤĨĦĮĻĽ
        ÌIJĹ ĨĮĤĦÑd âĽĨĮÍĬĦçIJŁĻĴŁĻd ĽĨĮÍĬĦçĤĿĞĬŁĦĽã ĨĮ ċ
                ĦĮŁİĦĹĞĻĦâŅĨĴâĽĦĬÌgIJŁĻĴŁĻd ĤĿĞĬŁĦĽããf
            ſ FĬĞĻĻĦĮ IJŁĻĴŁĻ ĞĹĹĞÑ
            ĽĨĮÍĬĦçIJŁĻĴŁĻ ł ĽĨĮÍĬĦçIJŁĻĴŁĻgĹĦĽĬĞĴĦâhĝd ĝã
            ſ CĞĬĢŁĬĞĻĦ JĞĢIJĠĬĞĮ İĞĻĹĨŇ IJÌ ĻĬĦ IJŁĻĴŁĻ ĞĮĤ
            ÏĞĢIJĠĬĞĮçİĞĻĹĨŇ ł ĮĴgĤĬĞĹÌÌĬĞĻâĽĨĮÍĬĦçIJŁĻĴŁĻã h ċ
                                ĮĴgĤIJĻâĽĨĮÍĬĦçIJŁĻĴŁĻd ĽĨĮÍĬĦçIJŁĻĴŁĻgTã
            ſ CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦhŃĨĽĦ ÍĹĞĤĨĦĮĻ
            ſ ĞĮĤ ĞĤĤ ĨĻ ĻIJ ĻĬĦ ĞĹĹĞÑ IJÌ ĽĞİĴĬĦ ÍĹĞĤĨĦĮĻĽ
            ĽĦĬÌgĤĨĮĴŁĻĽäĨĮĤĦÑã ł ĮĴgĤIJĻâÏĞĢIJĠĬĞĮçİĞĻĹĨŇd
                                        ĽĨĮÍĬĦçĤĿĞĬŁĦĽã

    ſ CĞĬĢŁĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ÂĦÌ ĴĹĦĤĨĢĻĨIJĮĽâĽĦĬÌd IJŁĻĴŁĻĽãf
        ĹĦĻŁĹĮ ĮĴgĞĹÍİĞÑâIJŁĻĴŁĻĽd ĞŇĨĽłĝã


ſ SĨÍİIJĨĤ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçSĨÍİIJĨĤf

    ſ FIJĹŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ſ SĞĿĦ ĨĮĴŁĻ ĞĮĤ ĢĞĬĢŁĬĞĻĦwĽĞĿĦ IJŁĻĴŁĻ
        ſ IJÌ ĻĬĦ ĽĨÍİIJĨĤ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĝ w âĝ κ ĮĴgĦŇĴâhĨĮĴŁĻĽãã

    ſ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ÂĿĞĬŁĦĽãf
        ſ DĦĹĨĿĞĻĨĿĦ h ĢĞĬĢŁĬĞĻĦĽ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĻĬĦ ĽĨÍİIJĨĤ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ÂĿĞĬŁĦĽ ắ âĝ h ĽĦĬÌgIJŁĻĴŁĻã ắ ĽĦĬÌgIJŁĻĴŁĻ

    ſ CĞĬĢŁĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ÂĦÌ ĴĹĦĤĨĢĻĨIJĮĽâĽĦĬÌd IJŁĻĴŁĻĽãf
        ĹĦĻŁĹĮ âIJŁĻĴŁĻĽ ń flgĥã ắ ĝ
```

```
# LĨĮĦĞĹ ĞĢĻĨŁĞĻĨIJĮ
ĢĬĞĹĹ AĢĻĨŁĞĻĨIJĮçLĨĮĦĞĹf

    # FIJĹŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ÌIJĹŃĞĹÂâĹĦĬÌd ĨĮĴŁĻĹd ĻĹĞĨĮĨĮĺãf
        # JŁĹĻ ĹĦİĦİĠĦĹ ŁĞĬŁĦĹ
        ĹĦĬÌgĨĮĴŁĻĹ ł ĨĮĴŁĻĹ
        ĹĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĹ

    # BĞĢĪŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ĠĞĢĪŃĞĹÂâĹĦĬÌd ÂŁĞĬŁĦĹãf
        # ÂĦĹĨŁĞĻĨŁĦ ĨĹ ĝd ĝ ǻ ÂŁĞĬŁĦĹ ł ÂŁĞĬŁĦĹ h ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ
        ĹĦĬÌgÂĨĮĴŁĻĹ ł ÂŁĞĬŁĦĹgĢIJĴÑâã

    # CĞĬĢŁĬĞĻĦ ĴĹĦÂĨĢĻĨIJĮĹ ÌIJĹ IJŁĻĴŁĻĹ
    ÂĦÌ ĴĹĦÂĨĢĻĨIJĮĹâĹĦĬÌd IJŁĻĴŁĻĹãf
        ĹĦĻŁĹĮ IJŁĻĴŁĻĹ


# SGD IJĴĻĨİĨŅĦĹ
ĢĬĞĹĹ OĴĻĨİĨŅĦĹçSGDf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮĺĹd
    # ĬĦĞĹĮĨĮĺ ĹĞĻĦ IJÌ ĝg ĨĹ ÂĦÌĞŁĬĻ ÌIJĹ ĻÎĨĹ IJĴĻĨİĨŅĦĹ
    ÂĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮĺçĹĞĻĦłĝgd ÂĦĢĞÑłflgd İIJİĦĮĻŁİłflgãf
        ĹĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĹĦĬÌgÂĦĢĞÑ ł ÂĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgİIJİĦĮĻŁİ ł İIJİĦĮĻŁİ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴÂĞĻĦĹ
    ÂĦÌ ĴĹĦçŁĴÂĞĻĦçĴĞĹĞİĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgÂĦĢĞÑf
            ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ǻ ċ
                âĝg w âĝg κ ĹĦĬÌgÂĦĢĞÑ ǻ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    # UĴÂĞĻĦ ĴĞĹĞİĦĻĦĹĹ
    ÂĦÌ ŁĴÂĞĻĦçĴĞĹĞİĹâĹĦĬÌd ĬĞÑĦĹãf

        # IÌ ŃĦ ŁĹĦ İIJİĦĮĻŁİ
        ĨÌ ĹĦĬÌgİIJİĦĮĻŁİf

            # IÌ ĬĞÑĦĹ ÂIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ İIJİĦĮĻŁİ ĞĹĹĞÑĹd ĢĹĦĞĻĦ ĻÎĦİ
            # ÌĨĬĬĦÂ ŃĨĻÎ ŅĦĹIJĹ
            ĨÌ ĮIJĻ ÎĞĹĞĻĻĹâĬĞÑĦĹd ďŃĦĨĬÎĻçİIJİĦĮĻŁİĹďãf
                ĬĞÑĦĹgŃĦĨĬÎĻçİIJİĦĮĻŁİĹ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgŃĦĨĬÎĻĹã
                # IÌ ĻÎĦĹĦ ĨĹ ĮIJ İIJİĦĮĻŁİ ĞĹĹĞÑ ÌIJĹ ŃĦĨĬÎĻĹ
                # TÎĦ ĞĹĹĞÑ ÂIJĦĹĮďĻ ĦŃĨĹĻ ÌIJĹ ĠĨĞĹĦĹ ÑĦĻ ĦĨĻÎĦĹg
                ĬĞÑĦĹgĠĨĞĹçİIJİĦĮĻŁİĹ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgĠĨĞĹĦĹã
```

```
            fi BłĨĬÎ ŃĦĨÍÎĮ łĴÎĞĻĦĽ ŃĨĮÎ ĲĴĲĦĮĻłĲ h ĻĞĨĦ ĴĹĦŁĨĲłĽ
            fi łĴÎĞĻĦĽ ĲłĬĮĨĴĬĨĦÎ ĠÑ ĹĦĻĞĨĮ ÌĞĢĻĲĹ ĞĮÎ łĴÎĞĻĦ ŃĨĮÎ
            fi ĢłĹĹĦĮĻ ÍĹĞÎĨĦĮĻĽ
            ŃĦĨÍÎĮçłĴÎĞĻĦĽ ł ċ
                ĽĦĬÌgĲĲĦĮĻłĲ ắ ĬĞÑĦĹgŃĦĨÍÎĮçĲĲĦĮĻłĲĽ h ċ
                ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ĬĞÑĦĹgÎŃĦĨÍÎĮĽ
            ĬĞÑĦĹgŃĦĨÍÎĮçĲĲĦĮĻłĲĽ ł ŃĦĨÍÎĮçłĴÎĞĻĦĽ

            fi BłĨĬÎ ĠĨĞĽ łĴÎĞĻĦĽ
            ĠĨĞĽçłĴÎĞĻĦĽ ł ċ
                ĽĦĬÌgĲĲĦĮĻłĲ ắ ĬĞÑĦĹgĠĨĞĽçĲĲĦĮĻłĲĽ h ċ
                ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ĬĞÑĦĹgÎĠĨĞĽĦĽ
            ĬĞÑĦĹgĠĨĞĽçĲĲĦĮĻłĲĽ ł ĠĨĞĽçłĴÎĞĻĦĽ

        fi VĞĮĨĬĬĞ SGD łĴÎĞĻĦĽ âĞĽ ĠĦÌĲĹĦ ĲĲĦĮĻłĲ łĴÎĞĻĦã
        ĦĬĽĦf
            ŃĦĨÍÎĮçłĴÎĞĻĦĽ ł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                                ĬĞÑĦĹgÎŃĦĨÍÎĮĽ
            ĠĨĞĽçłĴÎĞĻĦĽ ł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                               ĬĞÑĦĹgÎĠĨĞĽĦĽ

        fi UĴÎĞĻĦ ŃĦĨÍÎĮĽ ĞĮÎ ĠĨĞĽĦĽ łĽĨĮÍ ĦĨĮÎĦĹ
        fi ŁĞĮĨĬĬĞ ĲĹ ĲĲĦĮĻłĲ łĴÎĞĻĦĽ
        ĬĞÑĦĹgŃĦĨÍÎĮĽ κł ŃĦĨÍÎĮçłĴÎĞĻĦĽ
        ĬĞÑĦĹgĠĨĞĽĦĽ κł ĠĨĞĽçłĴÎĞĻĦĽ

    fi CĞĬĬ ĲĮĢĦ ĞÌĮĦĹ ĞĮÑ ĴĞĹĞĲĦĮĦĹ łĴÎĞĻĦĽ
    ÎĦÌ ĴĲĹĮçłĴÎĞĻĦçĴĞĹĞĲĽâĽĦĬÌãf
        ĽĦĬÌgĨĮĦĹĞĮĨĲĮĽ κł ĝ


fi AÎĞÍĹĞÎ ĲĴĮĨĲĨÑĦĹ
ĢĬĞĽĽ OĴĮĨĲĨÑĦĹçAÎĞÍĹĞÎf

    fi IĮĨĮĨĞĬĨÑĦ ĲĴĮĨĲĨÑĦĹ h ĽĦĮ ĽĦĮĮĨĮÍĽ
    ÎĦÌ ççĨĮĨĮççâĽĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ÎĦĢĞÑłflgd ĦĴĽĨĬĲĮłĝĦhìãf
        ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgÎĦĢĞÑ ł ÎĦĢĞÑ
        ĽĦĬÌgĨĮĦĹĞĮĨĲĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬĲĮ ł ĦĴĽĨĬĲĮ

    fi CĞĬĬ ĲĮĢĦ ĠĦÌĲĹĦ ĞĮÑ ĴĞĹĞĲĦĮĦĹ łĴÎĞĻĦĽ
    ÎĦÌ ĴĹĦçłĴÎĞĻĦçĴĞĹĞĲĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgÎĦĢĞÑf
            ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĽĦĬÌgÎĦĢĞÑ ắ ĽĦĬÌgĨĮĦĹĞĮĨĲĮĽãã
```

```
    # UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
    ĤĦÌ łĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ĤIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĨĦ ĞĹĹĞÑĽd
        # ĢĹĦĞĻĦ ĻĨĦİ ÌĨĬĬĦĤ ŃĨĻĨ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ĨĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĹĨĨĻçĢĞĢĨĦďãf
            ĬĞÑĦĹgŃĦĨĹĨĨĻçĢĞĢĨĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgŃĦĨĹĨĨĻĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgĠĨĞĽĦĽã

        # UĴĤĞĻĦ ĢĞĢĨĦ ŃĨĻĨ ĽĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ĹĹĞĤĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨĹĨĨĻçĢĞĢĨĦ κł ĬĞÑĦĹgĤŃĦĨĹĨĨĻĽăăğ
        ĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦ κł ĬĞÑĦĹgĤĠĨĞĽĦĽăăğ

        # VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹİĞĬĨŅĞĻĨIJĮ
        # ŃĨĻĨ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĨĦ
        ĬĞÑĦĹgŃĦĨĹĨĨĻĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ă ċ
                         ĬĞÑĦĹgĤŃĦĨĹĨĨĻĽ w ċ
                         âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨĹĨĨĻçĢĞĢĨĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ă ċ
                        ĬĞÑĦĹgĤĠĨĞĽĦĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢĨĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã

    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ


# RMSĴĹIJĴ IJĴĻĨİĨŅĦĹ
ĢĬĞĽĽ OĴĻĨİĨŅĦĹçRMSĴĹIJĴf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĹĽ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮĹçĹĞĻĦłflgflflĝd ĤĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìd
                ĹĴIJłflgîãf
        ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬÌgĹĴIJ ł ĹĴIJ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgĤĦĢĞÑf
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ă ċ
                âĝg w âĝg κ ĽĦĬÌgĤĦĢĞÑ ă ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã
```

```
    # UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
    ĤĦÌ łĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ĤIJĦĹ ĮIJĻ ǦIJĮĻĞĨĮ ǦĞǦĨĦ ĞĹĹĞÑĽd
        # ǦĹĦĞĻĦ ĻĨĦİ ÌĨĬĬĦĤ ŃĨĻĨ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ĨĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĹĨĨĻçǦĞǦĨĦďãf
            ĬĞÑĦĹgŃĦĨĹĨĨĻçǦĞǦĨĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgŃĦĨĹĨĨĻĽã
            ĬĞÑĦĹgĠĨĞĽçǦĞǦĨĦ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgĠĨĞĽĦĽã

        # UĴĤĞĻĦ ǦĞǦĨĦ ŃĨĻĨ ĽĶŁĞĹĦĤ ǦŁĹĹĦĮĻ ĹĹĞĤĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨĹĨĨĻçǦĞǦĨĦ ł ĽĦĬÌgĹĨIJ ǎ ĬĞÑĦĹgŃĦĨĹĨĨĻçǦĞǦĨĦ κ ċ
            âĝ h ĽĦĬÌgĹĨIJã ǎ ĬĞÑĦĹgĤŃĦĨĹĨĨĻĽǎǎğ
        ĬĞÑĦĹgĠĨĞĽçǦĞǦĨĦ ł ĽĦĬÌgĹĨIJ ǎ ĬĞÑĦĹgĠĨĞĽçǦĞǦĨĦ κ ċ
            âĝ h ĽĦĬÌgĹĨIJã ǎ ĬĞÑĦĹgĤĠĨĞĽĦĽǎǎğ

        # VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹİĞĬĨŅĞĻĨIJĮ
        # ŃĨĻĨ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ǦĞǦĨĦ
        ĬĞÑĦĹgŃĦĨĹĨĨĻĽ κł hĽĦĬÌgǦŁĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ǎ ċ
                         ĬĞÑĦĹgĤŃĦĨĹĨĨĻĽ w ċ
                         âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨĹĨĨĻçǦĞǦĨĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgǦŁĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ǎ ċ
                        ĬĞÑĦĹgĤĠĨĞĽĦĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçǦĞǦĨĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã

    # CĞĬĬ IJĮǦĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ


# AĤĞİ IJĴĻĨİĨŅĦĹ
ǦĬĞĽĽ OĴĻĨİĨŅĦĹçAĤĞİf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĹĽ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮĹçĹĞĻĦłflgflflĝd ĤĦǦĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìd
                ĠĦĻĞçĝłflgîd ĠĦĻĞçğłflgîîîãf
        ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgǦŁĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĬĦĞĹĮĨĮĹçĹĞĻĦ
        ĽĦĬÌgĤĦǦĞÑ ł ĤĦǦĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬÌgĠĦĻĞçĝ ł ĠĦĻĞçĝ
        ĽĦĬÌgĠĦĻĞçğ ł ĠĦĻĞçğ

    # CĞĬĬ IJĮǦĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgĤĦǦĞÑf
            ĽĦĬÌgǦŁĹĹĦĮĻçĬĦĞĹĮĨĮĹçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĹçĹĞĻĦ ǎ ċ
                âĝg w âĝg κ ĽĦĬÌgĤĦǦĞÑ ǎ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã
```

```
ƒ UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
ÎĦÌ łĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

    ƒ IÌ ĬĞÑĦĹ ĤIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢÎĦ ĞĹĹĞÑĽd
    ƒ ĢĹĦĞĻĦ ĻÎĦİ ÌĨĬĬĦĤ ŃĨĻÎ ŅĦĹIJĽ
    ĨÌ ĮIJĻ ÎĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍÎĻçĢĞĢÎĦďãf
        ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻłİĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨÍÎĻĽã
        ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨÍÎĻĽã
        ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã
        ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

    ƒ UĴĤĞĻĦ İIJİĦĮĻłİ  ŃĨĻÎ ĢłĹĹĦĮĻ ÍĹĞĤĨĦĮĻĽ
    ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻłİĽ ł ĽĦĬÌgĠĦĻĞçĝ ắ ċ
                                ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻłİĽ κ ċ
                                âĝ h ĽĦĬÌgĠĦĻĞçĝã ắ ĬĞÑĦĹgĤŃĦĨÍÎĻĽ
    ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ ł ĽĦĬÌgĠĦĻĞçĝ ắ ċ
                              ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ κ ċ
                              âĝ h ĽĦĬÌgĠĦĻĞçĝã ắ ĬĞÑĦĹgĤĠĨĞĽĦĽ
    ƒ GĦĻ ĢIJĹĹĦĢĻĦĤ İIJİĦĮĻłİ
    ƒ ĽĦĬÌgĨĮĦĹĞĻĨIJĮ ĨĽ fl ĞĻ ÌĨĹĹĮ ĴĞĽĽ
    ƒ ĞĮĤ ŃĦ ĮĦĦĤ ĻIJ ĽĮĞĹĮ ŃĨĻÎ ĝ ÎĦĹĦ
    ŃĦĨÍÎĻçİIJİĦĮĻłİĽçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻłİĽ w ċ
        âĝ h ĽĦĬÌgĠĦĻĞçĝ ắắ âĽĦĬÌgĨĮĦĹĞĻĨIJĮĽ κ ĝãã
    ĠĨĞĽçİIJİĦĮĻłİĽçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĽçİIJİĦĮĻłİĽ w ċ
        âĝ h ĽĦĬÌgĠĦĻĞçĝ ắắ âĽĦĬÌgĨĮĦĹĞĻĨIJĮĽ κ ĝãã
    ƒ UĴĤĞĻĦ ĢĞĢÎĦ ŃĨĻÎ ĽĶłĞĹĦĤ ĢłĹĹĦĮĻ ÍĹĞĤĨĦĮĻĽ
    ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĽĦĬÌgĠĦĻĞçğ ắ ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ κ ċ
        âĝ h ĽĦĬÌgĠĦĻĞçğã ắ ĬĞÑĦĹgĤŃĦĨÍÎĻĽắắğ
    ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĽĦĬÌgĠĦĻĞçğ ắ ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ κ ċ
        âĝ h ĽĦĬÌgĠĦĻĞçğã ắ ĬĞÑĦĹgĤĠĨĞĽĦĽắắğ
    ƒ GĦĻ ĢIJĹĹĦĢĻĦĤ ĢĞĢÎĦ
    ŃĦĨÍÎĻçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ w ċ
        âĝ h ĽĦĬÌgĠĦĻĞçğ ắắ âĽĦĬÌgĨĮĦĹĞĻĨIJĮĽ κ ĝãã
    ĠĨĞĽçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ w ċ
        âĝ h ĽĦĬÌgĠĦĻĞçğ ắắ âĽĦĬÌgĨĮĦĹĞĻĨIJĮĽ κ ĝãã

    ƒ VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ łĴĤĞĻĦ κ ĮIJĹİĞĬĨŅĞĻĨIJĮ
    ƒ ŃĨĻÎ ĽĶłĞĹĦ ĹIJIJĻĦĤ ĢĞĢÎĦ
    ĬĞÑĦĹgŃĦĨÍÎĻĽ κł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                     ŃĦĨÍÎĻçİIJİĦĮĻłİĽçĢIJĹĹĦĢĻĦĤ w ċ
                     âĮĴgĽĶĹĻâŃĦĨÍÎĻçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤã κ
                         ĽĦĬÌgĦĴĽĨĬIJĮã
    ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                     ĠĨĞĽçİIJİĦĮĻłİĽçĢIJĹĹĦĢĻĦĤ w ċ
                     âĮĴgĽĶĹĻâĠĨĞĽçĢĞĢÎĦçĢIJĹĹĦĢĻĦĤã κ
                         ĽĦĬÌgĦĴĽĨĬIJĮã
```

```
    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ÂĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ


# CIJİİIJĮ ĬIJĽĽ ĢĬĞĽĽ
ĢĬĞĽĽ LIJĽĽf

    # RĦÍŁĬĞĹĨĢĞĻĨIJĮ ĬIJĽĽ ĢĞĬĢŁĬĞĻĨIJĮ
    ÂĦÌ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽâĽĦĬÌãf

        # fl ĠÑ ÂĦÌĞŁĬĻ
        ĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ ł fl

        # CĞĬĢŁĬĞĻĦ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮ ĬIJĽĽ
        # ĨĻĦĹĞĻĦ ĞĬĬ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
        ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽf

            # Lĝ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮ h ŃĦĨÍÎĻĽ
            # ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ fl
            ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨĢĞĻĦĹçĬĝ ń flf
                ĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨĢĞĻĦĹçĬĝ ő ċ
                                        ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgŃĦĨÍÎĻĽãã

            # Lğ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮ h ŃĦĨÍÎĻĽ
            ĨÌ ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨĢĞĻĦĹçĬğ ń flf
                ĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨĢĞĻĦĹçĬğ ő ċ
                                        ĮĴgĽŁİâĬĞÑĦĹgŃĦĨÍÎĻĽ ő ċ
                                                ĬĞÑĦĹgŃĦĨÍÎĻĽã

            # Lĝ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮ h ĠĨĞĽĦĽ
            # ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ fl
            ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨĢĞĻĦĹçĬĝ ń flf
                ĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨĢĞĻĦĹçĬĝ ő ċ
                                        ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgĠĨĞĽĦĽãã

            # Lğ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮ h ĠĨĞĽĦĽ
            ĨÌ ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨĢĞĻĦĹçĬğ ń flf
                ĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ κł ĬĞÑĦĹgĠĨĞĽçĹĦÍŁĬĞĹĨĢĞĻĦĹçĬğ ő ċ
                                        ĮĴgĽŁİâĬĞÑĦĹgĠĨĞĽĦĽ ő ċ
                                                ĬĞÑĦĹgĠĨĞĽĦĽã

        ĹĦĻŁĹĮ ĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽ

    # SĦĻwĹĦİĦİĠĦĹ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
    ÂĦÌ ĹĦİĦİĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽâĽĦĬÌd ĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽãf
        ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ ł ĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ
```

```
    fi CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽĦĽ
    fi ÍĨĿĦĮ ĬIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñd ǻd ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮłFĞĬĽĦãf

        fi CĞĬĢŁĬĞĻĦ ĽĞĬĴĬĦ ĬIJĽĽĦĽ
        ĽĞĬĴĬĦçĬIJĽĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        fi CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĽĽ
        ÂĞĻĞçĬIJĽĽ ł ĮĴgĬĦĞĮâĽĞĬĴĬĦçĬIJĽĽĦĽã

        fi IÌ ÏŁĽĻ ÂĞĻĞ ĬIJĽĽ h ĹĦĻŁĹĮ ĨĻ
        ĨÌ ĮIJĻ ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮf
            ĹĦĻŁĹĮ ÂĞĻĞçĬIJĽĽ

        fi RĦĻŁĹĮ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽĦĽ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĽĽd ĽĦĬÌgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽâã


fi CĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ
ĢĬĞĽĽ LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâLIJĽĽãf

    fi FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ÑçĴĹĦÂd ÑçĻĹŁĦãf

        fi NŁĬĠĦĹ IJÌ ĽĞĬĴĬĦĽ ĨĮ Ğ ĠĞĻĢÎ
        ĽĞĬĴĬĦĽ ł ĬĦĮâÑçĴĹĦÂã

        fi CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦĿĦĮĻ ÂĨĿĨĽĨIJĮ ĠÑ fl
        fi CĬĨĴ ĠIJĻÎ ĹĨÂĦĽ ĻIJ ĮIJĻ ÂĹĞÍ ĬĦĞĮ ĻIJŃĞĹÂĽ ĞĮÑ ĿĞĬŁĦ
        ÑçĴĹĦÂçĢĬĨĴĴĦÂ ł ĮĴgĢĬĨĴâÑçĴĹĦÂd ĝĦhìd ĝ h ĝĦhìã

        fi PĹIJĠĞĠĨĬĨĻĨĦĽ ÌIJĹ ĻĞĹÍĦĻ ĿĞĬŁĦĽ h
        fi IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ĝf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĽ ł ÑçĴĹĦÂçĢĬĨĴĴĦÂä
                ĹĞĮÍĦâĽĞĬĴĬĦĽãd
                ÑçĻĹŁĦ
            ã

        fi MĞĽĨ ĿĞĬŁĦĽ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJÂĦÂ ĬĞĠĦĬĽ
        ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĽ ł ĮĴgĽŁĬâ
                ÑçĴĹĦÂçĢĬĨĴĴĦÂ ǻ ÑçĻĹŁĦd
                ĞŃĨĽłĝ
            ã

        fi LIJĽĽĦĽ
        ĮĦÍĞĻĨĿĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĽ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĽã
        ĹĦĻŁĹĮ ĮĦÍĞĻĨĿĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĽ
```

```
    fi BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎHÌ ĠĞĢĪŃĞĹÎâĽHĬÌd ÎĹĞĬŁHĽd ÑçĻĹŁHãf

        fi NŁİĠHĹ IJÌ ĽĞİĴĬHĽ
        ĽĞİĴĬHĽ ł ĬHĮâÎĹĞĬŁHĽã
        fi NŁİĠHĹ IJÌ ĬĞĠHĬĽ ĨĮ HĹHĹÑ ĽĞİĴĬH
        fi WHďĬĬ ŁĽH ĻÎH ÌĨĹĽĻ ĽĞİĴĬH ĻIJ ĢIJŁĮĻ ĻÎHİ
        ĬĞĠHĬĽ ł ĬHĮâÎĹĞĬŁHĽäflãã

        fi IÌ ĬĞĠHĬĽ ĞĹH ĽĴĞĹĽHd ĻŁĹĮ ĻÎHİ ĨĮĻIJ IJĮHhÎIJĻ ĹHĢĻIJĹ
        ĨÌ ĬHĮâÑçĻĹŁHgĽÎĞĴHã łł ĝf
            ÑçĻĹŁH ł ĮĴgHÑHâĬĞĠHĬĽãäÑçĻĹŁHã

        fi CĞĬĢŁĬĞĻH ÍĹĞÎĨHĮĻ
        ĽHĬÌgÎĨĮĴŁĻĽ ł hÑçĻĹŁH w ÎĹĞĬŁHĽ
        fi NIJĹİĞĬĨŅH ÍĹĞÎĨHĮĻ
        ĽHĬÌgÎĨĮĴŁĻĽ ł ĽHĬÌgÎĨĮĴŁĻĽ w ĽĞİĴĬHĽ


fi SIJÌĻİĞŇ ĢĬĞĽĽĨÌÌHĹ h ĢIJİĠĨĮHÎ SIJÌĻİĞŇ ĞĢĻĨĹĞĻĨIJĮ
fi ĞĮÎ ĢĹIJĹĽhHĮĻĹIJĴÑ ĬIJĹĽ ÌIJĹ ÌĞĽĻHĹ ĠĞĢĪŃĞĹÎ ĽĻHĴ
ĢĬĞĽĽ AĢĻĨĹĞĻĨIJĮçSIJÌĻİĞŇçLIJĹĽçCĞĻHÍIJĹĨĢĞĬCĹIJĹĽHĮĻĹIJĴÑâãf

    fi BĞĢĪŃĞĹÎ ĴĞĽĽ
    ÎHÌ ĠĞĢĪŃĞĹÎâĽHĬÌd ÎĹĞĬŁHĽd ÑçĻĹŁHãf

        fi NŁİĠHĹ IJÌ ĽĞİĴĬHĽ
        ĽĞİĴĬHĽ ł ĬHĮâÎĹĞĬŁHĽã

        fi IÌ ĬĞĠHĬĽ ĞĹH IJĮHhÎIJĻ HĮĢIJÎHHÎd
        fi ĻŁĹĮ ĻÎHİ ĨĮĻIJ ÎĨĽĢĹHĻH ĹĞĬŁHĽ
        ĨÌ ĬHĮâÑçĻĹŁHgĽÎĞĴHã łł ğf
            ÑçĻĹŁH ł ĮĴgĞĹÍİĞŇâÑçĻĹŁHd ĞŇĨĽłĝã

        fi CIJĴÑ ĽIJ ŃH ĢĞĮ ĽĞÌHĬÑ İIJÎĨÌÑ
        ĽHĬÌgÎĨĮĴŁĻĽ ł ÎĹĞĬŁHĽgĢIJĴÑâã
        fi CĞĬĢŁĬĞĻH ÍĹĞÎĨHĮĻ
        ĽHĬÌgÎĨĮĴŁĻĽäĹĞĮÍHâĽĞİĴĬHĽãd ÑçĻĹŁHã hł ĝ
        fi NIJĹİĞĬĨŅH ÍĹĞÎĨHĮĻ
        ĽHĬÌgÎĨĮĴŁĻĽ ł ĽHĬÌgÎĨĮĴŁĻĽ w ĽĞİĴĬHĽ
```

```
ƒ BĨĮĞĹÑ ĢĹIJĹĹhĦĮĻĹIJĴÑ ĬIJĹĹ
ĢĬĞĹĹ LIJĹĹçBĨĮĞĹÑCĹIJĹĹĦĮĻĹIJĴÑâLIJĹĹãf

    ƒ FIJĹŃĞĹĤ ĴĞĹĹ
    ÂĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        ƒ CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ÂĨŁĨĹĨIJĮ ĠÑ ƒl
        ƒ CĬĨĴ ĠIJĻÎ ĹĨÂĦĹ ĻIJ ĮIJĻ ÂĹĞÍ ĬĦĞĮ ĻIJŃĞĹÂĹ ĞĮÑ ŁĞĬŁĦ
        ÑçĴĹĦÂçĢĬĨĴĴĦÂ ł ĮĴgĢĬĨĴâÑçĴĹĦÂd ĝĦhìd ĝ h ĝĦhìã

        ƒ CĞĬĢŁĬĞĻĦ ĹĞĬĴĬĦhŃĨĹĦ ĬIJĹĹ
        ĹĞĬĴĬĦçĬIJĹĹĦĹ ł hâÑçĻĹŁĦ ǻ ĮĴgĬIJÍâÑçĴĹĦÂçĢĬĨĴĴĦÂã κ
                          âĝ h ÑçĻĹŁĦã ǻ ĮĴgĬIJÍâĝ h ÑçĴĹĦÂçĢĬĨĴĴĦÂãã
        ĹĞĬĴĬĦçĬIJĹĹĦĹ ł ĮĴgĬĦĞĮâĹĞĬĴĬĦçĬIJĹĹĦĹd ĞŃĨĹłhĝã

        ƒ RĦĻŁĹĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ĹĞĬĴĬĦçĬIJĹĹĦĹ

    ƒ BĞĢĪŃĞĹĤ ĴĞĹĹ
    ÂĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ÂĹĞĬŁĦĹd ÑçĻĹŁĦãf

        ƒ NŁĬĠĦĹ IJÌ ĹĞĬĴĬĦĹ
        ĹĞĬĴĬĦĹ ł ĬĦĮâÂĹĞĬŁĦĹã
        ƒ NŁĬĠĦĹ IJÌ IJŁĻĴŁĻĹ ĨĮ ĦĹĦĹÑ ĹĞĬĴĬĦ
        ƒ WĦďĬĬ ŁĹĦ ĻÎĦ ÌĨĹĹĻ ĹĞĬĴĬĦ ĻIJ ĢIJŁĮĻ ĻÎĦĬ
        IJŁĻĴŁĻĹ ł ĬĦĮâÂĹĞĬŁĦĹäƒlãã

        ƒ CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ÂĨŁĨĹĨIJĮ ĠÑ ƒl
        ƒ CĬĨĴ ĠIJĻÎ ĹĨÂĦĹ ĻIJ ĮIJĻ ÂĹĞÍ ĬĦĞĮ ĻIJŃĞĹÂĹ ĞĮÑ ŁĞĬŁĦ
        ĢĬĨĴĴĦÂçÂĹĞĬŁĦĹ ł ĮĴgĢĬĨĴâÂĹĞĬŁĦĹd ĝĦhìd ĝ h ĝĦhìã

        ƒ CĞĬĢŁĬĞĻĦ ÍĹĞÂĨĦĮĻ
        ĹĦĬÌgÂĨĮĴŁĻĹ ł hâÑçĻĹŁĦ w ĢĬĨĴĴĦÂçÂĹĞĬŁĦĹ h
                         âĝ h ÑçĻĹŁĦã w âĝ h ĢĬĨĴĴĦÂçÂĹĞĬŁĦĹãã w IJŁĻĴŁĻĹ
        ƒ NIJĹĬĞĬĨŅĦ ÍĹĞÂĨĦĮĻ
        ĹĦĬÌgÂĨĮĴŁĻĹ ł ĹĦĬÌgÂĨĮĴŁĻĹ w ĹĞĬĴĬĦĹ


ƒ MĦĞĮ SĶŁĞĹĦÂ EĹĹIJĹ ĬIJĹĹ
ĢĬĞĹĹ LIJĹĹçMĦĞĮSĶŁĞĹĦÂEĹĹIJĹâLIJĹĹãf  ƒ Lğ ĬIJĹĹ

    ƒ FIJĹŃĞĹĤ ĴĞĹĹ
    ÂĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        ƒ CĞĬĢŁĬĞĻĦ ĬIJĹĹ
        ĹĞĬĴĬĦçĬIJĹĹĦĹ ł ĮĴgĬĦĞĮââÑçĻĹŁĦ h ÑçĴĹĦÂãǻǻğd ĞŃĨĹłhĝã

        ƒ RĦĻŁĹĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ĹĞĬĴĬĦçĬIJĹĹĦĹ
```

```
    ſ BĞĢĪŃĞĹĤ ĴĞĹĹ
    ÎĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ÎĦĹĞĬŁĦĹd ÑçĻĹŁĦãf

        ſ NŁİĠĦĹ IJÌ ĹĞİĴĬĦĹ
        ĹĞİĴĬĦĹ ł ĬĦĮâÎĦĹĞĬŁĦĹã
        ſ NŁİĠĦĹ IJÌ IJŁĻĴŁĻĹ ĨĮ ĦĹĦĹÑ ĹĞİĴĬĦ
        ſ WĦďĬĬ ŁĹĦ ĻÎĦ ÌĨĹĹĻ ĹĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻÎĦİ
        IJŁĻĴŁĻĹ ł ĬĦĮâÎĦĹĞĬŁĦĹäflãã

        ſ GĹĞÎĨĨĦĮĻ IJĮ ĹĞĬŁĦĹ
        ĹĦĬÌgÎĨĨĮĴŁĻĹ ł hğ ấ âÑçĻĹŁĦ h ÎĦĹĞĬŁĦĹã w IJŁĻĴŁĻĹ
        ſ NIJĹİĞĬĨŅĦ ÍĹĞÎĨĨĦĮĻ
        ĹĦĬÌgÎĨĨĮĴŁĻĹ ł ĹĦĬÌgÎĨĨĮĴŁĻĹ w ĹĞİĴĬĦĹ


ſ MĦĞĮ AĠĹIJĬŁĻĦ EĹĹIJĹ ĬIJĹĹ
ĢĬĞĹĹ LIJĹĹçMĦĞĮAĠĹIJĬŁĻĦEĹĹIJĹâLIJĹĹãf  ſ Lĝ ĬIJĹĹ

    ÎĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        ſ CĞĬĢŁĬĞĻĦ ĬIJĹĹ
        ĹĞİĴĬĦçĬIJĹĹĦĹ ł ĮĴgİĦĞĮâĮĴgĞĠĹâÑçĻĹŁĦ h ÑçĴĹĦĤãd ĞŃĬĹłhĝã

        ſ RĦĻŁĹĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ĹĞİĴĬĦçĬIJĹĹĦĹ

    ſ BĞĢĪŃĞĹĤ ĴĞĹĹ
    ÎĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ÎĦĹĞĬŁĦĹd ÑçĻĹŁĦãf

        ſ NŁİĠĦĹ IJÌ ĹĞİĴĬĦĹ
        ĹĞİĴĬĦĹ ł ĬĦĮâÎĦĹĞĬŁĦĹã
        ſ NŁİĠĦĹ IJÌ IJŁĻĴŁĻĹ ĨĮ ĦĹĦĹÑ ĹĞİĴĬĦ
        ſ WĦďĬĬ ŁĹĦ ĻÎĦ ÌĨĹĹĻ ĹĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻÎĦİ
        IJŁĻĴŁĻĹ ł ĬĦĮâÎĦĹĞĬŁĦĹäflãã

        ſ CĞĬĢŁĬĞĻĦ ÍĹĞÎĨĨĦĮĻ
        ĹĦĬÌgÎĨĨĮĴŁĻĹ ł ĮĴgĹĨĨĮâÑçĻĹŁĦ h ÎĦĹĞĬŁĦĹã w IJŁĻĴŁĻĹ
        ſ NIJĹİĞĬĨŅĦ ÍĹĞÎĨĨĦĮĻ
        ĹĦĬÌgÎĨĨĮĴŁĻĹ ł ĹĦĬÌgÎĨĨĮĴŁĻĹ w ĹĞİĴĬĦĹ
```

```
ƒ CIJİİIJĮ ĞĢĢŁĹĞĢÑ ĢĬĞĽĽ
ĢĬĞĽĽ AĢĢŁĹĞĢÑf

    ƒ CĞĬĢŁĬĞĻĦĽ ĞĮ ĞĢĢŁĹĞĢÑ
    ƒ ÍĨĿĦĮ ĴĹĦĤĨĢĻĨIJĮĽ ĞĮĤ ÍĹIJŁĮĤ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ĤĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd ĴĹĦĤĨĢĻĨIJĮĽd Ñãf

        ƒ GĦĻ ĢIJİĴĞĹĨĽIJĮ ĹĦĽŁĬĻĽ
        ĢIJİĴĞĹĨĽIJĮĽ ł ĽĦĬÌgĢIJİĴĞĹĦâĴĹĦĤĨĢĻĨIJĮĽd Ñã

        ƒ CĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĢIJİĴĞĹĨĽIJĮĽã

        ƒ RĦĻŁĹĮ ĞĢĢŁĹĞĢÑ
        ĹĦĻŁĹĮ ĞĢĢŁĹĞĢÑ


ƒ AĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ ÌIJĹ ĢĬĞĽĽĨÌÌĨĢĞĻĨIJĮ İIJĤĦĬ
ĢĬĞĽĽ AĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâAĢĢŁĹĞĢÑãf

    ƒ NIJ ĨĮĨĻĨĞĬĨĨĢĞĻĨIJĮ ĨĽ ĮĦĦĤĦĤ
    ĤĦÌ ĨĮĨĻâĽĦĬÌd Ñãf
        ĴĞĽĽ

    ƒ CIJİĴĞĹĦĽ ĴĹĦĤĨĢĻĨIJĮĽ ĻIJ ĻĤĦ ÍĹIJŁĮĤ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ĤĦÌ ĢIJİĴĞĹĦâĽĦĬÌd ĴĹĦĤĨĢĻĨIJĮĽd Ñãf
        ĨÌ ĬĦĮâÑgĽÎĞĴĦã łł ğf
            Ñ ł ĮĴgĞĹÍİĞÑâÑd ĞÑĨĽłĝã
        ĹĦĻŁĹĮ ĴĹĦĤĨĢĻĨIJĮĽ łł Ñ


ƒ AĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ ÌIJĹ ĹĦÍĹĦĽĽĨIJĮ İIJĤĦĬ
ĢĬĞĽĽ AĢĢŁĹĞĢÑçRĦÍĹĦĽĽĨIJĮâAĢĢŁĹĞĢÑãf

    ĤĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ƒ CĹĦĞĻĦ ĴĹĦĢĨĽĨIJĮ ĴĹIJĴĦĹĻÑ
        ĽĦĬÌgĴĹĦĢĨĽĨIJĮ ł NIJĮĦ

    ƒ CĞĬĢŁĬĞĻĦĽ ĴĹĦĢĨĽĨIJĮ ĿĞĬŁĦ
    ƒ ĠĞĽĦĤ IJĮ ĴĞĽĽĦĤ ĨĮ ÍĹIJŁĮĤ ĻĹŁĻÎ ĿĞĬŁĦĽ
    ĤĦÌ ĨĮĨĻâĽĦĬÌd Ñd ĹĦĨĮĨĻłFĞĬĽĦãf
        ĨÌ ĽĦĬÌgĴĹĦĢĨĽĨIJĮ ĨĽ NIJĮĦ IJĹ ĹĦĨĮĨĻf
            ĽĦĬÌgĴĹĦĢĨĽĨIJĮ ł ĮĴgĽĻĤâÑã w ğĥfl
```

```
    ſ CIJİĴĞĹĦĽ ĴĹĦĤĨĢĻĨIJĮĽ ĻIJ ĻĨĦ ÍĹIJŁĮĤ ĻĹŁĻĨ ŁĞĬŁĦĽ
    ĤĦÌ ĢIJİĴĞĹĦâĽĦĬÌd ĴĹĦĤĨĢĻĨIJĮĽd Ñãf
        ĹĦĻŁĹĮ ĮĴgĞĠĽIJĬŁĻĦâĴĹĦĤĨĢĻĨIJĮĽ h Ñã l· ĽĦĬÌgĴĹĦĢĨĽĨIJĮ


ſ MIJĤĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJĤĦĬf

    ĤĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ſ CĹĦĞĻĦ Ğ ĬĨĽĻ IJÌ ĮĦĻŃIJĹĨ IJĠÏĦĢĻĽ
        ĽĦĬÌgĬĞÑĦĹĽ ł äā
        ſ SIJÌĻĬĞÑ ĢĬĞĽĽĨÌĨĦĹďĽ IJŁĻĴŁĻ IJĠÏĦĢĻ
        ĽĦĬÌgĽIJÌĻĬĞÑçĢĬĞĽĽĨÌĨĦĹçIJŁĻĴŁĻ ł NIJĮĦ

    ſ AĤĤ IJĠÏĦĢĻĽ ĻIJ ĻĨĦ ĬIJĤĦĬ
    ĤĦÌ ĞĤĤâĽĦĬÌd ĬĞÑĦĹãf
        ĽĦĬÌgĬĞÑĦĹĽgĞĴĴĦĮĤâĬĞÑĦĹã

    ſ SĦĻ ĬIJĽĽd IJĴĻĨĬĨÑĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ
    ĤĦÌ ĽĦĻâĽĦĬÌd ǻd ĬIJĽĽd IJĴĻĨĬĨÑĦĹd ĞĢĢŁĹĞĢÑãf
        ĽĦĬÌgĬIJĽĽ ł ĬIJĽĽ
        ĽĦĬÌgIJĴĻĨĬĨÑĦĹ ł IJĴĻĨĬĨÑĦĹ
        ĽĦĬÌgĞĢĢŁĹĞĢÑ ł ĞĢĢŁĹĞĢÑ

    ſ FĨĮĞĬĨÑĦ ĻĨĦ ĬIJĤĦĬ
    ĤĦÌ ÌĨĮĞĬĨÑĦâĽĦĬÌãf

        ſ CĹĦĞĻĦ ĞĮĤ ĽĦĻ ĻĨĦ ĨĮĴŁĻ ĬĞÑĦĹ
        ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ ł LĞÑĦĹçIĮĴŁĻâã

        ſ CIJŁĮĻ ĞĬĬ ĻĨĦ IJĠÏĦĢĻĽ
        ĬĞÑĦĹçĢIJŁĮĻ ł ĬĦĮâĽĦĬÌgĬĞÑĦĹĽã

        ſ IĮĨĻĨĞĬĨÑĦ Ğ ĬĨĽĻ ĢIJĮĻĞĨĮĨĮĬ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽf
        ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ ł äā

        ſ IĻĦĹĞĻĦ ĻĨĦ IJĠÏĦĢĻĽ
        ÌIJĹ Ĩ ĨĮ ĹĞĮĬĦâĬĞÑĦĹçĢIJŁĮĻãf

            ſ IÌ ĨĻďĽ ĻĨĦ ÌĨĹĽĻ ĬĞÑĦĹd
            ſ ĻĨĦ ĴĹĦĻĨIJŁĽ ĬĞÑĦĹ IJĠÏĦĢĻ ĨĽ ĻĨĦ ĨĮĴŁĻ ĬĞÑĦĹ
            ĨÌ Ĩ łł ſf
                ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦĻ ł ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ
                ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦÑĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝā

            ſ AĬĬ ĬĞÑĦĹĽ ĦÑĢĦĴĻ ÌIJĹ ĻĨĦ ÌĨĹĽĻ ĞĮĤ ĻĨĦ ĬĞĽĻ
            ĦĬĨÌ Ĩ l· ĬĞÑĦĹçĢIJŁĮĻ h ĝf
                ĽĦĬÌgĬĞÑĦĹĽäĨāgĴĹĦĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝā
                ĽĦĬÌgĬĞÑĦĹĽäĨāgĮĦÑĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝā
```

```
        # TÎĦ ĬĞĽĻ ĬĞÑĦĹ h ĻÎĦ ĮĦŇĻ ĮĠÏĦĢĻ ÎĽ ĻÎĦ ĬĮĽĽ
        # AĬĽĮĮ ĬĦĻďĽ ĽĞŁĦ ĞĽÎÂĦ ĻÎĦ ĹĦÌĦĹĦĮĢĦ ĻĮĮ ĻÎĦ ĬĞĽĻ ĮĠÏĦĢĻ
        # ŃÎĮĮĽĦ ĮŁĻĴŁĻ ÎĽ ĻÎĦ ĬĮÂĦĬďĽ ĮŁĻĴŁĻ
        ĦĬĽĦf
            ĽĦĬÌgĬĞÑĦĹĽäÎāgĴĹĦŁ ł ĽĦĬÌgĬĞÑĦĹĽäÎhĝā
            ĽĦĬÌgĬĞÑĦĹĽäÎāgĮĦŇĻ ł ĽĦĬÌgĬĮĮĽĽ
            ĽĦĬÌgĮŁĻĴŁĻçĬĞÑĦĹçĞĢĻÎŁĞĻÎĮĮ ł ĽĦĬÌgĬĞÑĦĹĽäÎā

        # IÌ ĬĞÑĦĹ ĢĮĮĻĞÎĮĽ ĞĮ ĞĻĻĹÎĠŁĻĦ ĢĞĬĬĦÂ ðŃĦÎÍÎĻĽðd
        # ÎĻďĽ Ğ ĻĹĞÎĮĞĠĬĦ ĬĞÑĦĹ h
        # ĞÂÂ ÎĻ ĻĮĮ ĻÎĦ ĬÎĽĻ ĮÌ ĻĹĞÎĮĞĠĬĦ ĬĞÑĦĹĽ
        # WĦ ÂĮĮďĻ ĮĦĦÂ ĻĮĮ ĢÎĦĢĪ ÌĮĮĹ ĠÎĞĽĦĹ h
        # ĢÎĦĢĪÎĮÍ ÌĮĮĹ ŃĦÎÍÎĻĽ ÎĽ ĦĮĮŁÍÎ
        ÎÌ ÎĞĽĞĻĻĹâĽĦĬÌgĬĞÑĦĹĽäÎād dŃĦÎÍÎĻĽďãf
            ĽĦĬÌgĻĹĞÎĮĞĠĬĦçĬĞÑĦĹĽgĞĴĴĦĮÂâĽĦĬÌgĬĞÑĦĹĽäÎāã

        # UĴÂĞĻĦ ĬĮĮĽĽ ĮĠÏĦĢĻ ŃÎĻÎ ĻĹĞÎĮĞĠĬĦ ĬĞÑĦĹĽ
        ĽĦĬÌgĬĮĮĽĽgĹĦĬĦĬĠĦĹçĻĹĞÎĮĞĠĬĦçĬĞÑĦĹĽâ
            ĽĦĬÌgĻĹĞÎĮĞĠĬĦçĬĞÑĦĹĽ
        ã

    # IÌ ĮŁĻĴŁĻ ĞĢĻÎŁĞĻÎĮĮ ÎĽ SĮĴÌĻĬĞÑ ĞĮÂ
    # ĬĮĮĽĽ ÌŁĮĢĻÎĮĮ ÎĽ CĞĻĦĹĮĮĹÎĢĞĬ CĹĮĮĽĽhEĮĻĹĮĮĴÑ
    # ĢĹĦĞĻĦ ĞĮ ĮĠÏĦĢĻ ĮĮÌ ĢĮĮĬĠÎĮĦÂ ĞĢĻÎŁĞĻÎĮĮ
    # ĞĮÂ ĬĮĮĽĽ ÌŁĮĢĻÎĮĮ ĢĮĮĻĞÎĮÎĮÍ
    # ÌĞĽĻĦĹ ÍĹĞÂÎĦĮĻ ĢĞĬĢŁĬĞĻÎĮĮ
    ÎÌ ÎĽÎĮĽĻĞĮĢĦâĽĦĬÌgĬĞÑĦĹĽähĝād AĢĻÎŁĞĻÎĮĮçSĮĴÌĻĬĞÑã ĞĮÂ ċ
       ÎĽÎĮĽĻĞĮĢĦâĽĦĬÌgĬĮĮĽĽd LĮĮĽĽçCĞĻĦĹĮĮĹÎĢĞĬCĹĮĮĽĽĦĮĻĹĮĮĴÑãf
        # CĹĦĞĻĦ ĞĮ ĮĠÏĦĢĻ ĮĮÌ ĢĮĮĬĠÎĮĦÂ ĞĢĻÎŁĞĻÎĮĮ
        # ĞĮÂ ĬĮĮĽĽ ÌŁĮĢĻÎĮĮĮĽ
        ĽĦĬÌgĽĮĮÌĻĬĞÑçĢĬĞĽĽÎÌÎĦĹçĮŁĻĴŁĻ ł ċ
            AĢĻÎŁĞĻÎĮĮçSĮĴÌĻĬĞÑçLĮĮĽĽçCĞĻĦĹĮĮĹÎĢĞĬCĹĮĮĽĽĦĮĻĹĮĮĴÑâã

# TĹĞÎĮ ĻÎĦ ĬĮÂĦĬ
ÂĦÌ ĻĹĞÎĮâĽĦĬÌd Xd Ñd ấd ĦÂĮĮĢÎĽłĝd ÂĹÎĮĻçĦŁĦĹÑłĝd
          ŁĞĬÎÂĞĻÎĮĮçÂĞĻĞłNĮĮĮĦãf

    # IĮÎĻÎĞĬÎÑĦ ĞĢĢŁĹĞĢÑ ĮĠÏĦĢĻ
    ĽĦĬÌgĞĢĢŁĹĞĢÑgÎĮÎĻâÑã

    # MĞÎĮ ĻĹĞÎĮÎĮÍ ĬĮĮĮÂ
    ÌĮĮĹ ĦÂĮĮĢÎ ÎĮ ĹĞĮÍĦâĝd ĦÂĮĮĢÎĽĸĝãf

        # PĦĹÌĮĮĹÌ ĻÎĦ ÌĮĮĹŃĞĹÂ ÂĞĽĽ
        ĮŁĻĴŁĻ ł ĽĦĬÌgÌĮĮĹŃĞĹÂâXd ĻĹĞÎĮÎĮÍłTĹŁĦã
```

```
    # CĞĬĢŁĬĞĻĦ ĬIJĹĽ
    ĤĞĻĞçĬIJĹĽd ĹĦÍŁĬĞĹĨŅĞĻĨIJIçĬIJĹĽ ł ċ
        ĽĦĬÌgĬIJĹĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd Ñd
                                ĬIĢĬŁĤĦçĹĦÍŁĬĞĹĨŅĞĻĨIJIłTĹŁĦã
    ĬIJĹĽ ł ĤĞĻĞçĬIJĹĽ κ ĹĦÍŁĬĞĹĨŅĞĻĨIJIçĬIJĹĽ

    # GĦĻ ĴĹĦĤĨĢĻĨIJIĻ ĞIĤ ĢĞĬĢŁĬĞĻĦ ĞI ĞĢĢŁĹĞĢÑ
    ĴĹĦĤĨĢĻĨIJIĻ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨŁĞĻĨIJIgĴĹĦĤĨĢĻĨIJIĻâ
                        IJŁĻĴŁĻã
    ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦĤĨĢĻĨIJIĻd Ñã

    # PĦĹÌIJĹİ ĠĞĢĪŃĞĹĤ ĴĞĹĽ
    ĽĦĬÌgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd Ñã

    # OĴĻĨİĨŅĦ âŁĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽã
    ĽĦĬÌgIJĴĻĨİĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâã
    ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽf
        ĽĦĬÌgIJĴĻĨİĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĬĞÑĦĹã
    ĽĦĬÌgIJĴĻĨİĨŅĦĹgĴIJĹĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâã

    # PĹĨĮĻ Ğ ĽŁİĬĞĹÑ
    ĨÌ ĮIJĻ ĦĴIJĢÎ ķ ĴĹĨĮçĦĹĦĹÑf
        ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåd ď κ
              ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĹĽf ăĬIJĹĽfgġÌå âď κ
              ÌďĤĞĻĞçĬIJĹĽf ăĤĞĻĞçĬIJĹĽfgġÌåd ď κ
              ÌďĹĦÍçĬIJĹĽf ăĹĦÍŁĬĞĹĨŅĞĻĨIJIçĬIJĹĽfgġÌåãd ď κ
              ÌďĬĹf ăĽĦĬÌgIJĴĻĨİĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦåďã

# IÌ ĻĤĦĹĦ ĨĽ ĻĤĦ ŁĞĬĨĤĞĻĨIJĮ ĤĞĻĞ
ĨÌ ŁĞĬĨĤĞĻĨIJIçĤĞĻĞ ĨĽ ĮIJĻ NIJĮĦf

    # FIJĹ ĠĦĻĻĦĹ ĹĦĞĤĞĠĨĬĨĻÑ
    XçŁĞĬd ÑçŁĞĬ ł ŁĞĬĨĤĞĻĨIJIçĤĞĻĞ

    # PĦĹÌIJĹİ ĻĤĦ ÌIJĹŃĞĹĤ ĴĞĹĽ
    IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹĤâXçŁĞĬd ĻĹĞĨĮĨĮĬłFĞĬĽĦã

    # CĞĬĢŁĬĞĻĦ ĻĤĦ ĬIJĹĽ
    ĬIJĹĽ ł ĽĦĬÌgĬIJĹĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ÑçŁĞĬã

    # GĦĻ ĴĹĦĤĨĢĻĨIJIĻ ĞIĤ ĢĞĬĢŁĬĞĻĦ ĞI ĞĢĢŁĹĞĢÑ
    ĴĹĦĤĨĢĻĨIJIĻ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨŁĞĻĨIJIgĴĹĦĤĨĢĻĨIJIĻâ
                        IJŁĻĴŁĻã
    ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦĤĨĢĻĨIJIĻd ÑçŁĞĬã
```

```
        # PĹĨĮĻ Ğ ĽŁİİĞĹÑ
        ĴĹĨĮĻâÌďĽĞĬĨĤĞĻĨIJĮd ď κ
              ÌďĞ̣Ğ̣f ăĞ̣Ğ̣ŁĹĞ̣Ñfgġìåd ď κ
              Ìďĭ̇IJĽĽf ăĬIJĽĽfgġìåďã

# PĦĹÌIJĹİĽ ÌIJĹŃĞĹĤ ĴĞĽĽ
ÎĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd Xd Ļ̣ĹĞĨĮĨĮÍãf

    # CĞĬĬ ÌIJĹŃĞĹĤ İĦĻÎIJĤ IJĮ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
    # ĻÎĨĽ ŃĨĬĬ ĽĦĻ ĻÎĦ IJŁĻĴŁĻ ĴĹIJĴĦĹĻÑ ĻÎĞĻ
    # ĻÎĦ ÌĨĹĽĻ ĬĞÑĦĹ ĨĮ ðĴĹĦĻð IJĠÏĦĢĻ ĨĽ ĦŃĴĦĢĻĨĮÍ
    ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹgÌIJĹŃĞĹĤâXd Ļ̣ĹĞĨĮĨĮÍã

    # CĞĬĬ ÌIJĹŃĞĹĤ İĦĻÎIJĤ IJÌ ĦĻĦĹÑ IJĠÏĦĢĻ ĨĮ Ğ ĢÎĞĨĮ
    # PĞĽĽ IJŁĻĴŁĻ IJÌ ĻÎĦ ĴĹĦĻĨIJŁĽ IJĠÏĦĢĻ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
    ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĬĞÑĦĹĽf
        ĬĞÑĦĹgÌIJĹŃĞĹĤâĬĞÑĦĹgĴĹĦĻgIJŁĻĴŁĻd Ļ̣ĹĞĨĮĨĮÍã

    # ðĬĞÑĦĹð ĨĽ ĮIJŃ ĻÎĦ ĬĞĽĻ IJĠÏĦĢĻ ÌĹIJİ ĻÎĦ ĬĨĽĻd
    # ĹĦĻŁĹĮ ĨĻĽ IJŁĻĴŁĻ
    ĹĦĻŁĹĮ ĬĞÑĦĹgIJŁĻĴŁĻ

# PĦĹÌIJĹİĽ ĠĞĢĪŃĞĹĤ ĴĞĽĽ
ÎĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd IJŁĻĴŁĻd Ñãf

    # IÌ ĽIJÌĻİĞÑ ĢĬĞĽĽĨÌÌĨĦĹ
    ĨÌ ĽĦĬÌgĽIJÌĻİĞÑçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻĴŁĻ ĨĽ ĮIJĻ NIJĮĦf
        # FĨĹĽĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻÎIJĤ
        # IJĮ ĻÎĦ ĢIJİĠĨĮĦĤ ĞĢĻĨĻĞĻĨIJĮwĬIJĽĽ
        # ĻÎĨĽ ŃĨĬĬ ĽĦĻ ĤĨĮĴŁĻĽ ĴĹIJĴĦĹĻÑ
        ĽĦĬÌgĽIJÌĻİĞÑçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻĴŁĻgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd Ñã

        # SĨĮĢĦ ŃĦďĬĬ ĮIJĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻÎIJĤ IJÌ ĻÎĦ ĬĞĽĻ ĬĞÑĦĹ
        # ŃÎĨĢÎ ĨĽ SIJÌĻİĞÑ ĞĢĻĨĻĞĻĨIJĮ
        # ĞĽ ŃĦ ŁĽĦĤ ĢIJİĠĨĮĦĤ ĞĢĻĨĻĞĻĨIJĮwĬIJĽĽ
        # IJĠÏĦĢĻd ĬĦĻďĽ ĽĦĻ ĤĨĮĴŁĻĽ ĨĮ ĻÎĨĽ IJĠÏĦĢĻ
        ĽĦĬÌgĬĞÑĦĹĽähĝāgĤĨĮĴŁĻĽ ł ċ
            ĽĦĬÌgĽIJÌĻİĞÑçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻĴŁĻgĤĨĮĴŁĻĽ

        # CĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻÎIJĤ ÍIJĨĮÍ ĻÎĹIJŁÍÎ
        # ĞĬĬ ĻÎĦ IJĠÏĦĢĻĽ ĠŁĻ ĬĞĽĻ
        # ĨĮ ĹĦĻĦĹĽĦĤ IJĹĤĦĹ ĴĞĽĽĨĮÍ ĤĨĮĴŁĻĽ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
        ÌIJĹ ĬĞÑĦĹ ĨĮ ĹĦĻĦĹĽĦĤâĽĦĬÌgĬĞÑĦĹĽäfhĝāãf
            ĬĞÑĦĹgĠĞĢĪŃĞĹĤâĬĞÑĦĹgĮĦŃĻgĤĨĮĴŁĻĽã

        ĹĦĻŁĹĮ
```

```
        # FĨĹĽĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ ĬĦĻĨIJĤ IJĮ ĻĨĦ ĬIJĽĽ
        # ĻĨĨĽ ŃĨĬĬ ĽĦĻ ĤĨĮĴŁĻĽ ĴĹIJĴĦĹĻÑ ĻĨĞĻ ĻĨĦ ĬĞĽĻ
        # ĬĞÑĦĹ ŃĨĬĬ ĻĹÑ ĻIJ ĞĢĢĦĽĽ ĽĨIJĹĻĬÑ
        ĽĦĬÌgĬIJĽĽgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd Ñã

        # CĞĬĬ ĠĞĢĪŃĞĹĤ ĬĦĻĨIJĤ ÍIJĨĮÍ ĻĨĹIJŁÍĨ ĞĬĬ ĻĨĦ IJĠÏĦĢĻĽ
        # ĨĮ ĹĦŁĦĹĽĦĤ IJĹĤĦĹ ĴĞĽĽĨĮÍ ĤĨĮĴŁĻĽ ĞĽ Ğ ĴĞĹĞĬĦĻĦĹ
        ÌIJĹ ĬĞÑĦĹ ĨĮ ĹĦŁĦĹĽĦĤâĽĦĬÌgĬĞÑĦĹĽãf
            ĬĞÑĦĹgĠĞĢĪŃĞĹĤâĬĞÑĦĹgĮĦŃĻgĤĨĮĴŁĻĽã


# CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞĬĴĬĦĽłĝﬂﬂﬂd ĢĬĞĽĽĦĽłġã
XçĻĦĽĻd ÑçĻĦĽĻ ł ĽĴĨĹĞĬçĤĞĻĞâĽĞĬĴĬĦĽłĝﬂﬂd ĢĬĞĽĽĦĽłġã

# IĮĽĻĞĮĻĨĞĻĦ ĻĨĦ ĬIJĤĦĬ
ĬIJĤĦĬ ł MIJĤĦĬâã

# AĤĤ ĬĞÑĦĹĽ
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâğd ĥĝğd ŃĦĨĹĨĻçĹĦĹłĬĞĹĨŅĦĹçĬğłĥĦĦhġd
                                ĠĨĞĽçĹĦĹłĬĞĹĨŅĦĹçĬğłĥĦĦhġãã
ĬIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçRĦLUâãã
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĹIJĴIJŁĻâﬂgĝãã
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĥĝğd ġãã
ĬIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçSIJÌĻĬĞŃâãã

# SĦĻ ĬIJĽĽd IJĴĻĨĬĬŅĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠÏĦĢĻĽ
ĬIJĤĦĬgĽĦĻâ
    ĬIJĽĽłLIJĽĽçCĞĻĦĬIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâãd
    IJĴĻĨĬĬŅĦĹłOĴĻĨĬĬŅĦĹçAĤĞĬâĬĦĞĹĮĨĮĬçĹĞĻĦłﬂgﬂĥd ĤĦĢĞÑłĥĦhĥãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçCĞĻĦĬIJĹĨĢĞĬâã
ã

# FĨĮĞĬĨŅĦ ĻĨĦ ĬIJĤĦĬ
ĬIJĤĦĬgÌĨĮĞĬĨŅĦâã

# TĹĞĨĮ ĻĨĦ ĬIJĤĦĬ
ĬIJĤĦĬgĻĹĞĨĮâXd Ñd ĽĞĬĨĤĞĻĨIJĮçĤĞĻĞłâXçĻĦĽĻd ÑçĻĦĽĻãd
            ĦĴIJĢĨĽłĝﬂﬂﬂﬂd ĴĹĨĮçĦĽĦĹÑłĝﬂﬂã
```

Supplementary Material: https://nnfs.io/ch18

Chapter code, further resources, and errata for this chapter.

Chapter 19

A Real Dataset

In practice, deep learning tends to involve massive datasets (often terabytes or more in size), and models can take days, weeks, or even months to train. This is why, so far, we've used programmatically-generated datasets to keep things manageable and fast, while we learn the math and other aspects of deep learning. The main objective of this book is to teach how neural networks work, rather than the application of deep learning to various problems. That said, we'll explore a more realistic dataset now, since this will present new challenges to deep learning that we've not yet had to consider.

If you have explored deep learning before reading this book, you have likely become acquainted (and possibly annoyed) with the MNIST dataset, which is a dataset of images of handwritten digits (0 through 9) at a resolution of 28x28 pixels. It's a relatively small dataset and is reasonably easy for models to learn. This dataset became the "hello world" of deep learning, and it was once a benchmark of machine learning algorithms. The problem with this dataset is that it's comically easy to get 99%+ accuracy on the MNIST dataset, and doesn't provide much room for learning how various parameters impact learning. In 2017, however, a company called Zalando released a dataset (https://arxiv.org/abs/1708.07747) called Fashion MNIST (https://github.com/zalandoresearch/fashion-mnist), which is a drop-in replacement for the regular MNIST dataset.

The Fashion MNIST dataset is a collection of 60,000 training samples and 10,000 testing samples of 28x28 images of 10 various clothing items like shoes, boots, shirts, bags, and more. We'll see some examples shortly, but first, we need the actual data. Since the original dataset consists of binary files containing encoded (in a specific format) image data, for this book, we have prepared and are hosting a preprocessed dataset consisting of .png images instead. It is usually wise to use lossless compression for images since lossy compression, like JPEG, affects images by changing their data). These images are also grouped by labels and separated into training and testing groups. The samples are the images of articles of clothing, and the labels are the classifications. Here are the numeric labels and their respective descriptions:

Label	**Description**
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Data preparation

First, we will retrieve the data from the nnfs.io site. Let's define the URL of the dataset, a filename to save it locally to and the folder for the extracted images:

```
URL ł ďÎĻĻĴĽfwwĮĮÌĽgĨIJwĤĞĻĞĽĦĻĽwÌĞĽÎĨIJĮçÌĮĨĽĻçĨÌĞÍĦĽgŅĨĴď
FILE ł ďÌĞĽÎĨIJĮçÌĮĨĽĻçĨÌĞÍĦĽgŅĨĴď
FOLDER ł ďÌĞĽÎĨIJĮçÌĮĨĽĻçĨÌĞÍĦĽď
```

Next, download the compressed data (if the file is absent under the given path) using the *urllib*, a standard Python library:

```
ĨÌĴIJĹĻ IJĽ
ĨÌĴIJĹĻ łĹĬĬĨĠ
ĨÌĴIJĹĻ łĹĬĬĨĠgĹĦĶłĦĽĻ

ĨÌ ĮIJĻ IJĽgĴĞĻÎgĨĽÌĨĬĦâFILEãf
    ĴĹĨĮĻâÌďDIJŃĮĬIJĞĤĨĮÍ ăURLå ĞĮĤ ĽĞĻĨĮÍ ĞĽ ăFILEåggďã
    łĹĬĬĨĠgĹĦĶłĦĽĻgłĹĬĹĦĻĹĨĦĻĦâURLd FILEã
```

From here, we'll unzip the files using another standard Python library called *zipfile*. We'll use the context wrapper (the `ŃĨĻÎ` keyword, which will open and close file for us) to gather the zipped file handler and extract all of the included files using `gĦŇĻĹĞĢĻĞĬĬ` and the given `FOLDER`:

```
ÌĹIJİ ŅĨĴÌĨĬĦ ĨÌĴIJĹĻ ZĨĴFĨĬĦ

ĴĹĨĮĻâďUĮŅĨĴĴĨĮÍ ĨÌĞÍĦĽgggďã
ŃĨĻÎ ZĨĴFĨĬĦâFILEã ĞĽ ŅĨĴçĨÌĞÍĦĽf
    ŅĨĴçĨÌĞÍĦĽgĦŇĻĹĞĢĻĞĬĬâFOLDERã
```

The full code for retrieving the data:

```
ÌĹIJİ ŅĨĴÌĨĬĦ ĨÌĴIJĹĻ ZĨĴFĨĬĦ
ĨÌĴIJĹĻ IJĽ
ĨÌĴIJĹĻ łĹĬĬĨĠ
ĨÌĴIJĹĻ łĹĬĬĨĠgĹĦĶłĦĽĻ
```

```
URL ł ďÎĻĻĴĽfwwĮĮÌĽgĨIJwĤĞĻĞĽĦĻĽwÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽgŅĨĴď
FILE ł ďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽgŅĨĴď
FOLDER ł ďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽď

ĨÌ ĮIJĻ IJĽgĴĞĻÎgĨĽÌĨĬĦâFILEãf
    ĴĹĨĮĻâÌďDIJŃĮĬIJĞĤĨĮÍ ăURLå ĞĮĤ ĽĞĻĨĮÍ ĞĽ ăFILEåggďã
    ŁĹĬĬĬĠgĹĦĶŁĦĽĻgŁĹĬĹĦĻĹĨĦĻĦâURLd FILEã

ĴĹĨĮĻâďUĮŅĨĴĴĨĮÍ ĨİĞÍĦĽgggďã
ŃĨĻÎ ZĨĴFĨĬĦâFILEã ĞĽ ŅĨĴçĨİĞÍĦĽf
    ŅĨĴçĨİĞÍĦĽgĦŇĻĹĞĢĻĞĬĬâFOLDERã

ĴĹĨĮĻâďDIJĮĦidďã
```

Running this:

```
ńńń
DIJŃĮĬIJĞĤĨĮÍ ÎĻĻĴĽfwwĮĮÌĽgĨIJwĤĞĻĞĽĦĻĽwÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽgŅĨĴ ĞĮĤ ĽĞĻĨĮÍ ĞĽ
ÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽgŅĨĴggg
UĮŅĨĴĴĨĮÍ ĨİĞÍĦĽggg
DIJĮĦi
```

You should now have a directory called *fashion_mnist_images*, containing *test* and *train* directories and the data license. Inside of both the *test* and *train* directories, we have ten subdirectories, numbered 0 through 9. These numbers are classifications that correspond to the images within. For example, if we open directory *0*, we can see these are images of shirts with either short sleeves or no sleeves at all. For example:

Fig 19.01: Example t-shirt image from the Fasion MNIST dataset.

Inside directory *7*, we have non-boot shoes, or *sneakers* as the creators of this dataset have classified them. For example:

Fig 19.02: Example sneaker image from the Fasion MNIST dataset.

It's common practice to grayscale (go from 3-channel RGB values per pixel to a single black to white range of 0-255 per pixel) images, though these images are already grayscaled. It is also a common practice to resize images to **normalize** their dimensions, but once again, the Fashion MNIST dataset is prepared so that all the images are already all the same shape (28x28).

Data loading

Next, we have to read these images into Python and associate the image (pixel) data with the respective labels. We can access the directories as follows:

```
ÎİĴIJĹĻ IJĽ

ĬĞĠĦĬĽ ł IJĽgĬÎĽĻĤÎĹâďÌĞĽÎÎIJĮçİĮÎĽĻçÎİĞÍĦĽwĻĹĞÎĮďã
ĴĹÎĮĻâĬĞĠĦĬĽã


ńńń
äďfldd ďĝďd ďğďd ďġďd ďģďd ďĥďd ďħďd ďìďd ďíďd ďîďã
```

Since the subdirectory names are the labels themselves, we can reference individual samples for each class by looking through the files in each numbered subdirectory:

```
ÌÎĬĦĽ ł IJĽgĬÎĽĻĤÎĹâďÌĞĽÎÎIJĮçİĮÎĽĻçÎİĞÍĦĽwĻĹĞÎĮwfldã
ĴĹÎĮĻâÌÎĬĦĽäfĝflãã
ĴĹÎĮĻâĬĦĮâÌÎĬĦĽãã
```

```
ńńń
äďflflflgĴĮÍďd ďflflflĝgĴĮÍďd ďflflflğgĴĮÍďd ďflflflġgĴĮÍďd ďflflflĝgĴĮÍďd ďflflflĥgĴĮÍďd
ďflflflħgĴĮÍďd ďflflflìgĴĮÍďd ďflflflígĴĮÍďd ďflflflîgĴĮÍďã
ħflflfl
```

As you can see, we have 6,000 samples of class 0. In total, we have 60,000 samples -- 6,000 per classification. Meaning our dataset is also already **balanced**; each class occurs with the same frequency. If a dataset is not already balanced, the neural network will likely become biased to predict the class containing the most images. This is because neural networks fundamentally seek out the steepest and quickest gradient descent to decrease loss, which might lead to a local minimum making the model unable to find the global loss minimum. We have a total of 10 classes here, so a random prediction, with a balanced dataset, would have an accuracy of about 10%.

Imagine, however, if the balance of classes in the dataset was 64% for class 0 and 4% for 1 through 9. The neural network might very quickly learn to always predict class 0. Though the model would rapidly decrease loss initially, it would likely remain stuck predicting class 0 with an accuracy closer to 64%. In such a case, we're better off trimming away samples from the high-frequency classes so that we have the same number of samples per class.

Another option is to use class weights, weighting classes that occur more often with a fraction of 1 when accounting for loss. Though we have never seen this work well in practice. With image data, another option would be to augment the samples through actions like cropping, rotation, and maybe flipping horizontally or vertically. Before applying such transformations, ensure they will generate valid samples that fit your objectives. Luckily for us, we don't need to worry about that since the Fashion MNIST data are indeed perfectly balanced. We'll now explore our data by looking at individual samples. To handle image data, we're going to make use of the Python package containing OpenCV, under the **cv2** library, which you can install with pip:

```
ĴĨĴ ĨĮĽĻĞĬĬ ĮĴĦĮĢĿĥĴÑĻĨĮĴĮ
```

And to load the image data:

```
ĨĬĴĮĴĹĻ ĢĿğ

ĨĬĞÍĦçĤĞĻĞ ł ĢĿğgĨĬĹĦĞĤâďÌĞĽĨĨĮçĬĮĨĽĻçĨĬĞÍĦĽwĻĹĞĨĮwìwflflflğgĴĮÍďd
                                ĢĿğgIMREADçUNCHANGEDã
ĴĹĨĮĻâĨĬĞÍĦçĤĞĻĞã
```

We read in images with `ĢĿğgĨĬĹĦĞĤâã`, where the first parameter is the path to the image. The `ĢĿğgIMREADçUNCHANGED` argument notifies the `ĢĿğ` package that we intend to read in these images in the same format as they were saved (grayscale in this case). By default, OpenCV will convert these images to use all 3 color channels, even though this is a grayscale image. As a result, we have a 2D array of numbers — grayscale pixel values. If we formatted this otherwise

messy array before printing with the following line of the code, which will inform NumPy, since the loaded image is a NumPy array object, to print more characters in a line:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĮĴgĽĦĻçĴĹĨĮĻIJĴĻĨIJĮĽâĬĨĮĦŃĨĤĻĬłğflflã
```

We'd still likely be able to identify the subject:

```
ää fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  ġ  fl  fl  fl  ġî  ĝġĥ ĝíġ ĝĥfl ĥî  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  ìí  ğĥĥ ğğfl ğĝğ ğĝî ğĥĥ ğġħ ĝîĝ ĝĥĥ  íì  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  ġ  fl  fl  ĥì ğflħ ğĝĥ ğflġ ĝîĝ ğflġ ğĝĝ ğĝħ ĝĝì ğğfl ğĝĝ  ĝĥ  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  ĝ  fl  fl  fl  ĥí ğĝĝ ğğfl ğĝfl ĝîî ğflî ğĝí ğĝí ğĝì ğflí ğflfl ğĝĥ  ĥħ  flā
 ä fl  fl  fl  fl  fl  ĝ  ğ  fl  fl  ġ  fl  fl  fl  fl ĝġĥ ĝĝġ ğflì ĝîî ĝíì ğflġ ğĝfl ğĝħ ğĝì ğĝĥ ğĝĥ ğflħ ğĝĥ ĝġfl  flā
 ä fl  fl  fl  fl  ĝ  ğ  ġ  fl  fl  fl  ġ ĝflĥ ğĝĥ ĝflĝ ĝîfl ĝflĝ ğĝfl ğĝġ ğĝġ ğĝĥ ğĝĥ ğĝğ ğĝğ ğflí ğflĥ ğflì ĝĝí  flā
 ä ĝ  ĥ  ì  fl  fl  fl  fl  fl ĥğ ĝħğ ĝĝì ĝíî ĝìġ ĝĥì ĝíì ĝîí ğflğ ğĝì ğğfl ğğġ ğğġ ğğğ ğĝì ğĝĝ ğĝì ğflĝ ğġì  ħĥā
 ä fl  fl  fl  fl  fl  fl ğĝ  ìğ ĝíĥ ĝíî ĝìĝ ĝìĝ ĝíĥ ğflġ ğflfl ğflì ğflí ğflî ğĝġ ğĝî ğğğ ğğğ ğğġ ğĝĥ ğĝí ĝĝĝ ğğğ ĝġíā
 ä fl  ìfl ĝĝġ ĝĝî ĝĝĥ ĝĥî ĝìî ĝìĥ ĝìğ ĝìĥ ĝíĥ ĝîĥ ĝîî ğflħ ğflĝ ğĝfl ğĝğ ğĝġ ğĝĥ ğĝí ğĝî ğĝì ğĝğ ğflì ğflí ğflfl ĝîí ĝìġā
 ä fl ĝğğ ĝĥí ĝíġ ĝîġ ĝîğ ĝîġ ĝîħ ğflġ ğflî ğĝĝ ğĝĝ ğĝĥ ğĝí ğĝĝ ğğğ ğğħ ğĝì ğĝì ğĝħ ğĝħ ğĝġ ğğğ ğĝħ ğĝĝ ğflí ğĝħ ĝíĥā
 ä ğĝ  fl  fl  ĝğ  ġí  íğ ĝğġ ĝĥğ ĝìfl ĝíġ ĝíĥ ğĝĝ ğĝĥ ğġğ ğġġ ğĝì ğġğ ğġğ ğĝfl ğĝfl ğĝí ğĝħ ğğğ ğflî ğflfl ĝîġ ĝíĥ ĝflħā
 ä ğħ  ġì  ĥġ  ĝí  ĥ  fl  fl  fl  fl  fl  fl  fl  fl  fl  ğ  ġ  ħ  î  î  í  î  ħ  ħ  ġ  ğ  fl  fl  flā
 ä fl  ĝfl  ğì  ġĥ  ĥĥ  ĥî  ĥì  ĥfl  ġġ  ĥğ  ĥí  ħğ  ħĥ  ĥħ  ĥġ  ĥì  ĥî  ħğ  ħfl  ħġ  ħí  ħì  ħħ  ìğ  ìì  ìġ  ħĥ  ġîā
 ä fl  fl  fl  fl  ġ  î  ĝí  ğġ  ğħ  ğĥ  ğğ  ğĥ  ğî  ġì  ġí  ġì  ġî  ġħ  ğî  ġğ  ġġ  ġġ  ğí  ğğ  ğfl  ĝğ  ì  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flā
 ä fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  fl  flāā
```

In this case, it's a sneaker. Rather than formatting the raw values to see what we're looking at this way, we can use Matplotlib to visualize this. For example:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĴĬĻgĨİĽÎIJŃâĨİĞÍĦçĤĞĻĞã
ĴĬĻgĽÎIJŃâã

ńńń
```

Fig 19.03: Sneaker image shown with Matplotlib after loading with Python

We can check another sample:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĨİĞÍĦçĤĞĻĞ ł ĢĿğgĨİĹĦĞĤâďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽwĻĹĞĨĮwǵwflflĝĝgĴĮÍďd
                                ĢĿğgIMREADçUNCHANGEDã
ĴĬĻgĨİĽÎIJŃâĨİĞÍĦçĤĞĻĞã
ĴĬĻgĽÎIJŃâã

ńńń
```

Fig 19.04: Jacket image shown with Matplotlib after loading with Python

It looks like a jacket. If we check our table from before, class 4 is "coat." You might wonder about the strange coloring, but this is just the default from matplotlib not expecting grayscale. We can notify Matplotlib that this is grayscale by specifying a *ĢİĞĴ* (colormap) during the `ĴĬĻgĨİĽÎIJŃâã` call:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĨİĞÍĦçĤĞĻĞ ł ĢĿğgĨİĹĦĞĤâďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽwĻĹĞĨĮwǵwflflĝĝgĴĮÍďd
                                ĢĿğgIMREADçUNCHANGEDã
ĴĬĻgĨİĽÎIJŃâĨİĞÍĦçĤĞĻĞd ĢİĞĴłďÍĹĞÑďã
ĴĬĻgĽÎIJŃâã

ńńń
```

Fig 19.05: Grayscaled jacket image

Now we can iterate over all of the samples, load them, and put them into the input data (**X**) and targets (**y**) lists. First, we scan the train folder, which, as noted before, contains folders named from 0 to 9, which also act as sample labels. We iterate through these folders and the images inside them, appending them to a list variable (named X*)* along with their respective labels to another list variable (named Ñ*)*, forming our samples and ground-truth, or target labels:

```
# SQĞĮ ĞĬĬ ĻÎĦ ÂĨĹĦQĻIJĹĨĦĽ ĞĮÂ QĹĦĞĻĦ Ğ ĬĨĽĻ IJÌ ĬĞĠĦĬĽ
ĬĞĠĦĬĽ ł IJĽgĬĨĽĻÂĨĹâďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽwĻĹĞĨĮďã

# CĹĦĞĻĦ ĬĨĽĻĽ ÌIJĹ ĽĞİĴĬĦĽ ĞĮÂ ĬĞĠĦĬĽ
X ł äã
Ñ ł äã

# FIJĹ ĦĞQÎ ĬĞĠĦĬ ÌIJĬÂĦĹ
ÌIJĹ ĬĞĠĦĬ ĨĮ ĬĞĠĦĬĽf
    # AĮÂ ÌIJĹ ĦĞQÎ ĨİĞÍĦ ĨĮ ÍĨL·ĦĮ ÌIJĬÂĦĹ
    ÌIJĹ ÌĨĬĦ ĨĮ IJĽgĬĨĽĻÂĨĹâIJĽgĴĞĻÎgÏIJĨĮâ
                    ďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽďd ďĻĹĞĨĮďd ĬĞĠĦĬ
    ããf
        # RĦĞÂ ĻÎĦ ĨİĞÍĦ
        ĨİĞÍĦ ł QL·ğgĨİĹĦĞÂâIJĽgĴĞĻÎgÏIJĨĮâ
                    ďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽwĻĹĞĨĮďd ĬĞĠĦĬd ÌĨĬĦ
                ãd QL·ğgIMREADçUNCHANGEDã

        # AĮÂ ĞĴĴĦĮÂ ĨĻ ĞĮÂ Ğ ĬĞĠĦĬ ĻIJ ĻÎĦ ĬĨĽĻĽ
        XgĞĴĴĦĮÂâĨİĞÍĦã
        ÑgĞĴĴĦĮÂâĬĞĠĦĬã
```

We need to do this operation on both the testing and training data. Again, they are already nicely split up for us. Many times, you will need to separate your data into train and test groups on your own. We'll convert the above code into a function to prevent duplicating the code for training and testing directories. This function will take a dataset type as a parameter: train or test, along with the path where those datasets are located:

```
ĩİĴIJĹĻ ĮłİĴÑ ĞĽ ĮĴ
ĩİĴIJĹĻ ĢĿğ
ĩİĴIJĹĻ IJĽ


ſ LIJĞĤĽ Ğ MNIST ĤĞĻĞĽĦĻ
ĤĦÌ ĬIJĞĤçİĮĨĬĽĻçĤĞĻĞĽĦĻâĤĞĻĞĽĦĻd ĴĞĻÎãf

    ſ SĢĞĮ ĞĬĬ ĻÎĦ ĤĨĹĦĢĻIJĹĨĦĽ ĞĮĤ ĢĹĦĞĻĦ Ğ ĬĨĬĽĻ IJÌ ĬĞĠĦĬĽ
    ĬĞĠĦĬĽ ł IJĽgĬĨĬĽĻĤĨĹâIJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ĤĞĻĞĽĦĻãã

    ſ CĹĦĞĻĦ ĬĨĬĽĻĽ ÌIJĹ ĽĞİĴĬĦĽ ĞĮĤ ĬĞĠĦĬĽ
    X ł äã
    Ñ ł äã

    ſ FIJĹ ĦĞĢÎ ĬĞĠĦĬ ÌIJĬĤĦĹ
    ÌIJĹ ĬĞĠĦĬ ĨĮ ĬĞĠĦĬĽf
        ſ AĮĤ ÌIJĹ ĦĞĢÎ ĨİĞÍĦ ĨĮ ÍĨĿĦĮ ÌIJĬĤĦĹ
        ÌIJĹ ÌĨĬĦ ĨĮ IJĽgĬĨĬĽĻĤĨĹâIJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ĤĞĻĞĽĦĻd ĬĞĠĦĬããf
            ſ RĦĞĤ ĻÎĦ ĨİĞÍĦ
            ĨİĞÍĦ ł ĢĿğgĨİĹĦĞĤâIJĽgĴĞĻÎgÏIJĨĮâ
                        ĴĞĻÎd ĤĞĻĞĽĦĻd ĬĞĠĦĬd ÌĨĬĦ
                    ãd ĢĿğgIMREADçUNCHANGEDã

            ſ AĮĤ ĞĴĴĦĮĤ ĨĻ ĞĮĤ Ğ ĬĞĠĦĬ ĻIJ ĻÎĦ ĬĨĬĽĻĽ
            XgĞĴĴĦĮĤâĨİĞÍĦã
            ÑgĞĴĴĦĮĤâĬĞĠĦĬã

    ſ CIJĮĿĦĹĻ ĻÎĦ ĤĞĻĞ ĻIJ ĴĹIJĴĦĹ ĮłİĴÑ ĞĹĹĞÑĽ ĞĮĤ ĹĦĻłĹĮ
    ĹĦĻłĹĮ ĮĴgĞĹĹĞÑâXãd ĮĴgĞĹĹĞÑâÑãgĞĽĻÑĴĦâďłĨĮĻíďã
```

Since X has been defined as a list, and we are adding images represented as NumPy arrays to this list, we'll call ĮĴgĞĹĹĞÑâã on X at the end to transform it from a list into a proper NumPy array. We will do the same with the labels (Ñ) since they are a list of numbers and additionally inform NumPy that our labels are integer (not float) values.

Then we can write a function that will create and return our train and test data:

```
ſ MNIST ĤĞĻĞĽĦĻ âĻĹĞĨĮ κ ĻĦĽĻã
ĤĦÌ ĢĹĦĞĻĦçĤĞĻĞçİĮĨĬĽĻâĴĞĻÎãf

    ſ LIJĞĤ ĠIJĻÎ ĽĦĻĽ ĽĦĴĞĹĞĻĦĬÑ
    Xd Ñ ł ĬIJĞĤçİĮĨĬĽĻçĤĞĻĞĽĦĻâďĻĹĞĨĮďd ĴĞĻÎã
    XçĻĦĽĻd ÑçĻĦĽĻ ł ĬIJĞĤçİĮĨĬĽĻçĤĞĻĞĽĦĻâďĻĦĽĻďd ĴĞĻÎã

    ſ AĮĤ ĹĦĻłĹĮ ĞĬĬ ĻÎĦ ĤĞĻĞ
    ĹĦĻłĹĮ Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ
```

Code up to this point for our new data:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĢĽğ
ĨİĴIJĹĻ IJĽ


ſ LIJĞĤĽ Ğ MNIST ĤĞĻĞĽĦĻ
ĤĦÌ ĬIJĞĤçİĮĨĽĻçĤĞĻĞĽĦĻâĤĞĻĞĽĦĻd ĴĞĻÎãf

    ſ SĢĞĮ ĞĬĬ ĻÎĦ ĤĨĹĦĢĻIJĹĨĦĽ ĞĮĤ ĢĹĦĞĻĦ Ğ ĬĨĽĻ IJÌ ĬĞĠĦĬĽ
    ĬĞĠĦĬĽ ł IJĽgĬĨĽĻĤĨĹâIJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ĤĞĻĞĽĦĻãã

    ſ CĹĦĞĻĦ ĬĨĽĻĽ ÌIJĹ ĽĞİĴĬĦĽ ĞĮĤ ĬĞĠĦĬĽ
    X ł äã
    Ñ ł äã

    ſ FIJĹ ĦĞĢÎ ĬĞĠĦĬ ÌIJĬĤĦĹ
    ÌIJĹ ĬĞĠĦĬ ĨĮ ĬĞĠĦĬĽf
        ſ AĮĤ ÌIJĹ ĦĞĢÎ ĨİĞÍĦ ĨĮ ÍĨL·ĦĮ ÌIJĬĤĦĹ
        ÌIJĹ ÌĨĬĦ ĨĮ IJĽgĬĨĽĻĤĨĹâIJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ĤĞĻĞĽĦĻd ĬĞĠĦĬããf
            ſ RĦĞĤ ĻÎĦ ĨİĞÍĦ
            ĨİĞÍĦ ł ĢL·ğgĨİĹĦĞĤâIJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ĤĞĻĞĽĦĻd ĬĞĠĦĬd ÌĨĬĦãd
ĢL·ğgIMREADçUNCHANGEDã

            ſ AĮĤ ĞĴĴĦĮĤ ĨĻ ĞĮĤ Ğ ĬĞĠĦĬ ĻIJ ĻÎĦ ĬĨĽĻĽ
            XgĞĴĴĦĮĤâĨİĞÍĦã
            ÑgĞĴĴĦĮĤâĬĞĠĦĬã

    ſ CIJĮL·ĦĹĻ ĻÎĦ ĤĞĻĞ ĻIJ ĴĹIJĴĦĹ ĮŁİĴÑ ĞĹĹĞÑĽ ĞĮĤ ĹĦĻŁĹĮ
    ĹĦĻŁĹĮ ĮĴgĞĹĹĞÑâXãd ĮĴgĞĹĹĞÑâÑãgĞĽĻÑĴĦâďŁĨĮĻíďã


ſ MNIST ĤĞĻĞĽĦĻ âĻĹĞĨĮ κ ĻĦĽĻã
ĤĦÌ ĢĹĦĞĻĦçĤĞĻĞçİĮĨĽĻâĴĞĻÎãf

    ſ LIJĞĤ ĠIJĻÎ ĽĦĻĽ ĽĦĴĞĹĞĻĦĬÑ
    Xd Ñ ł ĬIJĞĤçİĮĨĽĻçĤĞĻĞĽĦĻâďĻĹĞĨĮďd ĴĞĻÎã
    XçĻĦĽĻd ÑçĻĦĽĻ ł ĬIJĞĤçİĮĨĽĻçĤĞĻĞĽĦĻâďĻĦĽĻďd ĴĞĻÎã

    ſ AĮĤ ĹĦĻŁĹĮ ĞĬĬ ĻÎĦ ĤĞĻĞ
    ĹĦĻŁĹĮ Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ
```

Thanks to this function, we can load in our data by doing:

```
ſ CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ ł ĢĹĦĞĻĦçĤĞĻĞçİĮĨĽĻâďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽďã
```

Data preprocessing

Next, we will **scale** the data (not the images, but the data representing them, the numbers). Neural networks tend to work best with data in the range of either 0 to 1 or -1 to 1. Here, the image data are within the range 0 to 255. We have a decision to make with *how* to scale these data. Usually, this process will be some experimentation and trial and error. For example, we could scale images to be between the range of -1 and 1 by taking each pixel value, subtracting half the maximum of all pixel values (i.e., 255/2 = 127.5), then dividing by this same half to produce a range bounded by -1 and 1. We could also scale our data between 0 and 1 by simply dividing it by 255 (the maximum value). To start, we opt to scale between -1 and 1. Before we do that, we have to change the datatype of the NumPy array, which is currently `łÕĮĻí` (unsigned integer, holds integer values in the range of 0 to 255). If we don't do this, NumPy will convert it to a `ÌĬIJĞĻħǵ` data type while our intention is to use `ÌĬIJĞĻġğ`, a 32-bit float value. This can be achieved by calling `gĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã` on a NumPy array object. We will leave the labels untouched:

```
ſi CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ ł ĢĹĦĞĻĦçĤĞĻĞçĮĮĬĽĻâďÌĞĽÎĬIJĮçĮĮĬĽĻçĬĮĞÍĦĽďã

ſi SĢĞĬĦ ÌĦĞĻŁĹĦĽ
X ł âXgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã h ĝğìgĥã w ĝğìgĥ
XçĻĦĽĻ ł âXçĻĦĽĻgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã h ĝğìgĥã w ĝğìgĥ
```

Ensure that you scale both training and testing data using identical methods. Later, when making predictions, you will also need to scale the input data for inference. It can be easy to forget to scale your data in these different places. You also want to be sure that any preprocessing, like scaling, is informed only by your training dataset. In this example, we knew the minimum (min) and maximum (max) values would be 0 and 255, and performed linear scaling. You will often need to first query your dataset for min and max values for use in scaling. You may also use other methods to scale if your dataset has extreme outliers, as min/max may not work well. If this is the case, you might use some combination of the average value and standard deviation to create your scaling method.

A common mistake when scaling is to allow the testing dataset to inform transformations made to the training dataset. There is only one exception to this rule, which is when the data is being scaled linearly, for example, by the mentioned division by a constant number. Any non-linear scaling function could possibly leak information from testing or validation data into training data. Any preprocessing rules should be derived without knowledge of the testing dataset, but

then applied to the testing set. For example, your entire dataset might have a min value of 0 and a max of 125, while the training dataset only has a min of 0 and a max of 100. You will still use the 100 value when scaling your testing dataset. This means that your testing dataset might not fit neatly between the bounds of -1 and 1 after scaling, but this usually should not be a problem. In the case of a bigger difference, you can additionally scale the data linearly by dividing them by some number.

Back to our data, let's check that our data have been scaled:

```
ĴĹĨĮĻâXgİĨĮâãd XgİĞŇâãã


ńńń
hĝgfl ĝgfl
```

Next, we check the shape of our input data:

```
ĴĹĨĮĻâXgĽÎĞĴĦã


ńńń
âħflflflfld ǧíd ǧíã
```

Our Dense layers work on batches of 1-dimensional vectors. They cannot operate on images shaped as a 28x28, 2-dimensional array. We need to take these 28x28 images and **flatten** them, which means to take every row of an image array and append it to the first row of that array, transforming the 2-dimensional (2D) array of an image into a 1-dimensional (1D) array (i.e., vector), or in other words, we could see this as unwrapping numbers in a 2D array to a list-like form. There are neural network models called convolutional neural networks that will allow you to pass 2D image data "as is," but a dense neural network like we have here expects samples that are 1D. Even in convolutional neural networks, you will usually need to flatten data before feeding them to an output layer or a dense layer. To flatten an array with NumPy, we can reshape using hĝ as a first shape dimension, which means "however many elements are there" and effectively put them all in the first dimension making a flat 1D array. For an example of this concept:

```
ĦŇĞİĴĬĦ ł ĮĴgĞĹĹĞÑâäägdğādäġdġāãã
ÌĬĞĻĻĦĮĦĤ ł ĦŇĞİĴĬĦgĹĦĽÎĞĴĦâhĝã

ĴĹĨĮĻâĦŇĞİĴĬĦã
ĴĹĨĮĻâĦŇĞİĴĬĦgĽÎĞĴĦã

ĴĹĨĮĻâÌĬĞĻĻĦĮĦĤã
ĴĹĨĮĻâÌĬĞĻĻĦĮĦĤgĽÎĞĴĦã
```

```
ńńń
ääĝ ğā
 äġ ǵāā
âğd ğã
äĝ ğ ġ ǵā
âǵdã
```

We could also use ĮĴgÌĬĞĻĻĦĮâã, but our intention differs with a batch of samples. In the case of our samples, we still wish to retain all 60,000 of them, so we'd like to reshape our training data to be âħflflflfld hĝã. This will notify NumPy that we wish to keep the 60,000 samples (first dimension), but flatten the rest (-1 as the second dimension means that we want to put all of the sample data in this single dimension into a 1D array). This will create 60,000 samples of 784 features each. The 784 features are the result of 28·28. To do this, we'll use the number of samples from the training (XgĹÎĞĴĦäflā) and testing (XçĻĦĹĻgĹÎĞĴĦäflā) datasets respectively and reshape them:

```
ńı RĦĹÎĞĴĦ ĻIJ L·ĦĢĻIJĹĹ
X ł XgĹĦĹÎĞĴĦâXgĹÎĞĴĦäflād hĝã
XçĻĦĹĻ ł XçĻĦĹĻgĹĦĹÎĞĴĦâXçĻĦĹĻgĹÎĞĴĦäflād hĝã
```

You can achieve the same result by explicitly defining the shape, instead of relying on NumPy inference:

```
gĹĦĹÎĞĴĦâXgĹÎĞĴĦäflād XgĹÎĞĴĦäĝāǻXgĹÎĞĴĦäğāã
```

which would be more explicit, but we find this to be less legible.

Data Shuffling

Our dataset currently consists of samples and their target classifications, in order, from 0 to 9. To illustrate, we can query our *y* data at various points. The first 6000 will all be 0. For instance:

```
ĴĹĨĮĻâÑäflfĝflāã

ńńń
äfl fl fl fl fl fl fl fl fl flā
```

If we then query a bit later:

```
ĴĹĨĮĻâÑähflflflfhflĝflāã

ńńń
äĝ ĝ ĝ ĝ ĝ ĝ ĝ ĝ ĝ ĝā
```

This is a problem if we train our network with data in this order; for the same reason, an imbalanced dataset is problematic.

While we train on the first 6,000 samples, the model will learn that the quickest way to reduce loss is to always predict a 0, since it'll see several batches of the data with class 0 only. Then, between 6,000 and 12,000, the loss will initially spike as the label will change and the model will be predicting, incorrectly, still label 0, and will likely learn that now it needs to always predict class 1 (as that's all it sees in batches of labels which we optimize for). The model will cycle local minimums following whichever label is currently being repeated over batches and will most likely never find a global minimum. This process will continue until we get through all samples, repeating however many epochs we selected.

Preferably, there would be many classifications (ideally some from each class) of samples per fitment to keep the model from becoming biased towards any single class, simply because that's the class it's been seeing lately. Thus, we often will randomly shuffle the data. We didn't need to shuffle our previous training data, like the spiral data, since we were training on the entire dataset at once anyway, rather than individual batches. With this larger dataset that we're training on in batches, we want to shuffle the data, as it's currently organized in order of chunks

of 6,000 samples per label. When shuffling, we want to ensure that we shuffle the sample and target

arrays the same; otherwise, we'll have a very confused (and very wrong, in most cases) model as labels will no longer match samples. Hence, we cannot simply call `ĽÎŁÌÌĬĦâã` on both of them separately. There are many ways to achieve this, but what we'll do is gather all of the "keys," which are the same for samples and targets, and then shuffle them.

These keys will be values from 0 to 59999 in this case.

```
ĪĦÑĽ ł ĮĴgĞĹĹĞÑâĹĞĮÍĦâXgĽÎĞĴĦäflāãã
ĴĹĨĮĻâĪĦÑĽäfĝflāã

ńńń
ĞĹĹĞÑâäfl ĝ ğ ġ ǵ ĥ ħ ì í îāã
```

Then, we can shuffle these keys:

```
ĨİĴIJĹĻ ĮĮÌĽ

ĮĮÌĽgĨĮĨĻâã

ĮĴgĹĞĮĤIJİgĽÎŁÌÌĬĦâĪĦÑĽã
ĴĹĨĮĻâĪĦÑĽäfĝflāã

ńńń
äĥġħǵġ ĝǵħğġ ǵġĝíĝ ġħġflğ  ǵğîì ǵĝǵîġ ġîǵíĥ ĥflħġĝ ğîîflî ĝìħflǵā
```

Now, this is essentially the new order of indexes, which we can then apply by doing:

```
X ł XäĪĦÑĽã
Ñ ł ÑäĪĦÑĽã
```

This tells NumPy to return values of given indices as we would normally index NumPy arrays, but we're using a variable that keeps a list of randomly ordered indices. Then we can check a slice of targets:

```
ĴĹĨĮĻâÑäfĝĥāã

ńńń
äí ğ ì ħ fl ħ ħ í ǵ ğ î ǵ ğ ĝ flā
```

They seem to be shuffled. We can check individual samples as well:

```
ÕİĴIJĹĻ İĞĻĴĬIJĻĬÕĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĴĬĻgÕİĽÎIJŃââXägāgĹĦĽÎĞĴĦâğíd ğíããã  fi RĦĽÎĞĴĦ ĞĽ ÕİĞÍĦ ÕĽ Ğ LĦĢĻIJĹ ĞĬĹĦĞĤÑ
ĴĬĻgĽÎIJŃâã
ńńń
```

Fig 19.06: Random (shirt) image after shuffling

We can then check the class at the same index:

```
ĴĹÕĮĻâÑägāã
```

```
ńńń
fl
```

Class 0 is indeed “shirt,” and so these data do look properly shuffled. You may check a few more manually to ensure your data are as expected. If the model does not train or appears to be misbehaving, you will want to double-check how you preprocessed the data.

Batches

So far, we've trained our models by feeding the entire dataset as a single "batch" through the model. We discussed earlier in Chapter 2 why it is preferred to do more than 1 sample at a time, but is there a batch size that would be too big? Our dataset has been small enough for us to get away with the behavior of feeding the entire dataset at once, but real-world datasets can often be terabytes or more in size, which is a nonstarter for the majority of computers to process as a single batch. A batch is a slice of a fixed size from the data. When we train with batches, we iterate through the dataset in a chunk, or "batch, " of data at a time, performing a forward pass, loss calculation, backward pass, and optimization. If the data have been shuffled, and each batch is large enough and somewhat representative of the dataset, it is a fair assumption that each gradient of each batch should be a good approximation of the direction towards a global minimum. If the batch is too small, the direction of the gradient descent can fluctuate too much from batch to batch, causing the model to take a long time to train.

Common batch sizes range between 32 and 128 samples. You can go smaller if you're having trouble fitting everything into memory, or larger if you want training to go faster, but this range is the typical range of batch sizes. You will usually see accuracy and loss improvements by increasing the batch size from say 2 to 8, or 8 to 32. At some point, however, you will see diminishing returns regarding accuracy and loss if you keep increasing the batch size. Additionally, training with large batch sizes will become slow compared to the speed achievable with smaller batch sizes — like our examples earlier with the spiral data, requiring 10 thousand epochs to train! As is often the case with neural networks, it's a lot of trial and error with your specific data and model. For example, imagine we select a batch size of 128, and we opt to do 10 epochs.

This means, for each epoch, we will iterate over our data, fitting 128 samples at a time to train our model. Each batch of samples being trained is referred to as a **step**. We can calculate the number of **steps** by dividing the number of samples by the batch size:

```
ĹĻĦĴĹ ł XgĹÎĞĴĦäflā ww BATCHçSIZE
```

We use the integer division operator, `ww`, (instead of the floating-point division operator, `w`) to return an integer, as the number of steps cannot contain a fraction. This is the number of iterations that we'll be making per epoch in a loop. If there are some straggler samples left over, we can add

them in by simply adding one more step:

```
ÕÌ ĽĻĦĴĽ ắ BATCHçSIZE 1· XgĽÎĞĴĦäflāf
    ĽĻĦĴĽ кł ĝ
```

Why we add this 1 can be presented in a simple example:

```
ĠĞĻĢÎçĽÎŅĦ ł ğ

X ł äĝd ğd ġd ǵā
ĴĹÕĮĻâĬĦĮâXã ww ĠĞĻĢÎçĽÎŅĦã

ńńń
ğ


X ł äĝd ğd ġd ǵd ĥā
ĴĹÕĮĻâĬĦĮâXã ww ĠĞĻĢÎçĽÎŅĦã

ńńń
ğ
```

Integer division rounds down; thus, if there are any samples left, we'll add 1 to form the last batch with the remainder.

An example of code leading up to training a model using batches:

```
ÕİĴIJĹĻ ĮĮÌĽ
ÌĹIJİ ĮĮÌĽgĤĞĻĞĽĦĻĽ ÕİĴIJĹĻ ĽĴÕĹĞĬçĤĞĻĞ


ĮĮÌĽgÕĮÕĻâã

ſi CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñ ł ĽĴÕĹĞĬçĤĞĻĞâĽĞİĴĬĦĽłĝflfld ĢĬĞĽĽĦĽłġã

EPOCHS ł ĝfl
BATCHçSIZE ł ĝğí  ſi WĦ ĻĞĪĦ ĝğí ĽĞİĴĬĦĽ ĞĻ IJĮĢĦ

ſi CĞĬĢŁĬĞĻĦ ĮŁİĠĦĹ IJÌ ĽĻĦĴĽ
ĽĻĦĴĽ ł XgĽÎĞĴĦäflā ww BATCHçSIZE
ſi DÕĿÕĤÕĮÍ ĹIJŁĮĤĽ ĤIJŃĮg IÌ ĻÎĦĹĦ ĞĹĦ ĽIJİĦ ĹĦİĞÕĮÕĮÍ ĤĞĻĞd
ſi ĠŁĻ ĮIJĻ Ğ ÌŁĬĬ ĠĞĻĢÎd ĻÎÕĽ ŃIJĮďĻ ÕĮĢĬŁĤĦ ÕĻg
ſi AĤĤ ĝ ĻIJ ÕĮĢĬŁĤĦ ĻÎĦ ĹĦİĞÕĮÕĮÍ ĽĞİĴĬĦĽ ÕĮ ĝ İIJĹĦ ĽĻĦĴg
ÕÌ ĽĻĦĴĽ ắ BATCHçSIZE 1· XgĽÎĞĴĦäflāf
    ĽĻĦĴĽ кł ĝ
```

```
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâEPOCHSãf
    ÌIJĹ ĹĻĦĴ ĨĮ ĹĞĮÍĦâĹĻĦĴĹãf
        ĠĞĻĢÎçX ł XäĹĻĦĴǻBATCHçSIZEfâĹĻĦĴκĝãǻBATCHçSIZEã
        ĠĞĻĢÎçÑ ł ÑäĹĻĦĴǻBATCHçSIZEfâĹĻĦĴκĝãǻBATCHçSIZEã

        ħ NIJŃ ŃĦ ĴĦĹÌIJĹİ ÌIJĹŃĞĹÂ ĴĞĹĹd ĬIJĹĹ ĢĞĬĢŁĬĞĻĨIJĮd
        ħ ĠĞĢĪŃĞĹÂ ĴĞĹĹ ĞĮÂ ŁĴÂĞĻĦ ĴĞĹĞİĦĻĦĹĹ
```

We loaded the dataset, defined the number of epochs and a batch size, then calculated the number of steps. Next, we have two loops — an outer one over the epochs and an inner one over the steps. During each step in each epoch, we're selecting a slice of the training data. Now that we know how to train the model in batches, we're interested in the training loss and accuracy for each step and epoch. So far, we've only been calculating loss per fit, but recall that we fitted against the entire dataset at once. Now, we'll be interested in both batch-wise statistics and epoch-wise. For the overall loss and accuracy, we want to calculate a sample-wise average. To do this, we will accumulate the sum of losses from all epoch batches and counts to calculate the mean value at the end of each epoch. We'll start in the common LIJĹĹ class' ĢĞĬĢŁĬĞĻĦ method by adding:

```
        ħ AÂÂ ĞĢĢŁİŁĬĞĻĦÂ ĹŁİ IJÌ ĬIJĹĹĦĹ ĞĮÂ ĹĞİĴĬĦ ĢIJŁĮĻ
        ĹĦĬÌgĞĢĢŁİŁĬĞĻĦÂçĹŁİ κł ĮĴgĹŁİâĹĞİĴĬĦçĬIJĹĹĦĹã
        ĹĦĬÌgĞĢĢŁİŁĬĞĻĦÂçĢIJŁĮĻ κł ĬĦĮâĹĞİĴĬĦçĬIJĹĹĦĹã
```

Making the full ĢĞĬĢŁĬĞĻĦ method in the LIJĹĹ class:

```
    ħ CĞĬĢŁĬĞĻĦĹ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹÎŃĞĻĨIJĮ ĬIJĹĹĦĹ
    ħ ÍÎĿĦĮ ÌIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ĿĞĬŁĦĹ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĹĦĬÌd IJŁĻĴŁĻd Ñd ǻd ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹÎŃĞĻĨIJĮłFĞĬĹĦãf

        ħ CĞĬĢŁĬĞĻĦ ĹĞİĴĬĦ ĬIJĹĹĦĹ
        ĹĞİĴĬĦçĬIJĹĹĦĹ ł ĹĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        ħ CĞĬĢŁĬĞĻĦ ÌĦĞĮ ĬIJĹĹ
        ÂĞĻĞçĬIJĹĹ ł ĮĴgÌĦĞĮâĹĞİĴĬĦçĬIJĹĹĦĹã

        ħ AÂÂ ĞĢĢŁİŁĬĞĻĦÂ ĹŁİ IJÌ ĬIJĹĹĦĹ ĞĮÂ ĹĞİĴĬĦ ĢIJŁĮĻ
        ĹĦĬÌgĞĢĢŁİŁĬĞĻĦÂçĹŁİ κł ĮĴgĹŁİâĹĞİĴĬĦçĬIJĹĹĦĹã
        ĹĦĬÌgĞĢĢŁİŁĬĞĻĦÂçĢIJŁĮĻ κł ĬĦĮâĹĞİĴĬĦçĬIJĹĹĦĹã

        ħ IÌ ÏŁĹĻ ÂĞĻĞ ĬIJĹĹ h ĹĦĻŁĹĮ ĨĻ
        ĨÌ ĮIJĻ ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹÎŃĞĻĨIJĮf
            ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĹ

        ħ RĦĻŁĹĮ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹÎŃĞĻĨIJĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĹd ĹĦĬÌgĹĦÍŁĬĞĹÎŃĞĻĨIJĮçĬIJĹĹâã
```

We're saving the sum and the count so we can calculate the mean at any point. To do that, we'll add a new method called `ĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦĤ` inside the `LIJĹĽ` class:

```
# CĞĬĢŁĬĞĻĦĽ ĞĢĢŁİŁĬĞĻĦĤ ĬIJĹĽ
ĤĦÌ ĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦĤâĹĦĬÌd ǻd ĨĮĢĬŁĤĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮŁFĞĬĽĦãf

    # CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĹĽ
    ĤĞĻĞçĬIJĹĽ ł ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĽŁİ w ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ

    # IÌ ÏŁĽĻ ĤĞĻĞ ĬIJĹĽ h ĹĦĻŁĹĮ ĨĻ
    ĨÌ ĮIJĻ ĨĮĢĬŁĤĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮf
        ĹĦĻŁĹĮ ĤĞĻĞçĬIJĹĽ

    # RĦĻŁĹĮ ĻÎĦ ĤĞĻĞ ĞĮĤ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĽĦĹ
    ĹĦĻŁĹĮ ĤĞĻĞçĬIJĹĽd ĽĦĬÌgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĽâã
```

This method can also return the regularization loss if `ĨĮĢĬŁĤĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮ` is set to `TĹŁĦ`. The regularization loss does not need to be accumulated as it's calculated from the current state of layer parameters, at the time it's called. We'll be using this ability during training, but not while evaluating and predicting; we'll discuss this in more detail shortly.

Finally, in order to reset the sum and count values for a new epoch, we'll add one last method:

```
# RĦĽĦĻ ĻĞĹĨĞĠĬĦĽ ÌIJĹ ĞĢĢŁİŁĬĞĻĦĤ ĬIJĹĽ
ĤĦÌ ĮĦŃçĴĞĽĽâĹĦĬÌãf
    ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĽŁİ ł fl
    ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ ł fl
```

Making our full common LIJĹĹ class:

```
ﬁ CIJİİIJĮ ĬIJĹĹ ĢĬĞĹĹ
ĢĬĞĹĹ LIJĹĹf

    ﬁ RHÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĹ ĢĞĬĢŁĬĞĻĨIJĮ
    ÎHÌ ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹâĹHĬÌãf

        ﬁ ﬂ ĠÑ ÎHÌĞŁĬĻ
        ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ ł ﬂ

        ﬁ CĞĬĢŁĬĞĻH ĹHÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĹ
        ﬁ ĨĻHĹĞĻH ĞĬĬ ĻĹĞĨĮĞĠĬH ĬĞÑHĹĹ
        ÌIJĹ ĬĞÑHĹ ĨĮ ĹHĬÌgĻĹĞĨĮĞĠĬHçĬĞÑHĹĹf

            ﬁ Lĝ ĹHÍŁĬĞĹĨŅĞĻĨIJĮ h ŃHĨÍÎĻĹ
            ﬁ ĢĞĬĢŁĬĞĻH IJĮĬÑ ŃÎHĮ ÌĞĢĻIJĹ ÍĹHĞĻHĹ ĻÎĞĮ ﬂ
            ĨÌ ĬĞÑHĹgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬĝ ń ﬂf
                ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ ĸł ĬĞÑHĹgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬĝ ắ ċ
                                        ĮĴgĹŁİâĮĴgĞĠĹâĬĞÑHĹgŃHĨÍÎĻĹãã

            ﬁ Lğ ĹHÍŁĬĞĹĨŅĞĻĨIJĮ h ŃHĨÍÎĻĹ
            ĨÌ ĬĞÑHĹgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬğ ń ﬂf
                ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ ĸł ĬĞÑHĹgŃHĨÍÎĻçĹHÍŁĬĞĹĨŅHĹçĬğ ắ ċ
                                        ĮĴgĹŁİâĬĞÑHĹgŃHĨÍÎĻĹ ắ ċ
                                                ĬĞÑHĹgŃHĨÍÎĻĹã

            ﬁ Lĝ ĹHÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĹHĹ
            ﬁ ĢĞĬĢŁĬĞĻH IJĮĬÑ ŃÎHĮ ÌĞĢĻIJĹ ÍĹHĞĻHĹ ĻÎĞĮ ﬂ
            ĨÌ ĬĞÑHĹgĠĨĞĹçĹHÍŁĬĞĹĨŅHĹçĬĝ ń ﬂf
                ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ ĸł ĬĞÑHĹgĠĨĞĹçĹHÍŁĬĞĹĨŅHĹçĬĝ ắ ċ
                                        ĮĴgĹŁİâĮĴgĞĠĹâĬĞÑHĹgĠĨĞĹHĹãã

            ﬁ Lğ ĹHÍŁĬĞĹĨŅĞĻĨIJĮ h ĠĨĞĹHĹ
            ĨÌ ĬĞÑHĹgĠĨĞĹçĹHÍŁĬĞĹĨŅHĹçĬğ ń ﬂf
                ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ ĸł ĬĞÑHĹgĠĨĞĹçĹHÍŁĬĞĹĨŅHĹçĬğ ắ ċ
                                        ĮĴgĹŁİâĬĞÑHĹgĠĨĞĹHĹ ắ ċ
                                                ĬĞÑHĹgĠĨĞĹHĹã

        ĹHĻŁĹĮ ĹHÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ

    ﬁ SHĻwĹHİHİĠHĹ ĻĹĞĨĮĞĠĬH ĬĞÑHĹĹ
    ÎHÌ ĹHİHİĠHĹçĻĹĞĨĮĞĠĬHçĬĞÑHĹĹâĹHĬÌd ĻĹĞĨĮĞĠĬHçĬĞÑHĹĹãf
        ĹHĬÌgĻĹĞĨĮĞĠĬHçĬĞÑHĹĹ ł ĻĹĞĨĮĞĠĬHçĬĞÑHĹĹ
```

```
ƒ CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽĦĽ
ƒ ÍĨŁĦĮ ĬIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ŁĞĬŁĦĽ
ÂĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñd ắd ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮŁFĞĬĽĦãf

    ƒ CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦ ĬIJĽĽĦĽ
    ĽĞİĴĬĦçĬIJĽĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

    ƒ CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĽĽ
    ÂĞĻĞçĬIJĽĽ ł ĮĴgĬĦĞĮâĽĞİĴĬĦçĬIJĽĽĦĽã

    ƒ AÂÂ ĞĢĢŁĬŁĬĞĻĦÂ ĽŁĬ IJÌ ĬIJĽĽĦĽ ĞĮÂ ĽĞİĴĬĦ ĢIJŁĮĻ
    ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĽŁĬ κł ĮĴgĽŁĬâĽĞİĴĬĦçĬIJĽĽĦĽã
    ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĢIJŁĮĻ κł ĬĦĮâĽĞİĴĬĦçĬIJĽĽĦĽã

    ƒ IÌ ĬŁĽĻ ÂĞĻĞ ĬIJĽĽ h ĹĦĻŁĹĮ ĨĻ
    ĨÌ ĮIJĻ ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮf
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĽĽ

    ƒ RĦĻŁĹĮ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽĦĽ
    ĹĦĻŁĹĮ ÂĞĻĞçĬIJĽĽd ĽĦĬÌgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽâã

ƒ CĞĬĢŁĬĞĻĦĽ ĞĢĢŁĬŁĬĞĻĦÂ ĬIJĽĽ
ÂĦÌ ĢĞĬĢŁĬĞĻĦçĞĢĢŁĬŁĬĞĻĦÂâĽĦĬÌd ắd ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮŁFĞĬĽĦãf

    ƒ CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĽĽ
    ÂĞĻĞçĬIJĽĽ ł ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĽŁĬ w ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĢIJŁĮĻ

    ƒ IÌ ĬŁĽĻ ÂĞĻĞ ĬIJĽĽ h ĹĦĻŁĹĮ ĨĻ
    ĨÌ ĮIJĻ ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮf
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĽĽ

    ƒ RĦĻŁĹĮ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽĦĽ
    ĹĦĻŁĹĮ ÂĞĻĞçĬIJĽĽd ĽĦĬÌgĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽâã

ƒ RĦĽĦĻ ŁĞĹĨĞĠĬĦĽ ÌIJĹ ĞĢĢŁĬŁĬĞĻĦÂ ĬIJĽĽ
ÂĦÌ ĮĦŃçĴĞĽĽâĽĦĬÌãf
    ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĽŁĬ ł fl
    ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĢIJŁĮĻ ł fl
```

We'll want to implement the same things for the AĢĢŁĹĞĢÑ class now:

```
# CIJİİIJĮ ĞĢĢŁĹĞĢÑ ĢĬĞĹĹ
ĢĬĞĹĹ AĢĢŁĹĞĢÑf

    # CĞĬĢŁĬĞĻĦĹ ĞĮ ĞĢĢŁĹĞĢÑ
    # ÍĨĿĦĮ ĴĹĦĤĨĢĻĨIJĮĹ ĞĮĤ ÍĹIJŁĮĤ ĻĹŁĻÎ ĿĞĬŁĦĹ
    ĤĦÌ ĢĞĬĢŁĬĞĻĦâĹĦĬÌd ĴĹĦĤĨĢĻĨIJĮĹd Ñãf

        # GĦĻ ĢIJİĴĞĹĨĹIJĮ ĹĦĹŁĬĻĹ
        ĢIJİĴĞĹĨĹIJĮĹ ł ĹĦĬÌgĢIJİĴĞĹĦâĴĹĦĤĨĢĻĨIJĮĹd Ñã

        # CĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĢIJİĴĞĹĨĹIJĮĹã

        # AĤĤ ĞĢĢŁİŁĬĞĻĦĤ ĹŁİ IJÌ İĞĻĢĨĨĮÍ ĿĞĬŁĦĹ ĞĮĤ ĹĞİĴĬĦ ĢIJŁĮĻ
        ĹĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĹŁİ κł ĮĴgĹŁİâĢIJİĴĞĹĨĹIJĮĹã
        ĹĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ κł ĬĦĮâĢIJİĴĞĹĨĹIJĮĹã

        # RĦĻŁĹĮ ĞĢĢŁĹĞĢÑ
        ĹĦĻŁĹĮ ĞĢĢŁĹĞĢÑ

    # CĞĬĢŁĬĞĻĦĹ ĞĢĢŁİŁĬĞĻĦĤ ĞĢĢŁĹĞĢÑ
    ĤĦÌ ĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦĤâĹĦĬÌãf

        # CĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĞĢĢŁĹĞĢÑ ł ĹĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĹŁİ w ĹĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ

        # RĦĻŁĹĮ ĻÎĦ ĤĞĻĞ ĞĮĤ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ĞĢĢŁĹĞĢÑ

    # RĦĹĦĻ ĿĞĹĨĞĠĬĦĹ ÌIJĹ ĞĢĢŁİŁĬĞĻĦĤ ĞĢĢŁĹĞĢÑ
    ĤĦÌ ĮĦŃçĴĞĹĹâĹĦĬÌãf
        ĹĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĹŁİ ł fl
        ĹĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ ł fl
```

Here, we've added setting the ĞĢĢŁİŁĬĞĻĦĤçĹŁİ and ĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ properties in the ĢĞĬĢŁĬĞĻĦ method for the epoch accuracy calculation, added a new ĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦĤ method that returns this accuracy, and finally added a ĮĦŃçĴĞĹĹ method to reset the ĞĢĢŁİŁĬĞĻĦĤçĹŁİ and ĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ values that we'll use at the beginning of each epoch.

Now, we'll modify the ĻĹĞĨĮ method for our MIJĤĦĬ class. First, we'll add a new parameter called ĠĞĻĢĨçĹĨŅĦ:

```
    ĤĦÌ ĻĹĞĨĮâĹĦĬÌd Xd Ñd ắd ĦĴIJĢĨĹłĝd ĠĞĻĢĨçĹĨŅĦłNIJĮĦd
            ĴĹĨĮçĦĿĦĹÑłĝd ĿĞĬĨĤĞĻĨIJĮçĤĞĻĞłNIJĮĦãf
```

We'll default this parameter to NIJĮĦ, which means to use the entire dataset as the batch. In this case, training will take 1 step per epoch, where that step consists of feeding all the data through the network at once.

```
# DĦÌĞŁĬĻ ŁĞĬŁĦ ĨÌ ĠĞĻĢÎ ĹĨŅĦ ĨĹ ĮIJĻ ĹĦĻ
ĻĹĞĨĮçĹĻĦĴĹ ł ĝ

# IÌ ĻÎĦĹĦ ĨĹ ŁĞĬĨÎĞĻĨIJĮ ÎĞĻĞ ĴĞĹĹĦÎd
# ĹĦĻ ÎĦÌĞŁĬĻ ĮŁİĠĦĹ IJÌ ĹĻĦĴĹ ÌIJĹ ŁĞĬĨÎĞĻĨIJĮ ĞĹ ŃĦĬĬ
ĨÌ ŁĞĬĨÎĞĻĨIJĮçÎĞĻĞ ĨĹ ĮIJĻ NIJĮĦf
    ŁĞĬĨÎĞĻĨIJĮçĹĻĦĴĹ ł ĝ

    # FIJĹ ĠĦĻĻĦĹ ĹĦĞÎĞĠĞĨĬĨĻÑ
    XçŁĞĬd ÑçŁĞĬ ł ŁĞĬĨÎĞĻĨIJĮçÎĞĻĞ
```

As discussed, most "real life" datasets will require a batch size smaller than that of all samples. We'll handle that using the method that we described earlier: performing integer division of the number of all samples by the batch size and eventually adding 1 to include any remaining samples that did not form a full batch (we'll do that for both training and validation data):

```
# CĞĬĢŁĬĞĻĦ ĮŁİĠĦĹ IJÌ ĹĻĦĴĹ
ĨÌ ĠĞĻĢÎçĹĨŅĦ ĨĹ ĮIJĻ NIJĮĦf
    ĻĹĞĨĮçĹĻĦĴĹ ł ĬĦĮâXã ww ĠĞĻĢÎçĹĨŅĦ
    # DĨŁĨÎĨĮÍ ĹIJŁĮÎĹ ÎIJŃĮg IÌ ĻÎĦĹĦ ĞĹĦ ĹIJİĦ ĹĦİĞĨĮĨĮÍ
    # ÎĞĻĞd ĠŁĻ ĮIJĻ Ğ ÌŁĬĬ ĠĞĻĢÎd ĻÎĨĹ ŃIJĮďĻ ĨĮĢĬŁÎĦ ĨĻ
    # AÎÎ ĝ ĻIJ ĨĮĢĬŁÎĦ ĻÎĨĹ ĮIJĻ ÌŁĬĬ ĠĞĻĢÎ
    ĨÌ ĻĹĞĨĮçĹĻĦĴĹ ǻ ĠĞĻĢÎçĹĨŅĦ l· ĬĦĮâXãf
        ĻĹĞĨĮçĹĻĦĴĹ κł ĝ

    ĨÌ ŁĞĬĨÎĞĻĨIJĮçÎĞĻĞ ĨĹ ĮIJĻ NIJĮĦf
        ŁĞĬĨÎĞĻĨIJĮçĹĻĦĴĹ ł ĬĦĮâXçŁĞĬã ww ĠĞĻĢÎçĹĨŅĦ
        # DĨŁĨÎĨĮÍ ĹIJŁĮÎĹ ÎIJŃĮg IÌ ĻÎĦĹĦ ĞĹĦ ĹIJİĦ ĹĦİĞĨĮĨĮÍ
        # ÎĞĻĞd ĠŁĻ ĮIJĹ ÌŁĬĬ ĠĞĻĢÎd ĻÎĨĹ ŃIJĮďĻ ĨĮĢĬŁÎĦ ĨĻ
        # AÎÎ ĝ ĻIJ ĨĮĢĬŁÎĦ ĻÎĨĹ ĮIJĻ ÌŁĬĬ ĠĞĻĢÎ
        ĨÌ ŁĞĬĨÎĞĻĨIJĮçÎĞĻĞ ĨĹ ĮIJĻ NIJĮĦf
            ŁĞĬĨÎĞĻĨIJĮçĹĻĦĴĹ ǻ ĠĞĻĢÎçĹĨŅĦ l· ĬĦĮâXçŁĞĬãf
                ŁĞĬĨÎĞĻĨIJĮçĹĻĦĴĹ κł ĝ
```

Next, starting at the top, we'll modify the loop over epochs to print an epoch number and then reset the accumulated epoch loss and accuracy values. Then, inside of here, we'll add a new loop that will iterate over steps in the epoch.

```
    # PĹĨĮĻ ĦĴIJĢÎ ĮŁİĠĦĹ
    ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåďã

    # RĦĹĦĻ ĞĢĢŁİŁĬĞĻĦÎ ŁĞĬŁĦĹ ĨĮ ĬIJĹĹ ĞĮÎ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĹ
    ĹĦĬÌgĬIJĹĹgĮĦŃçĴĞĹĹâã
    ĹĦĬÌgĞĢĢŁĹĞĢÑgĮĦŃçĴĞĹĹâã
```

```
# IĻHĹĞĻH IJŁHĹ ĹĻHĴĽ
ÌIJĹ ĽĻHĴ ÕIĮ ĹĞĮÍHâĻĹĞÕIĮçĽĻHĴĽãf
```

Inside of each step, we'll need to grab the **batch** of data that we'll use to train — either the full dataset if the ĠĞĻĢÎçĽÕŅH parameter is still the default NIJĮH or a slice of size ĠĞĻĢÎçĽÕŅH:

```
# IÌ ĠĞĻĢÎ ĹÕŅH ÕĽ ĮIJĻ ĽHĻ h
# ĻĹĞÕĮ ŁĽÕĮÍ IJĮH ĽĻHĴ ĞĮÂ ÌŁĬÌ ÂĞĻĞĽHĻ
 ÕÌ ĠĞĻĢÎçĽÕŅH ÕĽ NIJĮHf
     ĠĞĻĢÎçX ł X
     ĠĞĻĢÎçÑ ł Ñ

 # OĻÎHĹŃÕĽH ĹĬÕĢH Ğ ĠĞĻĢÎ
 HĬĽHf
     ĠĞĻĢÎçX ł XäĽĻHĴåĠĞĻĢÎçĽÕŅHfâĽĻHĴкĝãåĠĞĻĢÎçĽÕŅHã
     ĠĞĻĢÎçÑ ł ÑäĽĻHĴåĠĞĻĢÎçĽÕŅHfâĽĻHĴкĝãåĠĞĻĢÎçĽÕŅHã
```

With each of these batches, we fit and print information, similar to how we were fitting per epoch. The difference now is we use ĠĞĻĢÎçX instead of X and ĠĞĻĢÎçÑ instead of Ñ. The other change is the ÕÌ statement for the summary printing that will account for steps instead of epochs:

```
# PHĹÌIJĹİ ĻÎH ÌIJĹŃĞĹÂ ĴĞĽĽ
IJŁĻĴŁĻ ł ĽHĬÌgÌIJĹŃĞĹÂâĠĞĻĢÎçXd ĻĹĞÕĮÕĮÍłTĹŁHã

# CĞĬĢŁĬĞĻH ĬIJĽĽ
ÂĞĻĞçĬIJĽĽd ĹHÍŁĬĞĹÕŅĞĻÕIJIçĬIJĽĽ ł ċ
    ĽHĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻHâIJŁĻĴŁĻd ĠĞĻĢÎçÑd
                        ÕĮĢĬŁÂHçĹHÍŁĬĞĹÕŅĞĻÕIJĮłTĹŁHã
ĬIJĽĽ ł ÂĞĻĞçĬIJĽĽ к ĹHÍŁĬĞĹÕŅĞĻÕIJIçĬIJĽĽ

# GHĻ ĴĹHÂÕĢĻÕIJĮĽ ĞĮÂ ĢĞĬĢŁĬĞĻH ĞĮ ĞĢĢŁĹĞĢÑ
ĴĹHÂÕĢĻÕIJĮĽ ł ĽHĬÌgIJŁĻĴŁĻçĬĞÑHĹçĞĢĻÕĽĞĻÕIJIgĴĹHÂÕĢĻÕIJĮĽâ
                  IJŁĻĴŁĻã
ĞĢĢŁĹĞĢÑ ł ĽHĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻHâĴĹHÂÕĢĻÕIJĮĽd
                                    ĠĞĻĢÎçÑã

# PHĹÌIJĹİ ĠĞĢĪŃĞĹÂ ĴĞĽĽ
ĽHĬÌgĠĞĢĪŃĞĹÂâIJŁĻĴŁĻd ĠĞĻĢÎçÑã

# OĴĻÕIÌÕŅH âŁĴÂĞĻH ĴĞĹĞİHĻHĹĽã
ĽHĬÌgIJĴĻÕIÌÕŅHĹgĴĹHçŁĴÂĞĻHçĴĞĹĞİĽâã
ÌIJĹ ĬĞÑHĹ ÕĮ ĽHĬÌgĻĹĞÕĮĞĠİHçĬĞÑHĹĽf
    ĽHĬÌgIJĴĻÕIÌÕŅHĹgŁĴÂĞĻHçĴĞĹĞİĽâĬĞÑHĹã
ĽHĬÌgIJĴĻÕIÌÕŅHĹgĴIJĽĻçŁĴÂĞĻHçĴĞĹĞİĽâã

# PĹÕĮĻ Ğ ĽŁİİĞĹÑ
ÕÌ ĮIJĻ ĽĻHĴ ķ ĴĹÕĮĻçHŁHĹÑ IJĹ ĽĻHĴ łł ĻĹĞÕĮçĽĻHĴĽ h ĝf
    ĴĹÕĮĻâÌďĽĻHĴf ăĽĻHĴåd ď к
          ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď к
          ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď к
```

```
            ÌďĤĞĻĞçĬIJĹĹf ăĤĞĻĞçĬIJĹĹfgġÌåd ď ĸ
            ÌďĹĦÍçĬIJĹĹf ăĹĦÍłĬĞĹĨŅĞĻĨIJĮçĬIJĹĹfgġÌåãd ď ĸ
            ÌďĬĹf ăĹĦĬÌgIJĴĻĨĬĨŅĦĹgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã
```

Then we'd like to print information like accuracy and loss per epoch:

```
ħ GĦĻ ĞĮĤ ĴĹĨĮĻ ĦĴIJĢĬ ĬIJĹĹ ĞĮĤ ĞĢĢłĹĞĢÑ
ĦĴIJĢĬçĤĞĻĞçĬIJĹĹd ĦĴIJĢĬçĹĦÍłĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ ł ċ
    ĹĦĬÌgĬIJĹĹgĢĞĬĢłĬĞĻĦçĞĢĢłİłĬĞĻĦĤâ
        ĨĮĢĬłĤĦçĹĦÍłĬĞĹĨŅĞĻĨIJĮłTĹłĦã
ĦĴIJĢĬçĬIJĹĹ ł ĦĴIJĢĬçĤĞĻĞçĬIJĹĹ ĸ ĦĴIJĢĬçĹĦÍłĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ
ĦĴIJĢĬçĞĢĢłĹĞĢÑ ł ĹĦĬÌgĞĢĢłĹĞĢÑgĢĞĬĢłĬĞĻĦçĞĢĢłİłĬĞĻĦĤâã

ĴĹĨĮĻâÌďĻĹĞĨĮĨĮÍd ď ĸ
      ÌďĞĢĢf ăĦĴIJĢĬçĞĢĢłĹĞĢÑfgġÌåd ď ĸ
      ÌďĬIJĹĹf ăĦĴIJĢĬçĬIJĹĹfgġÌå âď ĸ
      ÌďĤĞĻĞçĬIJĹĹf ăĦĴIJĢĬçĤĞĻĞçĬIJĹĹfgġÌåd ď ĸ
      ÌďĹĦÍçĬIJĹĹf ăĦĴIJĢĬçĹĦÍłĬĞĹĨŅĞĻĨIJĮçĬIJĹĹfgġÌåãd ď ĸ
      ÌďĬĹf ăĹĦĬÌgIJĴĻĨĬĨŅĦĹgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã
```

If the batch size is set, the chances are that our validation data will be larger than this batch size, so we need to add batching for the validation data as well:

```
ħ IÌ ĻĬĦĹĦ ĨĹ ĻĬĦ ŁĞĬĨĤĞĻĨIJĮ ĤĞĻĞ
ĨÌ ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞ ĨĹ ĮIJĻ NIJĮĦf

    ħ RĦĹĦĻ ĞĢĢłİłĬĞĻĦĤ ŁĞĬłĦĹ ĨĮ ĬIJĹĹ
    ħ ĞĮĤ ĞĢĢłĹĞĢÑ IJĠÏĦĢĻĹ
    ĹĦĬÌgĬIJĹĹgĮĦŃçĴĞĹĹâã
    ĹĦĬÌgĞĢĢłĹĞĢÑgĮĦŃçĴĞĹĹâã

    ħ IĻĦĹĞĻĦ IJŁĦĹ ĹĻĦĴĹ
    ÌIJĹ ĹĻĦĴ ĨĮ ĹĞĮÍĦâŁĞĬĨĤĞĻĨIJĮçĹĻĦĴĹãf

        ħ IÌ ĠĞĻĢĬ ĹĨŅĦ ĨĹ ĮIJĻ ĹĦĻ h
        ħ ĻĹĞĨĮ łĹĨĮÍ IJĮĦ ĹĻĦĴ ĞĮĤ ÌłĬĬ ĤĞĻĞĹĦĻ
        ĨÌ ĠĞĻĢĬçĹĨŅĦ ĨĹ NIJĮĦf
            ĠĞĻĢĬçX ł XçŁĞĬ
            ĠĞĻĢĬçÑ ł ÑçŁĞĬ

        ħ OĻĬĦĹŃĨĹĦ ĹĬĨĢĦ Ğ ĠĞĻĢĬ
        ĦĬĹĦf
            ĠĞĻĢĬçX ł XçŁĞĬä
                ĹĻĦĴáĠĞĻĢĬçĹĨŅĦfâĹĻĦĴĸĝãáĠĞĻĢĬçĹĨŅĦ
            ã
            ĠĞĻĢĬçÑ ł ÑçŁĞĬä
                ĹĻĦĴáĠĞĻĢĬçĹĨŅĦfâĹĻĦĴĸĝãáĠĞĻĢĬçĹĨŅĦ
            ã
```

```
            # PHĹÌIJĹİ ĻÎH ÌIJĹŃĞĹÂ ĴĞĽĽ
            IJŁĻĴŁĻ ł ĽHĬÌgÌIJĹŃĞĹÂâĠĞĻĢÎçXd ĻĹĞĨĮĨĮÍłFĞĬĽHã

            # CĞĬĢŁĬĞĻH ĻÎH ĬIJĽĽ
            ĽHĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻHâIJŁĻĴŁĻd ĠĞĻĢÎçÑã

            # GHĻ ĴĹHÂĨĢĻĨIJĮĽ ĞĮÂ ĢĞĬĢŁĬĞĻH ĞĮ ĞĢĢŁĹĞĢÑ
            ĴĹHÂĨĢĻĨIJĮĽ ł ĽHĬÌgIJŁĻĴŁĻçĬĞÑHĹçĞĢĻĨĹĞĻĨIJĮgĴĹHÂĨĢĻĨIJĮĽâ
                                IJŁĻĴŁĻã
            ĽHĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻHâĴĹHÂĨĢĻĨIJĮĽd ĠĞĻĢÎçÑã

        # GHĻ ĞĮÂ ĴĹĨĮĻ ĹĞĬĨÂĞĻĨIJĮ ĬIJĽĽ ĞĮÂ ĞĢĢŁĹĞĢÑ
        ĹĞĬĨÂĞĻĨIJĮçĬIJĽĽ ł ĽHĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻHçĞĢĢŁİŁĬĞĻHÂâã
        ĹĞĬĨÂĞĻĨIJĮçĞĢĢŁĹĞĢÑ ł ĽHĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻHçĞĢĢŁİŁĬĞĻHÂâã

        ĴĹĨĮĻâÌďĹĞĬĨÂĞĻĨIJĮd ď κ
              ÌďĞĢĢf ăĹĞĬĨÂĞĻĨIJĮçĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĽĽf ăĹĞĬĨÂĞĻĨIJĮçĬIJĽĽfgġÌåďã
```

Compared to our current codebase, we've added calls to the ĮHÑçĴĞĽĽ method, of both loss and accuracy objects, which reset values accumulated during the training step. Next, we introduced batches (a loop iterating over steps), and removed catching a return from the loss calculation (we don't care about batch loss during validation, just the final, overall loss). The last steps were to add handling for the overall validation loss and replace XçĹĞĬ with ĠĞĻĢÎçX and ÑçĹĞĬ to ĠĞĻĢÎçÑ to match the changes made to the training code.

This makes our full ĻĹĞĨĮ method for the MIJÂHĬ class:

```
# TĹĞĨĮ ĻÎH ĬIJÂHĬ
ÂHÌ ĻĹĞĨĮâĽHĬÌd Xd Ñd ấd HĴIJĢÎĽłĝd ĠĞĻĢÎçĽĨĢHłNIJĮHd
          ĴĹĨĮĻçHĹHĹÑłĝd ĹĞĬĨÂĞĻĨIJĮçÂĞĻĞłNIJĮHãf

    # IĮĨĻĨĞĬĨĢH ĞĢĢŁĹĞĢÑ IJĠĬHĢĻ
    ĽHĬÌgĞĢĢŁĹĞĢÑgĨĮĨĻâÑã

    # DHÌĞŁĬĻ ĹĞĬŁH ĨÌ ĠĞĻĢÎ ĽĨĢH ĨĽ ĮIJĻ ĠHĨĮÍ ĽHĻ
    ĻĹĞĨĮçĽĻHĴĽ ł ĝ

    # IÌ ĻÎHĹH ĨĽ ĹĞĬĨÂĞĻĨIJĮ ÂĞĻĞ ĴĞĽĽHÂd
    # ĽHĻ ÂHÌĞŁĬĻ ĮŁĬĠHĹ IJÌ ĽĻHĴĽ ÌIJĹ ĹĞĬĨÂĞĻĨIJĮ ĞĽ ŃHĬĬ
    ĨÌ ĹĞĬĨÂĞĻĨIJĮçÂĞĻĞ ĨĽ ĮIJĻ NIJĮHf
        ĹĞĬĨÂĞĻĨIJĮçĽĻHĴĽ ł ĝ

        # FIJĹ ĠHĻĻHĹ ĹHĞÂĞĠĨĬĨĻÑ
        XçĹĞĬd ÑçĹĞĬ ł ĹĞĬĨÂĞĻĨIJĮçÂĞĻĞ
```

```
# CĞĬĢŁĬĞĻĦ ĮŁİĠĦĹ IJÌ ĽĻĦĴĽ
ÎÌ ĠĞĻĢÎçĽÎŅĦ ÎĽ ĮIJĻ NIJĮĦf
    ĻĹĞĬĮçĽĻĦĴĽ ł ĬĦĮâXã ww ĠĞĻĢÎçĽÎŅĦ
    # DÎĿÎĤÎĮÍ ĹIJŁĮĤĽ ĤIJŃĮg IÌ ĻÎĦĹĦ ĞĹĦ ĽIJİĦ ĹĦİĞÎĮÎĮÍ
    # ĤĞĻĞd ĠŁĻ ĮIJĻ Ğ ÌŁĬĬ ĠĞĻĢÎd ĻÎÎĽ ŃIJĮďĻ ÎĮĢĬŁĤĦ ÎĻ
    # AĤĤ ẀĝẀ ĻIJ ÎĮĢĬŁĤĦ ĻÎÎĽ ĮIJĻ ÌŁĬĬ ĠĞĻĢÎ
    ÎÌ ĻĹĞĬĮçĽĻĦĴĽ ǻ ĠĞĻĢÎçĽÎŅĦ l· ĬĦĮâXãf
        ĻĹĞĬĮçĽĻĦĴĽ κł ĝ

    ÎÌ L·ĞĬÎĤĞĻÎIJĮçĤĞĻĞ ÎĽ ĮIJĻ NIJĮĦf
        L·ĞĬÎĤĞĻÎIJĮçĽĻĦĴĽ ł ĬĦĮâXçL·ĞĬã ww ĠĞĻĢÎçĽÎŅĦ
        # DÎĿÎĤÎĮÍ ĹIJŁĮĤĽ ĤIJŃĮg IÌ ĻÎĦĹĦ ĞĹĦ ĽIJİĦ ĹĦİĞÎĮÎĮÍ
        # ĤĞĻĞd ĠŁĻ ĮIJĹ ÌŁĬĬ ĠĞĻĢÎd ĻÎÎĽ ŃIJĮďĻ ÎĮĢĬŁĤĦ ÎĻ
        # AĤĤ ẀĝẀ ĻIJ ÎĮĢĬŁĤĦ ĻÎÎĽ ĮIJĻ ÌŁĬĬ ĠĞĻĢÎ
        ÎÌ L·ĞĬÎĤĞĻÎIJĮçĽĻĦĴĽ ǻ ĠĞĻĢÎçĽÎŅĦ l· ĬĦĮâXçL·ĞĬãf
            L·ĞĬÎĤĞĻÎIJĮçĽĻĦĴĽ κł ĝ

# MĞÎĮ ĻĹĞĬĮÎĮÍ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ÎĮ ĹĞĮÍĦâĝd ĦĴIJĢÎĽκĝãf

    # PĹÎĮĻ ĦĴIJĢÎ ĮŁİĠĦĹ
    ĴĹÎĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåďã

    # RĦĽĦĻ ĞĢĢŁİŁĬĞĻĦĤ L·ĞĬŁĦĽ ÎĮ ĬIJĽĽ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĽ
    ĽĦĬÌgĬIJĽĽgĮĦŃçĴĞĽĽâã
    ĽĦĬÌgĞĢĢŁĹĞĢÑgĮĦŃçĴĞĽĽâã

    # IĻĦĹĞĻĦ IJL·ĦĹ ĽĻĦĴĽ
    ÌIJĹ ĽĻĦĴ ÎĮ ĹĞĮÍĦâĻĹĞÎĮçĽĻĦĴĽãf

        # IÌ ĠĞĻĢÎ ĽÎŅĦ ÎĽ ĮIJĻ ĽĦĻ h
        # ĻĹĞÎĮ ŁĽÎĮÍ IJĮĦ ĽĻĦĴ ĞĮĤ ÌŁĬĬ ĤĞĻĞĽĦĻ
        ÎÌ ĠĞĻĢÎçĽÎŅĦ ÎĽ NIJĮĦf
            ĠĞĻĢÎçX ł X
            ĠĞĻĢÎçÑ ł Ñ

        # OĻÎĦĹŃÎĽĦ ĽĬÎĢĦ Ğ ĠĞĻĢÎ
        ĦĬĽĦf
            ĠĞĻĢÎçX ł XäĽĻĦĴǻĠĞĻĢÎçĽÎŅĦfâĽĻĦĴκĝãǻĠĞĻĢÎçĽÎŅĦã
            ĠĞĻĢÎçÑ ł ÑäĽĻĦĴǻĠĞĻĢÎçĽÎŅĦfâĽĻĦĴκĝãǻĠĞĻĢÎçĽÎŅĦã

        # PĦĹÌIJĹÌ ĻÎĦ ÌIJĹŃĞĹĤ ĴĞĽĽ
        IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹĤâĠĞĻĢÎçXd ĻĹĞÎĮÎĮÍŁTĹŁĦã

        # CĞĬĢŁĬĞĻĦ ĬIJĽĽ
        ĤĞĻĞçĬIJĽĽd ĹĦÍŁĬĞĹÎŅĞĻÎIJĮçĬIJĽĽ ł ċ
            ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ĠĞĻĢÎçÑd
                                ÎĮĢĬŁĤĦçĹĦÍŁĬĞĹÎŅĞĻÎIJĮŁTĹŁĦã
        ĬIJĽĽ ł ĤĞĻĞçĬIJĽĽ κ ĹĦÍŁĬĞĹÎŅĞĻÎIJĮçĬIJĽĽ
```

```
    ſı GĦĻ ĴĹĦĤĨĢĻĨIJĮĽ ĞĮĤ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
    ĴĹĦĤĨĢĻĨIJĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨŁĞĻĨIJĮgĴĹĦĤĨĢĻĨIJĮĽâ
                        IJŁĻĴŁĻã
    ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦĤĨĢĻĨIJĮĽd
                                            ĠĞĻĢÎçÑã

    ſı PĦĹÌIJĹİ ĠĞĢĪŃĞĹĤ ĴĞĽĽ
    ĽĦĬÌgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd ĠĞĻĢÎçÑã

    ſı OĴĻĨİĨĨŅĦ âŁĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽã
    ĽĦĬÌgIJĴĻĨİĨĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâã
    ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽf
        ĽĦĬÌgIJĴĻĨİĨĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĬĞÑĦĹã
    ĽĦĬÌgIJĴĻĨİĨĨŅĦĹgĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâã

    ſı PĹĨĮĻ Ğ ĽŁİİĞĹÑ
    ĨÌ ĮIJĻ ĽĻĦĴ ķ ĴĹĨĮĻçĦŁĦĹÑ IJĹ ĽĻĦĴ łł ĻĹĞĨĮçĽĻĦĴĽ h ĝf
        ĴĹĨĮĻâÌďĽĻĦĴf ăĽĻĦĴåd ď κ
              ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď κ
              ÌďĤĞĻĞçĬIJĽĽf ăĤĞĻĞçĬIJĽĽfgġÌåd ď κ
              ÌďĹĦÍçĬIJĽĽf ăĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽfgġÌåãd ď κ
              ÌďĬĹf ăĽĦĬÌgIJĴĻĨİĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

ſı GĦĻ ĞĮĤ ĴĹĨĮĻ ĦĴIJĢÎ ĬIJĽĽ ĞĮĤ ĞĢĢŁĹĞĢÑ
ĦĴIJĢÎçĤĞĻĞçĬIJĽĽd ĦĴIJĢÎçĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ ł ċ
    ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦĤâ
        ĬĮĢĬŁĤĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮłTĹŁĦã
ĦĴIJĢÎçĬIJĽĽ ł ĦĴIJĢÎçĤĞĻĞçĬIJĽĽ κ ĦĴIJĢÎçĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ
ĦĴIJĢÎçĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦĤâã

ĴĹĨĮĻâÌďĻĹĞĨĮĨĮÍd ď κ
      ÌďĞĢĢf ăĦĴIJĢÎçĞĢĢŁĹĞĢÑfgġÌåd ď κ
      ÌďĬIJĽĽf ăĦĴIJĢÎçĬIJĽĽfgġÌå âď κ
      ÌďĤĞĻĞçĬIJĽĽf ăĦĴIJĢÎçĤĞĻĞçĬIJĽĽfgġÌåd ď κ
      ÌďĹĦÍçĬIJĽĽf ăĦĴIJĢÎçĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽfgġÌåãd ď κ
      ÌďĬĹf ăĽĦĬÌgIJĴĻĨİĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

ſı IÌ ĻÎĦĹĦ ĨĽ ĻÎĦ ŁĞĬĨĤĞĻĨIJĮ ĤĞĻĞ
ĨÌ ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞ ĨĽ ĮIJĻ NIJĮĦf

    ſı RĦĽĦĻ ĞĢĢŁİŁĬĞĻĦĤ ŁĞĬŁĦĽ ĨĮ ĬIJĽĽ
    ſı ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠÏĦĢĻĽ
    ĽĦĬÌgĬIJĽĽgĮĦŃçĴĞĽĽâã
    ĽĦĬÌgĞĢĢŁĹĞĢÑgĮĦŃçĴĞĽĽâã
```

```
ƒ IĻĦĹĞĻĦ IJL·ĦĹ ĹĻĦĴĽ
ÌIJĹ ĹĻĦĴ ĨĮ ĹĞĮÍĦâL·ĞĬĨĤĞĻĨIJIçĹĻĦĴĽãƒ

    ƒ IÌ ĠĞĻĢÎ ĹĨĢĦ ĨĽ IJIJĻ ĹĦĻ h
    ƒ ĻĹĞĨĮ ŁĹĨĮÍ IJIĦ ĹĻĦĴ ĞĮĤ ÌŁĬÌ ĤĞĻĞĹĦĻ
    ĨÌ ĠĞĻĢÎçĹĨĢĦ ĨĽ NIJIĦƒ
        ĠĞĻĢÎçX ł XçL·ĞĬ
        ĠĞĻĢÎçÑ ł ÑçL·ĞĬ

    ƒ OĻÎĦĹŃĨĹĦ ĹĬĨĢĦ Ğ ĠĞĻĢÎ
    ĦĬĹĦƒ
        ĠĞĻĢÎçX ł XçL·ĞĬä
            ĹĻĦĴåĠĞĻĢÎçĹĨĢĦƒâĹĻĦĴκĝãåĠĞĻĢÎçĹĨĢĦ
        ã
        ĠĞĻĢÎçÑ ł ÑçL·ĞĬä
            ĹĻĦĴåĠĞĻĢÎçĹĨĢĦƒâĹĻĦĴκĝãåĠĞĻĢÎçĹĨĢĦ
        ã

    ƒ PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÎ ĴĞĹĹ
    IJŁĻĴŁĻ ł ĹĦĬÌgÌIJĹŃĞĹÎâĠĞĻĢÎçXd ĻĹĞĨĮĨĮÍłFĞĬĹĦã

    ƒ CĞĬĢŁĬĞĻĦ ĻÎĦ ĬIJĹĹ
    ĹĦĬÌgĬIJĹĹgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ĠĞĻĢÎçÑã

    ƒ GĦĻ ĴĹĦÎĨĢĻĨIJIĮĹ ĞĮÎ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
    ĴĹĦÎĨĢĻĨIJIĮĹ ł ĹĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨL·ĞĻĨIJIĮgĴĹĦÎĨĢĻĨIJIĮĹâ
                        IJŁĻĴŁĻã
    ĹĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦÎĨĢĻĨIJIĮĹd ĠĞĻĢÎçÑã

ƒ GĦĻ ĞĮÎ ĴĹĨĮĻ L·ĞĬĨÎĤĞĻĨIJĮ ĬIJĹĹ ĞĮÎ ĞĢĢŁĹĞĢÑ
L·ĞĬĨÎĤĞĻĨIJIçĬIJĹĹ ł ĹĦĬÌgĬIJĹĹgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦÎâã
L·ĞĬĨÎĤĞĻĨIJIçĞĢĢŁĹĞĢÑ ł ĹĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦÎâã

ƒ PĹĨĮĻ Ğ ĹŁİĬĞĹÑ
ĴĹĨĮĻâÌďL·ĞĬĨÎĤĞĻĨIJId ď κ
      ÌďĞĢĢƒ ăL·ĞĬĨÎĤĞĻĨIJIçĞĢĢŁĹĞĢÑƒfgġÌåd ď κ
      ÌďĬIJĹĹƒ ăL·ĞĬĨÎĤĞĻĨIJIçĬIJĹĹƒfgġÌåďã
```

Training

At this point, we're ready to train using batches and our new dataset. As a reminder, we create the data with:

```
# CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ ł ĢĹĦĞĻĦçĤĞĻĞçİĮĨĽĻâďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽďã
```

Then shuffle with:

```
# SÎŁÌÌĬĦ ĻÎĦ ĻĹĞĨĮĨĮÍ ĤĞĻĞĽĦĻ
ĪĦÑĽ ł ĮĴgĞĹĹĞÑâĹĞĮÍĦâXgĽÎĞĴĦäflāāā
ĮĴgĹĞĮĤIJİgĽÎŁÌÌĬĦâĪĦÑĽã
X ł XäĪĦÑĽā
Ñ ł ÑäĪĦÑĽā
```

Then flatten sample-wise and scale to the range of -1 to 1:

```
# SĢĞĬĦ ĞĮĤ ĹĦĽÎĞĴĦ ĽĞİĴĬĦĽ
X ł âXgĹĦĽÎĞĴĦâXgĽÎĞĴĦäflād hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġĝã h ĝĝìghĥã w ĝĝìghĥ
XçĻĦĽĻ ł âXçĻĦĽĻgĹĦĽÎĞĴĦâXçĻĦĽĻgĽÎĞĴĦäflād hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġĝã h
              ĝĝìghĥã w ĝĝìghĥ
```

Then construct our model consisting of 2 hidden layers using ReLU activation, an output layer with softmax activation since we're building a classification model, cross-entropy loss, Adam optimizer, and categorical accuracy:

```
# IĮĽĻĞĮĻĨĞĻĦ ĻÎĦ İIJĤĦĬ
İIJĤĦĬ ł MIJĤĦĬâã

# AĤĤ ĬĞÑĦĹĽ
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâXgĽÎĞĴĦäĝād ħǵãã
İIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâħǵd ħǵãã
İIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâħǵd ĝflãã
İIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçSIJÌĻİĞŇâãã
```

Set loss, optimizer and accuracy objects:

```
# SHL ĬIJĹĽd IJĴĻĨĬĨŅHĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬHĢĻĽ
ĬIJĤHĬgĽHĻâ
    ĬIJĹĽłLIJĹĽçCĞĻHÍIJĹĨĢĞĬCĹIJĹĽHĮĻĹIJĴÑâãd
    IJĴĻĨĬĨŅHĹłOĴĻĨĬĨŅHĹçAĤĞĬâĤHĢĞÑłĥHHĥãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçCĞĻHÍIJĹĨĢĞĬâã
ã
```

Finally, we finalize and train!

```
# FĨĮĞĬĨŅH ĻÎH ĬIJĤHĬ
ĬIJĤHĬgÌĨĮĞĬĨŅHâã

# TĹĞĨĮ ĻÎH ĬIJĤHĬ
ĬIJĤHĬgĻĹĞĨĮâXd Ñd ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞłâXçĻHĽĻd ÑçĻHĽĻãd
            HĴIJĢÎĽłĥd ĠĞĻĢÎçĽĨŅHłĝĝíd ĴĹĨĮçHŁHĹÑłĝflflã


ńńń
HĴIJĢÎf ĝ
ĽĻHĴf fld ĞĢĢf flgflìíd ĬIJĹĽf ggġflġ âĤĞĻĞçĬIJĹĽf ggġflġd ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflĝ
ĽĻHĴf ĝflfld ĞĢĢf flgìĝîd ĬIJĹĽf flghhfl âĤĞĻĞçĬIJĹĽf flghhfld ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîîĥflĝġíìĥhĝĝíîflì
ĽĻHĴf ĝflfld ĞĢĢf flgìíîd ĬIJĹĽf flgĥhfl âĤĞĻĞçĬIJĹĽf flgĥhfld ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîîflflîîflflîîflflîîflĝ
ĽĻHĴf ġflfld ĞĢĢf flgìíĝd ĬIJĹĽf flghĝĝ âĤĞĻĞçĬIJĹĽf flghĝĝd ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîíĥĝĝĝhìġíìhíġìġ
ĽĻHĴf ġflfld ĞĢĢf flgìíĝd ĬIJĹĽf flgĥĝí âĤĞĻĞçĬIJĹĽf flgĥĝíd ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîíflġîĝĝĥhíhĝìġĥ
ĽĻHĴf ġhíd ĞĢĢf flgíġġd ĬIJĹĽf flgġflfl âĤĞĻĞçĬIJĹĽf flgġflfld ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîììĝġĥflġflflhĝĥġhì
ĻĹĞĨĨĮĨĮÍd ĞĢĢf flgìĝfld ĬIJĹĽf flgìġh âĤĞĻĞçĬIJĹĽf flgìġhd ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîììĝġĥflġflflhĝĥġhì
ŁĞĬĨĤĞĻĨIJĮd ĞĢĢf flgíflĥd ĬIJĹĽf flgĥġì
HĴIJĢÎf ĝ
ĽĻHĴf fld ĞĢĢf flgíĥîd ĬIJĹĽf flgġġġ âĤĞĻĞçĬIJĹĽf flgġġġd ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîììflíìġflĝìĥflĥflflí
ĽĻHĴf ĝflfld ĞĢĢf flgìíîd ĬIJĹĽf flgġìĥ âĤĞĻĞçĬIJĹĽf flgġìĥd ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîìĝġġìflĝĝflflíġhĝ
ĽĻHĴf ĝflfld ĞĢĢf flgíĥîd ĬIJĹĽf flgġĥì âĤĞĻĞçĬIJĹĽf flgġĥìd ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîhìhġĝhíhhġĝĝĥġġ
ĽĻHĴf ġflfld ĞĢĢf flgíġhd ĬIJĹĽf flgġhĝ âĤĞĻĞçĬIJĹĽf flgġhĝd ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflîhĝîìġhhĝhìflġĝĥî
ĽĻHĴf ġflfld ĞĢĢf flgìíîd ĬIJĹĽf flgġġì âĤĞĻĞçĬIJĹĽf flgġġìd ĹHÍçĬIJĹĽf flgflflflãd ĬĹf
```

flgflflflîĥíġĥîğíííîìġflìħ
ĽĻĦĴf ġĥíd ĞĢĢf flgííĥd ĬIJĽĽf flgġğġ âĤĞĻĞçĬIJĽĽf flgġğġd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîĥĥğġĥĥîğġĥíġğìġ
ĻĹĞĬĨĮĨĮÍd ĞĢĢf flgíġğd ĬIJĽĽf flgġĥĝ âĤĞĻĞçĬIJĽĽf flgġĥĝd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîĥĥğġĥĥîğġĥíġğìġ
ĽĞĬĨĤĞĻĨIJĮd ĞĢĢf flgíġĥd ĬIJĽĽf flgġĥí
ĦĴIJĢÎf ġ
ĽĻĦĴf fld ĞĢĢf flgíĥîd ĬIJĽĽf flgġííì âĤĞĻĞçĬIJĽĽf flgġííd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîĥĥğflĝflĥîíğĥĝîíġ
ĽĻĦĴf ĝflfld ĞĢĢf flgíğfld ĬIJĽĽf flgġġġ âĤĞĻĞçĬIJĽĽf flgġġġd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîĥflĥĥflìflîĝîğííîĝ
ĽĻĦĴf ğflfld ĞĢĢf flgíĥîd ĬIJĽĽf flgġğfl âĤĞĻĞçĬIJĽĽf flgġğfld ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîġĥĝĥġġflìííĥîğġĝ
ĽĻĦĴf ġflfld ĞĢĢf flgíĥîd ĬIJĽĽf flgġğġ âĤĞĻĞçĬIJĽĽf flgġğġd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîġĝìflíğĥíííĝġĝîĥ
ĽĻĦĴf ġflfld ĞĢĢf flgíĝğd ĬIJĽĽf flgġîġ âĤĞĻĞçĬIJĽĽf flgġîġd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîġìğîġîĥĥìğĥflğíì
ĽĻĦĴf ġĥíd ĞĢĢf flgííìĥd ĬIJĽĽf flgğíĥ âĤĞĻĞçĬIJĽĽf flgğíĥd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîġġġĝìġíĝflíflflìĝ
ĻĹĞĬĨĮĨĮÍd ĞĢĢf flgíĥĝd ĬIJĽĽf flgġflì âĤĞĻĞçĬIJĽĽf flgġflìd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîġġġĝìġíĝflíflflìĝ
ĽĞĬĨĤĞĻĨIJĮd ĞĢĢf flgíġìd ĬIJĽĽf flgġğğ
ĦĴIJĢÎf ġ
ĽĻĦĴf fld ĞĢĢf flgíĥîd ĬIJĽĽf flgġĥfl âĤĞĻĞçĬIJĽĽf flgġĥfld ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîġġğìġíġĥĥĥĝğġğġ
ĽĻĦĴf ĝflfld ĞĢĢf flgíğíd ĬIJĽĽf flgġîí âĤĞĻĞçĬIJĽĽf flgġîíd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîğîîğîìîflġflflíġğġ
ĽĻĦĴf ğflfld ĞĢĢf flgíĥìd ĬIJĽĽf flgġĝfl âĤĞĻĞçĬIJĽĽf flgġĝfld ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîğĥĥğĥîĥġĥĥĝìĥĥì
ĽĻĦĴf ġflfld ĞĢĢf flgííĝd ĬIJĽĽf flgġîġ âĤĞĻĞçĬIJĽĽf flgġîġd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîğĝĝĥĝììğìflflflĥflĥ
ĽĻĦĴf ġflfld ĞĢĢf flgíġĥd ĬIJĽĽf flgġĥġ âĤĞĻĞçĬIJĽĽf flgġĥġd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîĝìĝĝĥĥîîğĝĥflĝîĥ
ĽĻĦĴf ġĥíd ĞĢĢf flgííĥd ĬIJĽĽf flgğĥġ âĤĞĻĞçĬIJĽĽf flgğĥġd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîĝġğíĥìĝĝğíĥìĝġġ
ĻĹĞĬĨĮĨĮÍd ĞĢĢf flgíĥğd ĬIJĽĽf flgġìí âĤĞĻĞçĬIJĽĽf flgġìíd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîĝġğíĥìĝĝğíĥìĝġġ
ĽĞĬĨĤĞĻĨIJĮd ĞĢĢf flgíĥĥd ĬIJĽĽf flgġflġ
ĦĴIJĢÎf ĥ
ĽĻĦĴf fld ĞĢĢf flgíġĥd ĬIJĽĽf flgġġġ âĤĞĻĞçĬIJĽĽf flgġġġd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîĝġğġġîğflğìííġflğ
ĽĻĦĴf ĝflfld ĞĢĢf flgíğíd ĬIJĽĽf flgġĥí âĤĞĻĞçĬIJĽĽf flgġĥíd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîĝflflíġìğììflğîġíì
ĽĻĦĴf ğflfld ĞĢĢf flgíĥìd ĬIJĽĽf flgġflì âĤĞĻĞçĬIJĽĽf flgġflìd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîflĥîĥĝğġğíĥîĥìĥî
ĽĻĦĴf ġflfld ĞĢĢf flgííĝd ĬIJĽĽf flgġífl âĤĞĻĞçĬIJĽĽf flgġífld ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflîflĝíìĥîflĝíìĥîflĝî
ĽĻĦĴf ġflfld ĞĢĢf flgíĥîd ĬIJĽĽf flgġğğ âĤĞĻĞçĬIJĽĽf flgġğğd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflííìíğìğĥíflġĥĥĥġĝ

```
ĽĻĦĴf ġħíd ĞĢĢf flgííĥd ĬIJĽĽf flgğġĝ âĤĞĻĞçĬIJĽĽf flgğġĝd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflííĥflîġííflflĥìğíħĝ
ĻĹĞĬĮĬĮÍd ĞĢĢf flgíħîd ĬIJĽĽf flgġĥì âĤĞĻĞçĬIJĽĽf flgġĥìd ĹĦÍçĬIJĽĽf flgflflflãd ĬĹf
flgflflflííĥflîġííflflĥìğíħĝ
ĻĞĬĬĤĞĻĬIJĮd ĞĢĢf flgíħfld ĬIJĽĽf flgġííî
```

The model trained successfully and achieved pretty good accuracy. This was done with a new, real, much more challenging dataset and in just 5 epochs instead of 10000. Training also went faster than with our previous attempts at spiral data, where we trained by fitting the whole dataset at once.

So far, we've only mentioned how important it is to shuffle the training data and what might happen if we attempt to train on non-shuffled data. Now would be a good time to exemplify what happens when we don't shuffle it. We can comment out the shuffling code:

```
# SÎŁÌÌĬĦ ĻÎĦ ĻĹĞĬĮĬĮÍ ĤĞĻĞĽĦĻ
# ĪĦÑĽ ł ĮĴgĞĹĹĞÑâĹĞĮÍĦâXgĽÎĞĴĦäflāāā
# ĮĴgĹĞĮĤIJİgĽÎŁÌÌĬĦâĪĦÑĽā
# X ł XäĪĦÑĽā
# Ñ ł ÑäĪĦÑĽā
```

Running again, we can see that we end on:

```
ńńń
ĦĴIJĢÎf ĝ
ĽĻĦĴf fld ĞĢĢf flgflflfld ĬIJĽĽf ğgġflğ âĤĞĻĞçĬIJĽĽf ğgġflğd ĹĦÍçĬIJĽĽf flgflflfld ĬĹf
flgflflĝ
ĽĻĦĴf ĝflfld ĞĢĢf flgflflfld ĬIJĽĽf ğgġgí âĤĞĻĞçĬIJĽĽf ğgġgíd ĹĦÍçĬIJĽĽf flgflflfld ĬĹf
flgflflflííĥflğgíìĥĦğĝíîflì
ĽĻĦĴf ğflfld ĞĢĢf flgflflfld ĬIJĽĽf ğgġflĝ âĤĞĻĞçĬIJĽĽf ğgġflĝd ĹĦÍçĬIJĽĽf flgflflfld ĬĹf
flgflflflîîflflîîflflîîflflîîflĝ
ĽĻĦĴf ġflfld ĞĢĢf flgflflfld ĬIJĽĽf ğgğĝġ âĤĞĻĞçĬIJĽĽf ğgğĝġd ĹĦÍçĬIJĽĽf flgflflfld ĬĹf
flgflflfliíĥğğĝĥìġíìĥíġìġ
ĽĻĦĴf ġflfld ĞĢĢf flgflflfld ĬIJĽĽf ğgğìí âĤĞĻĞçĬIJĽĽf ğgğìíd ĹĦÍçĬIJĽĽf flgflflfld ĬĹf
flgflflfliíflġîğĝĥĥíĥğìġĥ
ĽĻĦĴf ġħíd ĞĢĢf ĝgflflfld ĬIJĽĽf flgflĝí âĤĞĻĞçĬIJĽĽf flgflĝíd ĹĦÍçĬIJĽĽf flgflflfld ĬĹf
flgflflfliììĝġĥflġflflĥğĥġĥì
ĻĹĞĬĮĬĮÍd ĞĢĢf flgġíĝd ĬIJĽĽf ğgğġĥ âĤĞĻĞçĬIJĽĽf ğgğġĥd ĹĦÍçĬIJĽĽf flgflflfld ĬĹf
flgflflfliììĝġĥflġflflĥğĥġĥì
ĻĞĬĬĤĞĻĬIJĮd ĞĢĢf flgĝflfld ĬIJĽĽf ĥgîíğ
ĦĴIJĢÎf ğ
ĽĻĦĴf fld ĞĢĢf flgflflfld ĬIJĽĽf íğğflĝ âĤĞĻĞçĬIJĽĽf íğğflĝd ĹĦÍçĬIJĽĽf flgflflfld ĬĹf
```

flgflflflîììflíìġflğìĥflĥflflí
ĽĻĦĴf ĝflfld ĞĢĢf flgflflfld ĬIJĽĽf ġgĥìì âĤĞĻĞçĬIJĽĽf ġgĥììd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîìğġġìflĝğflflíġħğ
ĽĻĦĴf ğflfld ĞĢĢf flgġíġd ĬIJĽĽf ĝgíğĝ âĤĞĻĞçĬIJĽĽf ĝgíğĝd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĥìĥġğĥíĥĥġğĝĥġġ
ĽĻĦĴf ġflfld ĞĢĢf flgflflfld ĬIJĽĽf flgîĥġ âĤĞĻĞçĬIJĽĽf flgîĥġd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĥğîìġĥĥğĥìflġğĥî
ĽĻĦĴf ġflfld ĞĢĢf flgflflfld ĬIJĽĽf ĝgĥġĥ âĤĞĻĞçĬIJĽĽf ĝgĥġĥd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĥíġĥîğíííìġflìĥ
ĽĻĦĴf ġĥíd ĞĢĢf ĝgflflfld ĬIJĽĽf flgflĝġ âĤĞĻĞçĬIJĽĽf flgflĝġd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĥĥğġĥĥîğġĥíġġìġ
ĻĹĞĬĨĨĮĬd ĞĢĢf flgĥîìd ĬIJĽĽf ĝgĥìġ âĤĞĻĞçĬIJĽĽf ĝgĥìġd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĥĥğġĥĥîğġĥíġġìġ
ŁĞĬĨĨĤĞĻĨIJĮd ĞĢĢf flgĝflîd ĬIJĽĽf ġgîĝì
ĦĴIJĢÎf ġ
ĽĻĦĴf fld ĞĢĢf flgflflfld ĬIJĽĽf ġgġġĝ âĤĞĻĞçĬIJĽĽf ġgġġĝd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĥĥğflĝflĥîíġĥĝîíġ
ĽĻĦĴf ĝflfld ĞĢĢf flgflflfld ĬIJĽĽf ġgĥĝî âĤĞĻĞçĬIJĽĽf ġgĥĝîd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĥflĥĥlìflîĝîğííîĝ
ĽĻĦĴf ğflfld ĞĢĢf flgíĥîd ĬIJĽĽf flgĥĥî âĤĞĻĞçĬIJĽĽf flgĥĥîd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîġĥĝĥġġflìíĥîğġĝ
ĽĻĦĴf ġflfld ĞĢĢf ĝgflflfld ĬIJĽĽf flgğğĥ âĤĞĻĞçĬIJĽĽf flgğğĥd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîġĝìflíğĥíìíĝġğîĥ
ĽĻĦĴf ġflfld ĞĢĢf ĝgflflfld ĬIJĽĽf flgĝĥĝ âĤĞĻĞçĬIJĽĽf flgĝĥĝd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîġìğîġîĥhìğĥflğíì
ĽĻĦĴf ġĥíd ĞĢĢf ĝgflflfld ĬIJĽĽf flgflĝğ âĤĞĻĞçĬIJĽĽf flgflĝğd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîġġġĝìġíĝflíflflìĝ
ĻĹĞĬĨĨĮĬd ĞĢĢf flgĥġíd ĬIJĽĽf ĝgġìí âĤĞĻĞçĬIJĽĽf ĝgġìíd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîġġġĝìġíĝflíflflìĝ
ŁĞĬĨĨĤĞĻĨIJĮd ĞĢĢf flgĝġġd ĬIJĽĽf ġgĝflí
ĦĴIJĢÎf ġ
ĽĻĦĴf fld ĞĢĢf flgflġĝd ĬIJĽĽf ğgĥğfl âĤĞĻĞçĬIJĽĽf ğgĥğfld ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîġġğìġíġĥĥĥĝğġğġ
ĽĻĦĴf ĝflfld ĞĢĢf flgflflfld ĬIJĽĽf ġgĝğí âĤĞĻĞçĬIJĽĽf ġgĝğíd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîğîîğîìîflġflflíġğġ
ĽĻĦĴf ğflfld ĞĢĢf flgflflfld ĬIJĽĽf ĝgíîĝ âĤĞĻĞçĬIJĽĽf ĝgíîĝd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîğĥĥğĥîĥġĥĥĝìĥĥì
ĽĻĦĴf ġflfld ĞĢĢf ĝgflflfld ĬIJĽĽf flgĝĝí âĤĞĻĞçĬIJĽĽf flgĝĝíd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîğĝġĥĝìğìflflflĥflĥ
ĽĻĦĴf ġflfld ĞĢĢf ĝgflflfld ĬIJĽĽf flgflĥĥ âĤĞĻĞçĬIJĽĽf flgflĥĥd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĝìĝġĥîîğğĥflĝîĥ
ĽĻĦĴf ġĥíd ĞĢĢf ĝgflflfld ĬIJĽĽf flgflĝĝ âĤĞĻĞçĬIJĽĽf flgflĝĝd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĝġğíĥìĝġğíĥìĝġġ
ĻĹĞĬĨĨĮĬd ĞĢĢf flgĥġġd ĬIJĽĽf ĝgġġĥ âĤĞĻĞçĬIJĽĽf ĝgġġĥd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĝġğíĥìĝġğíĥìĝġġ
ŁĞĬĨĨĤĞĻĨIJĮd ĞĢĢf flgĝííd ĬIJĽĽf ġgflĥfl
ĦĴIJĢÎf ĥ
ĽĻĦĴf fld ĞĢĢf flgflĝĥd ĬIJĽĽf ğgìġġ âĤĞĻĞçĬIJĽĽf ğgìġġd ĹĦÍçĬIJĽĽf flgflflfãd ĬĹf
flgflflflîĝġğġġîğflğìîîġflğ

```
ĹĻĦĴf ĝflfld ĞĢĢf flgflflfld ĬIJĹĹf ğgíġí âĤĞĻĞçĬIJĹĹf ğgíġíd ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflflîĝflflíġìğììflğîġíì
ĹĻĦĴf ğflfld ĞĢĢf flgĥġìd ĬIJĹĹf ĝgĝflí âĤĞĻĞçĬIJĹĹf ĝgĝflíd ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflflîflĥîħĝğğğíĥîĥìĥî
ĹĻĦĴf ġflfld ĞĢĢf flgîîğd ĬIJĹĹf flgflĝí âĤĞĻĞçĬIJĹĹf flgflĝíd ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflflîflĝíìĥîflĝíìĥîflĝî
ĹĻĦĴf ǵflfld ĞĢĢf ĝgflflfld ĬIJĹĹf flgflħĥ âĤĞĻĞçĬIJĹĹf flgflħĥd ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflflíîìíğìğĥíflġĥĥĥġĝ
ĹĻĦĴf ǵħíd ĞĢĢf ĝgflflfld ĬIJĹĹf flgflĝfl âĤĞĻĞçĬIJĹĹf flgflĝfld ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflflíîĥflîġííflflĥìğíħĝ
ĻĹĞĬĬĮĬĮÍd ĞĢĢf flgìǵǵd ĬIJĹĹf flgîħĝ âĤĞĻĞçĬIJĹĹf flgîħĝd ĹĦÍçĬIJĹĹf flgflflfãd ĬĹf
flgflflflíîĥflîġííflflĥìğíħĝ
ĻĞĬĬĤĞĻĬIJĮd ĞĢĢf flgğflfld ĬIJĹĹf ġgġĝĝ
```

As we can see, this doesn't work well at all. We can observe how the model approached a perfect accuracy of 1 during training, but epoch accuracy remained poor, and the validation accuracy proved that the model did not learn. Training accuracy quickly became high, since the model learned to predict just one label (as it repeatedly saw only that label). Once the label changed in the training data, the model quickly learned to predict only that new label, as that's all it saw in every batch. This process repeated to the end of an epoch and then over all epochs. Epoch accuracy is lower because it took a while for the model to learn the new label after a switch, and it showed a low accuracy during this period. Validation accuracy was calculated after training for a given epoch ended, and as we remember, the model learned to predict just one label. In the case of validation, the label that the model predicted was the last label it had seen — its accuracy was close to 1/10 as our training dataset consists of 10 classes.

Re-enable shuffling, and then you can tinker around with your model to see if you can further improve results. Here is an example with a larger model, a higher learning rate decay, and twice as many epochs:

```
fi AĤĤ ĬĞÑĦĹĽ
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâXgĽĬĞĴĦäĝãd ĝğíãã
ĬIJĤĦĬgĞĤĤâAĢĻĬĻĞĻĬIJĮçRĦLUâãã
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝğíãã
ĬIJĤĦĬgĞĤĤâAĢĻĬĻĞĻĬIJĮçRĦLUâãã
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝflãã
ĬIJĤĦĬgĞĤĤâAĢĻĬĻĞĻĬIJĮçSIJÌĻĬĞÑâãã

fi SĦĻ ĬIJĹĽd IJĴĻĬĬĬÑĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĽ
ĬIJĤĦĬgĽĦĻâ
    ĬIJĹĽŁLIJĹĽçCĞĻĦÍIJĹĬĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâãd
    IJĴĻĬĬĬÑĦĹŁOĴĻĬĬĬÑĦĹçAĤĞĬâĤĦĢĞÑŁĝĦhġãd
    ĞĢĢŁĹĞĢÑŁAĢĢŁĹĞĢÑçCĞĻĦÍIJĹĬĢĞĬâã
ã
```

```
ń FĨĮĞĬĨŅĦ ĻÎĦ ĬIJÂĦĬ
ĬIJÂĦĬgÌĨĮĞĬĨŅĦâã

ń TĹĞĨĮ ĻÎĦ ĬIJÂĦĬ
ĬIJÂĦĬgĻĹĞĨĮâXd Ñd ŁĞĬĨÂĞĻĨIJĮçÂĞĻĞłâXçĻĦĹĻd ÑçĻĦĹĻãd
            ĦĴIJĢÎĹłĝfld ĠĞĻĢÎçĹĨŅĦłĝĝíd ĴĹĨĮĻçĦŁĦĹÑłĝflflã


ńńń
ggg
ĦĴIJĢÎf ĝfl
ĹĻĦĴf fld ĞĢĢf flgíîĝd ĬIJĹĹf flgĝĥġ âÂĞĻĞçĬIJĹĹf flgĝĥġd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflflĝîĝĥġĝĝíííĥĝìĝflĝ
ĹĻĦĴf ĝflfld ĞĢĢf flgííġd ĬIJĹĹf flgĝĥì âÂĞĻĞçĬIJĹĹf flgĝĥìd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflflĝíìîġĝĥîííìĥîĥġĝĥì
ĹĻĦĴf ĝflfld ĞĢĢf flgîĝĝd ĬIJĹĹf flgĝĝì âÂĞĻĞçĬIJĹĹf flgĝĝìd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflflĝíġĝĥìíĝflġĥìflîflîġ
ĹĻĦĴf ġflfld ĞĢĢf flgíííd ĬIJĹĹf flgĝíĝ âÂĞĻĞçĬIJĹĹf flgĝíĝd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflflĝíĝĝĝĥĥflìġîíĥġĝĥĥ
ĹĻĦĴf ġflfld ĞĢĢf flgîĝĝd ĬIJĹĹf flgĝîî âÂĞĻĞçĬIJĹĹf flgĝîîd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflflĝìîîflġĝíìġîġĝĝíĥĥ
ĹĻĦĴf ġĥíd ĞĢĢf flgîĝìd ĬIJĹĹf flgĝîĝ âÂĞĻĞçĬIJĹĹf flgĝîĝd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflflĝìĥììíĝĥíġîĥĝĝíĥ
ĻĹĞĨĮĨĮÍd ĞĢĢf flgííġd ĬIJĹĹf flgĝîĝ âÂĞĻĞçĬIJĹĹf flgĝîĝd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflflĝìĥììíĝĥíġîĥĝĝíĥ
ŁĞĬĨÂĞĻĨIJĮd ĞĢĢf flgíìġd ĬIJĹĹf flgġĥġ
```

We improved accuracy and decreased loss a bit by simply increasing the model size, decay, and number of epochs.

Full code up to now:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ
ĨİĴIJĹĻ IJĽ
ĨİĴIJĹĻ ĢŁğ

ĮĮÌĽgĨĮĨĻâã


ſı DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ſı LĞÑĦĹ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽd
                ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝłfld ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğłfld
                ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝłfld ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğłflãf
        ſı IĮĨĻĨĞĬĨŅĦ ŃĦĨÍÎĻĽ ĞĮÂ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍÎĻĽ ł flgflĝ ǻ ĮĴgĹĞĮÂIJİgĹĞĮÂIJâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgŅĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã
        ſı SĦĻ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĽĻĹĦĮÍĻÎ
        ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ł ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ
        ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ł ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ
        ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ł ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ
        ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ł ĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ

    ſı FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ſı RĦİĦİĠĦĹ ĨĮĴŁĻ ŁĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ſı CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ŁĞĬŁĦĽ ÌĹIJİ ĨĮĴŁĻĽd ŃĦĨÍÎĻĽ ĞĮÂ ĠĨĞĽĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgÂIJĻâĨĮĴŁĻĽd ĽĦĬÌgŃĦĨÍÎĻĽã ĸ ĽĦĬÌgĠĨĞĽĦĽ

    ſı BĞĢĨŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ĠĞĢĨŃĞĹÂâĽĦĬÌd ÂŁĞĬŁĦĽãf
        ſı GĹĞÂĨĦĮĻĽ IJĮ ĴĞĹĞĬĦĻĦĹĽ
        ĽĦĬÌgÂŃĦĨÍÎĻĽ ł ĮĴgÂIJĻâĽĦĬÌgĨĮĴŁĻĽgTd ÂŁĞĬŁĦĽã
        ĽĦĬÌgÂĠĨĞĽĦĽ ł ĮĴgĽŁİâÂŁĞĬŁĦĽd ĞÑĨĽłfld ĨĦĦĴÂĨİĽłTĹŁĦã
```

```
        # GĹĞĤĨĦĮĻĽ IJĮ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ
        # Lĝ IJĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń ƒf
            ĤLĝ ł ĮĴgIJĮĦĽçĬĨĨĦâĽĦĬÌgŃĦĨÍÎĻĽã
            ĤLĝäĽĦĬÌgŃĦĨÍÎĻĽ l· ƒlā ł hĝ
            ĽĦĬÌgĤŃĦĨÍÎĻĽ κł ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ő ĤLĝ
        # Lğ IJĮ ŃĦĨÍÎĻĽ
        ĨÌ ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ƒf
            ĽĦĬÌgĤŃĦĨÍÎĻĽ κł ğ ő ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                ĽĦĬÌgŃĦĨÍÎĻĽ
        # Lĝ IJĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń ƒf
            ĤLĝ ł ĮĴgIJĮĦĽçĬĨĨĦâĽĦĬÌgĠĨĞĽĦĽã
            ĤLĝäĽĦĬÌgĠĨĞĽĦĽ l· ƒlā ł hĝ
            ĽĦĬÌgĤĠĨĞĽĦĽ κł ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ő ĤLĝ
        # Lğ IJĮ ĠĨĞĽĦĽ
        ĨÌ ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń ƒf
            ĽĦĬÌgĤĠĨĞĽĦĽ κł ğ ő ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨŅĦĹçĬğ ő ċ
                                ĽĦĬÌgĠĨĞĽĦĽ

        # GĹĞĤĨĦĮĻ IJĮ L·ĞĬŁĦĽ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĮĴgĤIJĻâĤL·ĞĬŁĦĽd ĽĦĬÌgŃĦĨÍÎĻĽgTã


# DĹIJĴIJŁĻ
ĢĬĞĽĽ LĞÑĦĹçDĹIJĴIJŁĻf

    # IĮĨĻ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĹĞĻĦãf
        # SĻIJĹĦ ĹĞĻĦd ŃĦ ĨĮL·ĦĹĻ ĨĻ ĞĽ ÌIJĹ ĦŃĞĬĴĬĦ ÌIJĹ ĤĹIJĴIJŁĻ
        # IJÌ ƒlgĝ ŃĦ ĮĦĦĤ ĽŁĢĢĦĽĽ ĹĞĻĦ IJÌ ƒlgî
        ĽĦĬÌgĹĞĻĦ ł ĝ h ĹĞĻĦ

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        # SĞL·Ħ ĨĮĴŁĻ L·ĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ

        # IÌ ĮIJĻ ĨĮ ĻĤĦ ĻĹĞĨĮĨĮÍ ĬIJĤĦ h ĹĦĻŁĹĮ L·ĞĬŁĦĽ
        ĨÌ ĮIJĻ ĻĹĞĨĮĨĮÍf
            ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽgĢIJĴÑâã
            ĹĦĻŁĹĮ

        # GĦĮĦĹĞĻĦ ĞĮĤ ĽĞL·Ħ ĽĢĞĬĦĤ ĬĞĽĨ
        ĽĦĬÌgĠĨĮĞĹÑçĬĞĽĨ ł ĮĴgĹĞĮĤIJĬgĠĨĮIJĬĨĞĬâĝd ĽĦĬÌgĹĞĻĦd
                            ĹĨĢĦŁĨĮĴŁĻĽgĹĨĞĴĦã w ĽĦĬÌgĹĞĻĦ
        # AĴĴĬÑ ĬĞĽĨ ĻIJ IJŁĻĴŁĻ L·ĞĬŁĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽ ő ĽĦĬÌgĠĨĮĞĹÑçĬĞĽĨ
```

```
    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ÂĽĞĬŁHĽãf
        # GĹĞĤĨHĮĻ IJĮ ĽĞĬŁHĽ
        ĽHĬÌgĤĨĮĴŁĻĽ ł ÂĽĞĬŁHĽ ǻ ĽHĬÌgĠĨĮĞĹÑçĬĞĽĪ


# IĮĴŁĻ ðĬĞÑHĹð
ĢĬĞĽĽ LĞÑHĹçIĮĴŁĻf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ĽHĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽ


# RHLU ĞĢĻĨĽĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĽĞĻĨIJĮçRHLUf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        # RHİHİĠHĹ ĨĮĴŁĻ ĽĞĬŁHĽ
        ĽHĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        # CĞĬĢŁĬĞĻH IJŁĻĴŁĻ ĽĞĬŁHĽ ÌĹIJİ ĨĮĴŁĻĽ
        ĽHĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĬİŁİâfld ĨĮĴŁĻĽã

    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÂHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ÂĽĞĬŁHĽãf
        # SĨĮĢH ŃH ĮHHÂ ĻIJ İIJÂĨÌÑ IJĹĨÍĨĮĞĬ ĽĞĹĨĞĠĬHd
        # ĬHĻďĽ İĞĪH Ğ ĢIJĴÑ IJÌ ĽĞĬŁHĽ ÌĨĹĽĻ
        ĽHĬÌgÂĨĮĴŁĻĽ ł ÂĽĞĬŁHĽgĢIJĴÑâã

        # ZHĹIJ ÍĹĞĤĨHĮĻ ŃÎHĹH ĨĮĴŁĻ ĽĞĬŁHĽ ŃHĹH ĮHÍĞĻĨĽH
        ĽHĬÌgÂĨĮĴŁĻĽäĽHĬÌgĨĮĴŁĻĽ l·ł flã ł fl

    # CĞĬĢŁĬĞĻH ĴĹHÂĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ÂHÌ ĴĹHÂĨĢĻĨIJĮĽâĽHĬÌd IJŁĻĴŁĻĽãf
        ĹHĻŁĹĮ IJŁĻĴŁĻĽ


# SIJÌĻİĞŃ ĞĢĻĨĽĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĽĞĻĨIJĮçSIJÌĻİĞŃf

    # FIJĹŃĞĹĤ ĴĞĽĽ
    ÂHÌ ÌIJĹŃĞĹĤâĽHĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        # RHİHİĠHĹ ĨĮĴŁĻ ĽĞĬŁHĽ
        ĽHĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ

        # GHĻ ŁĮĮIJĹİĞĬĨĨŅHHÂ ĴĹIJĠĞĠĨĬĨĨĻĨHĽ
        HŃĴçĽĞĬŁHĽ ł ĮĴgHŃĴâĨĮĴŁĻĽ h ĮĴgİĞŃâĨĮĴŁĻĽd ĞŃĨĽłĝd
                                                ĪHHĴÂĨİĽłTĹŁHãã
```

```
        ƒ NIJĹĬĞĬĨŅĦ ĻÎĦİ ÌIJĹ ĦĞĢÎ ĽĞĬĴĬĦ
        ĴĹIJĠĞĠĬĬĨĻĨĦĽ ł ĦŇĴçĻĞĬłĦĽ w ĮĴgĽłİâĦŇĴçĻĞĬłĦĽd ĞŇĨĽłĝd
                                                ĪĦĦĴĤĨİĽłTĹłĦã

        ĽĦĬÌgIJłĻĴłĻ ł ĴĹIJĠĞĠĬĬĨĻĨĦĽ

    ƒ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĻĞĬłĦĽãf

        ƒ CĹĦĞĻĦ łĮĨĮĨĻĨĞĬĨŅĦĤ ĞĹĹĞÑ
        ĽĦĬÌgĤĨĮĴłĻĽ ł ĮĴgĦİĴĻÑçĬĨĨĦâĤĻĞĬłĦĽã

        ƒ EĮłİĦĹĞĻĦ IJłĻĴłĻĽ ĞĮĤ ÍĹĞĤĨĦĮĻĽ
        ÌIJĹ ĨĮĤĦÑd âĽĨĮÍĬĦçIJłĻĴłĻd ĽĨĮÍĬĦçĤĻĞĬłĦĽã ĨĮ ċ
                ĦĮłİĦĹĞĻĦâŅĨĴâĽĦĬÌgIJłĻĴłĻd ĤĻĞĬłĦĽããf
            ƒ FĬĞĻĻĦĮ IJłĻĴłĻ ĞĹĹĞÑ
            ĽĨĮÍĬĦçIJłĻĴłĻ ł ĽĨĮÍĬĦçIJłĻĴłĻgĹĦĽĬĞĴĦâhĝd ĝã
            ƒ CĞĬĢłĬĞĻĦ JĞĢIJĠĨĞĮ İĞĻĹĨŇ IJÌ ĻÎĦ IJłĻĴłĻ ĞĮĤ
            ÏĞĢIJĠĨĞĮçİĞĻĹĨŇ ł ĮĴgĤĨĞĹÌĬĞĻâĽĨĮÍĬĦçIJłĻĴłĻã h ċ
                                ĮĴgĤIJĻâĽĨĮÍĬĦçIJłĻĴłĻd ĽĨĮÍĬĦçIJłĻĴłĻgTã
            ƒ CĞĬĢłĬĞĻĦ ĽĞĬĴĬĦhŃĨĽĦ ÍĹĞĤĨĦĮĻ
            ƒ ĞĮĤ ĞĤĤ ĨĻ ĻIJ ĻÎĦ ĞĹĹĞÑ IJÌ ĽĞĬĴĬĦ ÍĹĞĤĨĦĮĻĽ
            ĽĦĬÌgĤĨĮĴłĻĽäĨĮĤĦŇã ł ĮĴgĤIJĻâÏĞĢIJĠĨĞĮçİĞĻĹĨŇd
                                        ĽĨĮÍĬĦçĤĻĞĬłĦĽã

    ƒ CĞĬĢłĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĽ ÌIJĹ IJłĻĴłĻĽ
    ĤĦÌ ĴĹĦĤĨĢĻĨIJĮĽâĽĦĬÌd IJłĻĴłĻĽãf
        ĹĦĻłĹĮ ĮĴgĞĹÍİĞŇâIJłĻĴłĻĽd ĞŇĨĽłĝã


ƒ SĨÍİIJĨĤ ĞĢĻĨĻĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĻĞĻĨIJĮçSĨÍİIJĨĤf

    ƒ FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴłĻĽd ĻĹĞĨĮĨĮÍãf
        ƒ SĞĻĦ ĨĮĴłĻ ĞĮĤ ĢĞĬĢłĬĞĻĦwĽĞĻĦ IJłĻĴłĻ
        ƒ IJÌ ĻÎĦ ĽĨÍİIJĨĤ ÌłĮĢĻĨIJĮ
        ĽĦĬÌgĨĮĴłĻĽ ł ĨĮĴłĻĽ
        ĽĦĬÌgIJłĻĴłĻ ł ĝ w âĝ κ ĮĴgĦŇĴâhĨĮĴłĻĽãã

    ƒ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĻĞĬłĦĽãf
        ƒ DĦĹĨĻĞĻĨĻĦ h ĢĞĬĢłĬĞĻĦĽ ÌĹIJİ IJłĻĴłĻ IJÌ ĻÎĦ ĽĨÍİIJĨĤ ÌłĮĢĻĨIJĮ
        ĽĦĬÌgĤĨĮĴłĻĽ ł ĤĻĞĬłĦĽ ǻ âĝ h ĽĦĬÌgIJłĻĴłĻã ǻ ĽĦĬÌgIJłĻĴłĻ

    ƒ CĞĬĢłĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĽ ÌIJĹ IJłĻĴłĻĽ
    ĤĦÌ ĴĹĦĤĨĢĻĨIJĮĽâĽĦĬÌd IJłĻĴłĻĽãf
        ĹĦĻłĹĮ âIJłĻĴłĻĽ ń flgĥã ǻ ĝ
```

```
# LĨĮĦĞĹ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĹĹ AĢĻĨĿĞĻĨIJĮçLĨĮĦĞĹf

    # FIJĹŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ÌIJĹŃĞĹÂâĹĦĬÌd ĨĮĴŁĻĹd ĻĹĞĨĮĨĮÍãf
        # JŁĹĻ ĹĦİĦİĠĦĹ ĿĞĬŁĦĹ
        ĹĦĬÌgĨĮĴŁĻĹ ł ĨĮĴŁĻĹ
        ĹĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĹ

    # BĞĢĪŃĞĹÂ ĴĞĹĹ
    ÂĦÌ ĠĞĢĪŃĞĹÂâĹĦĬÌd ÂĿĞĬŁĦĹãf
        # ÂĦĹĨĿĞĻĨĿĦ ĨĹ ĝd ĝ ǻ ÂĿĞĬŁĦĹ ł ÂĿĞĬŁĦĹ h ĻÎĦ ĢÎĞĨĮ ĹŁĬĦ
        ĹĦĬÌgÂĨĮĴŁĻĹ ł ÂĿĞĬŁĦĹgĢIJĴÑâã

    # CĞĬĢŁĬĞĻĦ ĴĹĦÂĨĢĻĨIJĮĹ ÌIJĹ IJŁĻĴŁĻĹ
    ÂĦÌ ĴĹĦÂĨĢĻĨIJĮĹâĹĦĬÌd IJŁĻĴŁĻĹãf
        ĹĦĻŁĹĮ IJŁĻĴŁĻĹ


# SGD IJĴĻĨİĨŅĦĹ
ĢĬĞĹĹ OĴĻĨİĨŅĦĹçSGDf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮÍĹd
    # ĬĦĞĹĮĨĮÍ ĹĞĻĦ IJÌ ĝg ĨĹ ÂĦÌĞŁĬĻ ÌIJĹ ĻÎĨĹ IJĴĻĨİĨŅĦĹ
    ÂĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ÂĦĢĞÑłflgd İIJİĦĮĻŁİłflgãf
        ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgÂĦĢĞÑ ł ÂĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgİIJİĦĮĻŁİ ł İIJİĦĮĻŁİ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴÂĞĻĦĹ
    ÂĦÌ ĴĹĦçŁĴÂĞĻĦçĴĞĹĞİĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgÂĦĢĞÑf
            ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ǻ ċ
                âĝg w âĝg κ ĹĦĬÌgÂĦĢĞÑ ǻ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    # UĴÂĞĻĦ ĴĞĹĞİĦĻĦĹĹ
    ÂĦÌ ŁĴÂĞĻĦçĴĞĹĞİĹâĹĦĬÌd ĬĞÑĦĹãf

        # IÌ ŃĦ ŁĹĦ İIJİĦĮĻŁİ
        ĨÌ ĹĦĬÌgİIJİĦĮĻŁİf

            # IÌ ĬĞÑĦĹ ÂIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ İIJİĦĮĻŁİ ĞĹĹĞÑĹd ĢĹĦĞĻĦ ĻÎĦİ
            # ÌĨĬĬĦÂ ŃĨĻÎ ŅĦĹIJĹ
            ĨÌ ĮIJĻ ÎĞĹĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍÎĻçİIJİĦĮĻŁİĹďãf
                ĬĞÑĦĹgŃĦĨÍÎĻçİIJİĦĮĻŁİĹ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgŃĦĨÍÎĻĹã
```

```
            ƒ IÌ ĻÎĦĹĦ ÎĽ ĮIJ İIJİĦĮĻŁİ ĞĹĹĞÑ ÌIJĹ ŃĦÎÍÎĻĽ
            ƒ TÎĦ ĞĹĹĞÑ ÂIJĦĹĮďĻ ĦŃÎĽĻ ÌIJĹ ĠÎĞĽĦĹ ÑĦĻ ĦÎĻÎĦĹg
            ŘĞÑĦĹgĠÎĞĽçİIJİĦĮĻŁİĽ ł ĮĴgŅĦĹIJĽçŘÎÎĦâŘĞÑĦĹgĠÎĞĽĦĽã

        ƒ BŁÎŘÂ ŃĦÎÍÎĻ ŁĴÂĞĻĦĽ ŃÎĻÎ İIJİĦĮĻŁİ h ĻĞÎĦ ĴĹĦŁÎIJŁĽ
        ƒ ŁĴÂĞĻĦĽ İŁŘĮÎĴŘÎĦÂ ĠÑ ĹĦĻĞÎĮ ÌĞĢĻIJĹ ĞĮÂ ŁĴÂĞĻĦ ŃÎĻÎ
        ƒ ĢŁĹĹĦĮĻ ÍĹĞÂÎĦĮĻĽ
        ŃĦÎÍÎĻçŁĴÂĞĻĦĽ ł ċ
            ĽĦŘÌgİIJİĦĮĻŁİ ő ŘĞÑĦĹgŃĦÎÍÎĻçİIJİĦĮĻŁİĽ h ċ
            ĽĦŘÌgĢŁĹĹĦĮĻçŘĦĞĹĮÎĮÍçĹĞĻĦ ő ŘĞÑĦĹgÂŃĦÎÍÎĻĽ
        ŘĞÑĦĹgŃĦÎÍÎĻçİIJİĦĮĻŁİĽ ł ŃĦÎÍÎĻçŁĴÂĞĻĦĽ

        ƒ BŁÎŘÂ ĠÎĞĽ ŁĴÂĞĻĦĽ
        ĠÎĞĽçŁĴÂĞĻĦĽ ł ċ
            ĽĦŘÌgİIJİĦĮĻŁİ ő ŘĞÑĦĹgĠÎĞĽçİIJİĦĮĻŁİĽ h ċ
            ĽĦŘÌgĢŁĹĹĦĮĻçŘĦĞĹĮÎĮÍçĹĞĻĦ ő ŘĞÑĦĹgÂĠÎĞĽĦĽ
        ŘĞÑĦĹgĠÎĞĽçİIJİĦĮĻŁİĽ ł ĠÎĞĽçŁĴÂĞĻĦĽ

    ƒ VĞĮÎŘŘĞ SGD ŁĴÂĞĻĦĽ âĞĽ ĠĦÌIJĹĦ İIJİĦĮĻŁİ ŁĴÂĞĻĦã
    ĦŘĽĦf
        ŃĦÎÍÎĻçŁĴÂĞĻĦĽ ł hĽĦŘÌgĢŁĹĹĦĮĻçŘĦĞĹĮÎĮÍçĹĞĻĦ ő ċ
                            ŘĞÑĦĹgÂŃĦÎÍÎĻĽ
        ĠÎĞĽçŁĴÂĞĻĦĽ ł hĽĦŘÌgĢŁĹĹĦĮĻçŘĦĞĹĮÎĮÍçĹĞĻĦ ő ċ
                           ŘĞÑĦĹgÂĠÎĞĽĦĽ

    ƒ UĴÂĞĻĦ ŃĦÎÍÎĻĽ ĞĮÂ ĠÎĞĽĦĽ ŁĽÎĮÍ ĦÎĻÎĦĹ
    ƒ ŁĞĮŘŘĞ IJĹ İIJİĦĮĻŁİ ŁĴÂĞĻĦĽ
    ŘĞÑĦĹgŃĦÎÍÎĻĽ κł ŃĦÎÍÎĻçŁĴÂĞĻĦĽ
    ŘĞÑĦĹgĠÎĞĽĦĽ κł ĠÎĞĽçŁĴÂĞĻĦĽ

ƒ CĞŘŘ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴÂĞĻĦĽ
ÂĦÌ ĴIJĽĻçŁĴÂĞĻĦçĴĞĹĞİĽâĽĦŘÌã f
    ĽĦŘÌgÎĻĦĹĞĻÎIJĮĽ κł ĝ


ƒ AÂĞÍĹĞÂ IJĴĻÎŘÎÑĦĹ
ĢŘĞĽĽ OĴĻÎŘÎÑĦĹçAÂĞÍĹĞÂf

    ƒ IĮÎĻÎĞŘÎÑĦ IJĴĻÎŘÎÑĦĹ h ĽĦĻ ĽĦĻĻÎĮÍĽ
    ÂĦÌ ççÎĮÎĻççâĽĦŘÌd ŘĦĞĹĮÎĮÍçĹĞĻĦłĝgd ÂĦĢĞÑłflgd ĦĴĽÎŘIJĮłĝĦhìã f
        ĽĦŘÌgŘĦĞĹĮÎĮÍçĹĞĻĦ ł ŘĦĞĹĮÎĮÍçĹĞĻĦ
        ĽĦŘÌgĢŁĹĹĦĮĻçŘĦĞĹĮÎĮÍçĹĞĻĦ ł ŘĦĞĹĮÎĮÍçĹĞĻĦ
        ĽĦŘÌgÂĦĢĞÑ ł ÂĦĢĞÑ
        ĽĦŘÌgÎĻĦĹĞĻÎIJĮĽ ł fl
        ĽĦŘÌgĦĴĽÎŘIJĮ ł ĦĴĽÎŘIJĮ
```

```
    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĹ
    ÂĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgĤĦĢĞÑf
            ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĹĦĬÌgĤĦĢĞÑ ắ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    # UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĹ
    ÂĦÌ ŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ĤIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĴĦ ĞĹĹĞÑĹd
        # ĢĹĦĞĻĦ ĻĴĦİ ÌĨĬĬĦĤ ŃĨĻĴ ŅĦĹIJĹ
        ĨÌ ĮIJĻ ĴĞĹĞĻĻĹâĬĞÑĦĹd dŃĦĨÍĴĻçĢĞĢĴĦdãf
            ĬĞÑĦĹgŃĦĨÍĴĻçĢĞĢĴĦ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgŃĦĨÍĴĻĹã
            ĬĞÑĦĹgĠĨĞĹçĢĞĢĴĦ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgĠĨĞĹĦĹã

        # UĴĤĞĻĦ ĢĞĢĴĦ ŃĨĻĴ ĹĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ÍĹĞĤĨĦĮĻĹ
        ĬĞÑĦĹgŃĦĨÍĴĻçĢĞĢĴĦ κł ĬĞÑĦĹgĤŃĦĨÍĴĻĹắắğ
        ĬĞÑĦĹgĠĨĞĹçĢĞĢĴĦ κł ĬĞÑĦĹgĤĠĨĞĹĦĹắắğ

        # VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ IJĹİĞĬĨŅĞĻĨIJĮ
        # ŃĨĻĴ ĹĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĴĦ
        ĬĞÑĦĹgŃĦĨÍĴĻĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                         ĬĞÑĦĹgĤŃĦĨÍĴĻĹ w ċ
                         âĮĴgĹĶĹĻâĬĞÑĦĹgŃĦĨÍĴĻçĢĞĢĴĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĹĦĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                        ĬĞÑĦĹgĤĠĨĞĹĦĹ w ċ
                        âĮĴgĹĶĹĻâĬĞÑĦĹgĠĨĞĹçĢĞĢĴĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã

    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĹ
    ÂĦÌ ĴIJĹĻçŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌãf
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κł ĝ


# RMSĴĹIJĴ IJĴĻĨİĨŅĦĹ
ĢĬĞĹĹ OĴĻĨİĨŅĦĹçRMSĴĹIJĴf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮĬĹ
    ÂĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłflgflflĝd ÂĦĢĞÑłflgd ĦĴĹĨĬIJĮłĝĦhìd
                ĹÎIJłflgîãf
        ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĹĦĬÌgĤĦĢĞÑ ł ÂĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgĦĴĹĨĬIJĮ ł ĦĴĹĨĬIJĮ
        ĹĦĬÌgĹÎIJ ł ĹÎIJ
```

```
    ſ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĹ
    ÂĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌãf
        ĨÌ ĹĦĬÌgĤĦĢĞÑf
            ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮĬçĹĞĻĦ ő ċ
                âĝg w âĝg κ ĹĦĬÌgĤĦĢĞÑ ő ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

    ſ UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĹ
    ÂĦÌ ŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌd ĬĞÑĦĹãf

        ſ IÌ ĬĞÑĦĹ ĤIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĨĦ ĞĹĹĞÑĹd
        ſ ĢĹĦĞĻĦ ĻĨĦİ ÌĨĬĬĦĤ ŃĨĻĨ ŅĦĹIJĹ
        ĨÌ ĮIJĻ ĨĞĹĞĻĻĹâĬĞÑĦĹd ďŃĦĨĬĨĻçĢĞĢĨĦďãf
            ĬĞÑĦĹgŃĦĨĬĨĻçĢĞĢĨĦ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgŃĦĨĬĨĻĹã
            ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ ł ĮĴgŅĦĹIJĹçĬĨĬĦâĬĞÑĦĹgĠĨĞĹĦĹã

        ſ UĴĤĞĻĦ ĢĞĢĨĦ ŃĨĻĨ ĹĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ĬĹĞĤĨĦĮĻĹ
        ĬĞÑĦĹgŃĦĨĬĨĻçĢĞĢĨĦ ł ĹĦĬÌgĹĨIJ ő ĬĞÑĦĹgŃĦĨĬĨĻçĢĞĢĨĦ κ ċ
            âĝ h ĹĦĬÌgĹĨIJã ő ĬĞÑĦĹgĤŃĦĨĬĨĻĹőőğ
        ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ ł ĹĦĬÌgĹĨIJ ő ĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦ κ ċ
            âĝ h ĹĦĬÌgĹĨIJã ő ĬĞÑĦĹgĤĠĨĞĹĦĹőőğ

        ſ VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹİĞĬĨŅĞĻĨIJĮ
        ſ ŃĨĻĨ ĹĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĨĦ
        ĬĞÑĦĹgŃĦĨĬĨĻĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ő ċ
                         ĬĞÑĦĹgĤŃĦĨĬĨĻĹ w ċ
                         âĮĴgĹĶĹĻâĬĞÑĦĹgŃĦĨĬĨĻçĢĞĢĨĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĹĦĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ő ċ
                        ĬĞÑĦĹgĤĠĨĞĹĦĹ w ċ
                        âĮĴgĹĶĹĻâĬĞÑĦĹgĠĨĞĹçĢĞĢĨĦã κ ĹĦĬÌgĦĴĹĨĬIJĮã

    ſ CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĹ
    ÂĦÌ ĴIJĹĻçŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌãf
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κł ĝ


ſ AĤĞİ IJĴĻĨİĨŅĦĹ
ĢĬĞĹĹ OĴĻĨİĨŅĦĹçAĤĞİf

    ſ IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĹĦĻ ĹĦĻĻĨĮĹĹ
    ÂĦÌ ççĨĮĨĻççâĹĦĬÌd ĬĦĞĹĮĨĮĬçĹĞĻĦłflgflflĝd ÂĦĢĞÑłflgd ĦĴĹĨĬIJĮłĝĦhìd
                ĠĦĻĞçĝłflgîd ĠĦĻĞçğłflgîîîãf
        ĹĦĬÌgĬĦĞĹĮĨĮĬçĹĞĻĦ ł ĬĦĞĹĮĨĮĬçĹĞĻĦ
        ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĬçĹĞĻĦ ł ĬĦĞĹĮĨĮĬçĹĞĻĦ
        ĹĦĬÌgĤĦĢĞÑ ł ÂĦĢĞÑ
        ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ ł fl
        ĹĦĬÌgĦĴĹĨĬIJĮ ł ĦĴĹĨĬIJĮ
        ĹĦĬÌgĠĦĻĞçĝ ł ĠĦĻĞçĝ
        ĹĦĬÌgĠĦĻĞçğ ł ĠĦĻĞçğ
```

```
ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĹ
ÂĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌãƒ
    ĨÌ ĹĦĬÌgĤĦĢĞÑƒ
        ĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĹĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
            âĝg w âĝg κ ĹĦĬÌgĤĦĢĞÑ ắ ĹĦĬÌgĨĻĦĹĞĻĨIJĮĹãã

ƒ UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĹ
ÂĦÌ ŁĴĤĞĻĦçĴĞĹĞİĹâĹĦĬÌd ĬĞÑĦĹãƒ

    ƒ IÌ ĬĞÑĦĹ ĤIJĦĹ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢĴĦ ĞĹĹĞÑĹd
    ƒ ĢĹĦĞĻĦ ĻĴĦİ ÌĨĬĬĦĤ ŃĨĻĴ ŅĦĹIJĹ
    ĨÌ ĮIJĻ ĴĞĹĞĻĻĹâĬĞÑĦĹd dŃĦĨÍĴĻçĢĞĢĴĦdãƒ
        ĬĞÑĦĹgŃĦĨÍĴĻçİIJİĦĮĻŁİĹ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgŃĦĨÍĴĻĹã
        ĬĞÑĦĹgŃĦĨÍĴĻçĢĞĢĴĦ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgŃĦĨÍĴĻĹã
        ĬĞÑĦĹgĠĨĞĹçİIJİĦĮĻŁİĹ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã
        ĬĞÑĦĹgĠĨĞĹçĢĞĢĴĦ ł ĮĴgŅĦĹIJĹçĬĨĨĦâĬĞÑĦĹgĠĨĞĹĦĹã

    ƒ UĴĤĞĻĦ İIJİĦĮĻŁİ  ŃĨĻĴ ĢŁĹĹĦĮĻ ÍĹĞĤĨĦĮĻĹ
    ĬĞÑĦĹgŃĦĨÍĴĻçİIJİĦĮĻŁİĹ ł ĹĦĬÌgĠĦĻĞçĝ ắ ċ
                              ĬĞÑĦĹgŃĦĨÍĴĻçİIJİĦĮĻŁİĹ κ ċ
                              âĝ h ĹĦĬÌgĠĦĻĞçĝã ắ ĬĞÑĦĹgĤŃĦĨÍĴĻĹ
    ĬĞÑĦĹgĠĨĞĹçİIJİĦĮĻŁİĹ ł ĹĦĬÌgĠĦĻĞçĝ ắ ċ
                            ĬĞÑĦĹgĠĨĞĹçİIJİĦĮĻŁİĹ κ ċ
                            âĝ h ĹĦĬÌgĠĦĻĞçĝã ắ ĬĞÑĦĹgĤĠĨĞĹĦĹ
    ƒ GĦĻ ĢIJĹĹĦĢĻĦĤ İIJİĦĮĻŁİ
    ƒ ĹĦĬÌgĨĻĦĹĞĻĨIJĮ ĨĹ ƒl ĞĻ ÌĨĹĹĻ ĴĞĹĹ
    ƒ ĞĮĤ ŃĦ ĮĦĦĤ ĻIJ ĹĻĞĹĻ ŃĨĻĴ ĝ ĴĦĹĦ
    ŃĦĨÍĴĻçİIJİĦĮĻŁİĹçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨÍĴĻçİIJİĦĮĻŁİĹ w ċ
        âĝ h ĹĦĬÌgĠĦĻĞçĝ ắắ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
    ĠĨĞĹçİIJİĦĮĻŁİĹçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĹçİIJİĦĮĻŁİĹ w ċ
        âĝ h ĹĦĬÌgĠĦĻĞçĝ ắắ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
    ƒ UĴĤĞĻĦ ĢĞĢĴĦ ŃĨĻĴ ĹĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ÍĹĞĤĨĦĮĻĹ
    ĬĞÑĦĹgŃĦĨÍĴĻçĢĞĢĴĦ ł ĹĦĬÌgĠĦĻĞçğ ắ ĬĞÑĦĹgŃĦĨÍĴĻçĢĞĢĴĦ κ ċ
        âĝ h ĹĦĬÌgĠĦĻĞçğã ắ ĬĞÑĦĹgĤŃĦĨÍĴĻĹắắğ
    ĬĞÑĦĹgĠĨĞĹçĢĞĢĴĦ ł ĹĦĬÌgĠĦĻĞçğ ắ ĬĞÑĦĹgĠĨĞĹçĢĞĢĴĦ κ ċ
        âĝ h ĹĦĬÌgĠĦĻĞçğã ắ ĬĞÑĦĹgĤĠĨĞĹĦĹắắğ
    ƒ GĦĻ ĢIJĹĹĦĢĻĦĤ ĢĞĢĴĦ
    ŃĦĨÍĴĻçĢĞĢĴĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgŃĦĨÍĴĻçĢĞĢĴĦ w ċ
        âĝ h ĹĦĬÌgĠĦĻĞçğ ắắ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã
    ĠĨĞĹçĢĞĢĴĦçĢIJĹĹĦĢĻĦĤ ł ĬĞÑĦĹgĠĨĞĹçĢĞĢĴĦ w ċ
        âĝ h ĹĦĬÌgĠĦĻĞçğ ắắ âĹĦĬÌgĨĻĦĹĞĻĨIJĮĹ κ ĝãã

    ƒ VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹİĞĬĨŅĞĻĨIJĮ
    ƒ ŃĨĻĴ ĹĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢĴĦ
    ĬĞÑĦĹgŃĦĨÍĴĻĹ κł hĹĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                      ŃĦĨÍĴĻçİIJİĦĮĻŁİĹçĢIJĹĹĦĢĻĦĤ w ċ
                      âĮĴgĹĶĹĻâŃĦĨÍĴĻçĢĞĢĴĦçĢIJĹĹĦĢĻĦĤã κ
                          ĹĦĬÌgĦĴĹĨĬIJĮã
```

```
        ĬĞÑĦĹgĠĬĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĬĮÍçĹĞĻĦ ắ ċ
                        ĠĬĞĽçĬIJİĦĮĻŁİĽçĢIJĹĹĦĢĻĦĤ w ċ
                        âĮĴgĽĶĹĻâĠĬĞĽçĢĞĢĬĦçĢIJĹĹĦĢĻĦĤã κ
                            ĽĦĬÌgĦĴĽĬĬIJĮã

    fi CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ÂĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĽĦĬÌgĬĻĦĹĞĻĬIJĮĽ κł ĝ


fi CIJİĬIJĮ ĬIJĽĽ ĢĬĞĽĽ
ĢĬĞĽĽ LIJĽĽf

    fi RĦÍŁĬĞĹĬĢĞĻĬIJĮ ĬIJĽĽ ĢĞĬĢŁĬĞĻĬIJĮ
    ÂĦÌ ĹĦÍŁĬĞĹĬĢĞĻĬIJĮçĬIJĽĽâĽĦĬÌãf

        fi fl ĠÑ ÂĦÌĞŁĬĻ
        ĹĦÍŁĬĞĹĬĢĞĻĬIJĮçĬIJĽĽ ł fl

        fi CĞĬĢŁĬĞĻĦ ĹĦÍŁĬĞĹĬĢĞĻĬIJĮ ĬIJĽĽ
        fi ĬĻĦĹĞĻĦ ĞĬĬ ĻĹĞĬĮĞĠĬĦ ĬĞÑĦĹĽ
        ÌIJĹ ĬĞÑĦĹ ĬĮ ĽĦĬÌgĻĹĞĬĮĞĠĬĦçĬĞÑĦĹĽf

            fi Lĝ ĹĦÍŁĬĞĹĬĢĞĻĬIJĮ h ŃĦĬÍÎĻĽ
            fi ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ fl
            ĬÌ ĬĞÑĦĹgŃĦĬÍÎĻçĹĦÍŁĬĞĹĬĢĦĹçĬĝ ń flf
                ĹĦÍŁĬĞĹĬĢĞĻĬIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĬÍÎĻçĹĦÍŁĬĞĹĬĢĦĹçĬĝ ắ ċ
                                        ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgŃĦĬÍÎĻĽãã

            fi Lğ ĹĦÍŁĬĞĹĬĢĞĻĬIJĮ h ŃĦĬÍÎĻĽ
            ĬÌ ĬĞÑĦĹgŃĦĬÍÎĻçĹĦÍŁĬĞĹĬĢĦĹçĬğ ń flf
                ĹĦÍŁĬĞĹĬĢĞĻĬIJĮçĬIJĽĽ κł ĬĞÑĦĹgŃĦĬÍÎĻçĹĦÍŁĬĞĹĬĢĦĹçĬğ ắ ċ
                                        ĮĴgĽŁİâĬĞÑĦĹgŃĦĬÍÎĻĽ ắ ċ
                                                ĬĞÑĦĹgŃĦĬÍÎĻĽã

            fi Lĝ ĹĦÍŁĬĞĹĬĢĞĻĬIJĮ h ĠĬĞĽĦĽ
            fi ĢĞĬĢŁĬĞĻĦ IJĮĬÑ ŃÎĦĮ ÌĞĢĻIJĹ ÍĹĦĞĻĦĹ ĻÎĞĮ fl
            ĬÌ ĬĞÑĦĹgĠĬĞĽçĹĦÍŁĬĞĹĬĢĦĹçĬĝ ń flf
                ĹĦÍŁĬĞĹĬĢĞĻĬIJĮçĬIJĽĽ κł ĬĞÑĦĹgĠĬĞĽçĹĦÍŁĬĞĹĬĢĦĹçĬĝ ắ ċ
                                        ĮĴgĽŁİâĮĴgĞĠĽâĬĞÑĦĹgĠĬĞĽĦĽãã

            fi Lğ ĹĦÍŁĬĞĹĬĢĞĻĬIJĮ h ĠĬĞĽĦĽ
            ĬÌ ĬĞÑĦĹgĠĬĞĽçĹĦÍŁĬĞĹĬĢĦĹçĬğ ń flf
                ĹĦÍŁĬĞĹĬĢĞĻĬIJĮçĬIJĽĽ κł ĬĞÑĦĹgĠĬĞĽçĹĦÍŁĬĞĹĬĢĦĹçĬğ ắ ċ
                                        ĮĴgĽŁİâĬĞÑĦĹgĠĬĞĽĦĽ ắ ċ
                                                ĬĞÑĦĹgĠĬĞĽĦĽã

        ĹĦĻŁĹĮ ĹĦÍŁĬĞĹĬĢĞĻĬIJĮçĬIJĽĽ
```

```
ƒ SĦĻwĹĦİĦİĠĦĹ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
ÎĦÌ ĹĦİĦİĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽâĽĦĬÌd ĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽãf
    ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ ł ĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ

ƒ CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÎĞĻĞ ĞĮÎ ĹĦĹŁĬĞĹĨĢĞĻĨIJĮ ĬIJĽĽĦĽ
ƒ ÍĨŁĦĮ ĬIJÎĦĦĬ IJŁĻĴŁĻ ĞĮÎ ÍĹIJŁĮÎ ĻĹŁĻÎ ŁĞĬŁĦĽ
ÎĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñd ős ĨĮĢĬŁÎĦçĹĦĹŁĬĞĹĨĢĞĻĨIJĮłFĞĬĽĦãf

    ƒ CĞĬĢŁĬĞĻĦ ĽĞĬĴĬĦ ĬIJĽĽĦĽ
    ĽĞĬĴĬĦçĬIJĽĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÎâIJŁĻĴŁĻd Ñã

    ƒ CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĽĽ
    ÎĞĻĞçĬIJĽĽ ł ĮĴgĬĦĞĮâĽĞĬĴĬĦçĬIJĽĽĦĽã

    ƒ AÎÎ ĞĢĢŁĬŁĬĞĻĦÎ ĽŁĬ IJÌ ĬIJĽĽĦĽ ĞĮÎ ĽĞĬĴĬĦ ĢIJŁĮĻ
    ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÎçĽŁĬ κł ĮĴgĽŁĬâĽĞĬĴĬĦçĬIJĽĽĦĽã
    ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÎçĢIJŁĮĻ κł ĬĦĮâĽĞĬĴĬĦçĬIJĽĽĦĽã

    ƒ IÌ ĬŁĽĻ ÎĞĻĞ ĬIJĽĽ h ĹĦĻŁĹĮ ĨĻ
    ĨÌ ĮIJĻ ĨĮĢĬŁÎĦçĹĦĹŁĬĞĹĨĢĞĻĨIJĮf
        ĹĦĻŁĹĮ ÎĞĻĞçĬIJĽĽ

    ƒ RĦĻŁĹĮ ĻÎĦ ÎĞĻĞ ĞĮÎ ĹĦĹŁĬĞĹĨĢĞĻĨIJĮ ĬIJĽĽĦĽ
    ĹĦĻŁĹĮ ÎĞĻĞçĬIJĽĽd ĽĦĬÌgĹĦĹŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽâã

ƒ CĞĬĢŁĬĞĻĦĽ ĞĢĢŁĬŁĬĞĻĦÎ ĬIJĽĽ
ÎĦÌ ĢĞĬĢŁĬĞĻĦçĞĢĢŁĬŁĬĞĻĦÎâĽĦĬÌd ős ĨĮĢĬŁÎĦçĹĦĹŁĬĞĹĨĢĞĻĨIJĮłFĞĬĽĦãf

    ƒ CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĽĽ
    ÎĞĻĞçĬIJĽĽ ł ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÎçĽŁĬ w ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÎçĢIJŁĮĻ

    ƒ IÌ ĬŁĽĻ ÎĞĻĞ ĬIJĽĽ h ĹĦĻŁĹĮ ĨĻ
    ĨÌ ĮIJĻ ĨĮĢĬŁÎĦçĹĦĹŁĬĞĹĨĢĞĻĨIJĮf
        ĹĦĻŁĹĮ ÎĞĻĞçĬIJĽĽ

    ƒ RĦĻŁĹĮ ĻÎĦ ÎĞĻĞ ĞĮÎ ĹĦĹŁĬĞĹĨĢĞĻĨIJĮ ĬIJĽĽĦĽ
    ĹĦĻŁĹĮ ÎĞĻĞçĬIJĽĽd ĽĦĬÌgĹĦĹŁĬĞĹĨĢĞĻĨIJĮçĬIJĽĽâã

ƒ RĦĽĦĻ ŁĞĹĨĞĠĬĦĽ ÌIJĹ ĞĢĢŁĬŁĬĞĻĦÎ ĬIJĽĽ
ÎĦÌ ĮĦŃçĴĞĽĽâĽĦĬÌã f
    ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÎçĽŁĬ ł fl
    ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÎçĢIJŁĮĻ ł fl
```

```
ƒ CĹIJĽĽhĦĮĻĹIJĴÑ ĬIJĽĽ
ĢĬĞĽĽ LIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâLIJĽĽãf

    ƒ FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        ƒ NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ ĨĮ Ğ ĠĞĻĢÎ
        ĽĞİĴĬĦĽ ł ĬĦĮâÑçĴĹĦĤã

        ƒ CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ĤĨŁĨĽĨIJĮ ĠÑ ƒl
        ƒ CĬĨĴ ĠIJĻÎ ĽĨĤĦĽ ĻIJ ĮIJĻ ĤĹĞÍ İĦĞĮ ĻIJŃĞĹĤĽ ĞĮÑ ŁĞĬŁĦ
        ÑçĴĹĦĤçĢĬĨĴĴĦĤ ł ĮĴgĢĬĨĴâÑçĴĹĦĤd ĝĦhìd ĝ h ĝĦhìã

        ƒ PĹIJĠĞĠĨĬĨĻĨĦĽ ÌIJĹ ĻĞĹÍĦĻ ŁĞĬŁĦĽ h
        ƒ IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĽ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽĤĞĴĦã łł ĝf
            ĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĽ ł ÑçĴĹĦĤçĢĬĨĴĴĦĤä
                ĹĞĮÍĦâĽĞİĴĬĦĽãd
                ÑçĻĹŁĦ
            ã

        ƒ MĞĽĨ ŁĞĬŁĦĽ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJĤĦĤ ĬĞĠĦĬĽ
        ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĽĤĞĴĦã łł ğf
            ĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĽ ł ĮĴgĽŁİâ
                ÑçĴĹĦĤçĢĬĨĴĴĦĤ ẩ ÑçĻĹŁĦd
                ĞÑĨĽłĝ
            ã

        ƒ LIJĽĽĦĽ
        ĮĦÍĞĻĨŁĦçĬIJÍçĬĨĨĦĬĨÎIJIJĤĽ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨĤĦĮĢĦĽã
        ĹĦĻŁĹĮ ĮĦÍĞĻĨŁĦçĬIJÍçĬĨĨĦĬĨÎIJIJĤĽ

    ƒ BĞĢĨŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤŁĞĬŁĦĽd ÑçĻĹŁĦãf

        ƒ NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâĤŁĞĬŁĦĽã
        ƒ NŁİĠĦĹ IJÌ ĬĞĠĦĬĽ ĨĮ ĦŁĦĹÑ ĽĞİĴĬĦ
        ƒ WĦďĬĬ ŁĽĦ ĻĤĦ ÌĨĹĽĻ ĽĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦİ
        ĬĞĠĦĬĽ ł ĬĦĮâĤŁĞĬŁĦĽäflãã

        ƒ IÌ ĬĞĠĦĬĽ ĞĹĦ ĽĴĞĹĽĦd ĻŁĹĮ ĻĤĦİ ĨĮĻIJ IJĮĦhÎIJĻ ŁĦĢĻIJĹ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĽĤĞĴĦã łł ĝf
            ÑçĻĹŁĦ ł ĮĴgĦÑĦâĬĞĠĦĬĽääÑçĻĹŁĦã

        ƒ CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł hÑçĻĹŁĦ w ĤŁĞĬŁĦĽ
        ƒ NIJĹĬĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽĞİĴĬĦĽ
```

```
ſı SIJÌĻİĞŇ ĢĬĞĹĹĨÌÌĨĦĹ h ĢIJİĠĨĮĦĤ SIJÌĻİĞŇ ĞĢĻĨĿĞĻĨIJĮ
ſı ĞĮĤ ĢĹIJĹĹhĦĮĻĹIJĴÑ ĬIJĹĹ ÌIJĹ ÌĞĹĻĦĹ ĠĞĢĪŃĞĹĤ ĹĻĦĴ
ĢĬĞĹĹ AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇçLIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑâãf

    ſı BĞĢĪŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ĤĿĞĬŁĦĹd ÑçĻĹŁĦãf

        ſı NŁİĠĦĹ IJÌ ĹĞİĴĬĦĹ
        ĹĞİĴĬĦĹ ł ĬĦĮâĤĿĞĬŁĦĹã

        ſı IÌ ĬĞĠĦĬĹ ĞĹĦ IJĮĦhÎIJĻ ĦĮĢIJĤĦĤd
        ſı ĻŁĹĮ ĻÎĦİ ĨĮĻIJ ĤĨĹĢĹĦĻĦ ĿĞĬŁĦĹ
        ĨÌ ĬĦĮâÑçĻĹŁĦgĹÎĞĴĦã łł ğf
            ÑçĻĹŁĦ ł ĮĴgĞĹÍİĞÑâÑçĻĹŁĦd ĞÑĨĹłĝã

        ſı CIJĴÑ ĹIJ ŃĦ ĢĞĮ ĹĞÌĦĬÑ İIJĤĨÌÑ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĤĿĞĬŁĦĹgĢIJĴÑâã
        ſı CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹäĹĞĮÍĦâĹĞİĴĬĦĹãd ÑçĻĹŁĦã hł ĝ
        ſı NIJĹİĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĹĦĬÌgĤĨĮĴŁĻĹ w ĹĞİĴĬĦĹ


ſı BĨĮĞĹÑ ĢĹIJĹĹhĦĮĻĹIJĴÑ ĬIJĹĹ
ĢĬĞĹĹ LIJĹĹçBĨĮĞĹÑCĹIJĹĹĦĮĻĹIJĴÑâLIJĹĹãf

    ſı FIJĹŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        ſı CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦĿĦĮĻ ĤĨĿĨĹĨIJĮ ĠÑ ſl
        ſı CĬĨĴ ĠIJĻÎ ĹĨĤĦĹ ĻIJ ĮIJĻ ĤĹĞÍ İĦĞĮ ĻIJŃĞĹĤĹ ĞĮÑ ĿĞĬŁĦ
        ÑçĴĹĦĤçĢĬĨĴĴĦĤ ł ĮĴgĢĬĨĴâÑçĴĹĦĤd ĝĦhìd ĝ h ĝĦhìã

        ſı CĞĬĢŁĬĞĻĦ ĹĞİĴĬĦhŃĨĹĦ ĬIJĹĹ
        ĹĞİĴĬĦçĬIJĹĹĦĹ ł hâÑçĻĹŁĦ ắ ĮĴgĬIJÍâÑçĴĹĦĤçĢĬĨĴĴĦĤã κ
                          âĝ h ÑçĻĹŁĦã ắ ĮĴgĬIJÍâĝ h ÑçĴĹĦĤçĢĬĨĴĴĦĤãã
        ĹĞİĴĬĦçĬIJĹĹĦĹ ł ĮĴgİĦĞĮâĹĞİĴĬĦçĬIJĹĹĦĹd ĞÑĨĹłhĝã

        ſı RĦĻŁĹĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ĹĞİĴĬĦçĬIJĹĹĦĹ

    ſı BĞĢĪŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ĤĿĞĬŁĦĹd ÑçĻĹŁĦãf

        ſı NŁİĠĦĹ IJÌ ĹĞİĴĬĦĹ
        ĹĞİĴĬĦĹ ł ĬĦĮâĤĿĞĬŁĦĹã
        ſı NŁİĠĦĹ IJÌ IJŁĻĴŁĻĹ ĨĮ ĦĿĦĹÑ ĹĞİĴĬĦ
        ſı WĦdĬĬ ŁĹĦ ĻÎĦ ÌĨĹĹĻ ĹĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻÎĦİ
        IJŁĻĴŁĻĹ ł ĬĦĮâĤĿĞĬŁĦĹäflãã
```

```
        # CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦĿĦĮĻ ĤĨĿĨĹĨIJĮ ĠÑ #
        # CĬĨĴ ĠIJĻÎ ĹĨĤĦĹ ĻIJ ĮIJĻ ĤĹĞÍ ĬĦĞĮ ĻIJŃĞĹĤĹ ĞĮÑ ĿĞĬŁĦ
        ĢĬĨĴĴĦĤçĤĿĞĬŁĦĹ ł ĮĴgĢĬĨĴâĤĿĞĬŁĦĹd ĝĦhìd ĝ h ĝĦhìã

        # CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł hâÑçĻĹŁĦ w ĢĬĨĴĴĦĤçĤĿĞĬŁĦĹ h
                        âĝ h ÑçĻĹŁĦã w âĝ h ĢĬĨĴĴĦĤçĤĿĞĬŁĦĹãã w IJŁĻĴŁĻĹ
        # NIJĹĬĞĬĨŅĦ ÍĹĞĤĨĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĹĦĬÌgĤĨĮĴŁĻĹ w ĹĞĬĴĬĦĹ


# MĦĞĮ SĶŁĞĹĦĤ EĹĹIJĹ ĬIJĹĹ
ĢĬĞĹĹ LIJĹĹçMĦĞĮSĶŁĞĹĦĤEĹĹIJĹâLIJĹĹãf  # Lğ ĬIJĹĹ

    # FIJĹŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        # CĞĬĢŁĬĞĻĦ ĬIJĹĹ
        ĹĞĬĴĬĦçĬIJĹĹĦĹ ł ĮĴgĬĦĞĮââÑçĻĹŁĦ h ÑçĴĹĦĤãååğd ĞŃĨĹłhĝã

        # RĦĻŁĹĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ĹĞĬĴĬĦçĬIJĹĹĦĹ

    # BĞĢĨŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĹĦĬÌd ĤĿĞĬŁĦĹd ÑçĻĹŁĦãf

        # NŁĬĠĦĹ IJÌ ĹĞĬĴĬĦĹ
        ĹĞĬĴĬĦĹ ł ĬĦĮâĤĿĞĬŁĦĹã
        # NŁĬĠĦĹ IJÌ IJŁĻĴŁĻĹ ĨĮ ĦĿĦĹÑ ĹĞĬĴĬĦ
        # WĦďĬĬ ŁĹĦ ĻĨĦ ÌĨĹĹĻ ĹĞĬĴĬĦ ĻIJ ĢIJŁĮĻ ĻĨĦĬ
        IJŁĻĴŁĻĹ ł ĬĦĮâĤĿĞĬŁĦĹäflãã

        # GĹĞĤĨĨĦĮĻ IJĮ ĿĞĬŁĦĹ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł hğ ǎ âÑçĻĹŁĦ h ĤĿĞĬŁĦĹã w IJŁĻĴŁĻĹ
        # NIJĹĬĞĬĨŅĦ ÍĹĞĤĨĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĹĦĬÌgĤĨĮĴŁĻĹ w ĹĞĬĴĬĦĹ


# MĦĞĮ AĠĹIJĬŁĻĦ EĹĹIJĹ ĬIJĹĹ
ĢĬĞĹĹ LIJĹĹçMĦĞĮAĠĹIJĬŁĻĦEĹĹIJĹâLIJĹĹãf  # Lĝ ĬIJĹĹ

    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        # CĞĬĢŁĬĞĻĦ ĬIJĹĹ
        ĹĞĬĴĬĦçĬIJĹĹĦĹ ł ĮĴgĬĦĞĮâĮĴgĞĠĹâÑçĻĹŁĦ h ÑçĴĹĦĤãd ĞŃĨĹłhĝã

        # RĦĻŁĹĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ĹĞĬĴĬĦçĬIJĹĹĦĹ
```

```
    # BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÎĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ÎĦĽĞĬŁĦĽd ÑçĻĹŁĦãf

        # NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâĤĦĽĞĬŁĦĽã
        # NŁİĠĦĹ IJÌ IJŁĻĴŁĻĽ ĨĮ ĦĽĦĹÑ ĽĞİĴĬĦ
        # WĦďĬĬ ŁĽĦ ĻĤĦ ÌĨĹĽĻ ĽĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦİ
        IJŁĻĴŁĻĽ ł ĬĦĮâĤĦĽĞĬŁĦĽäflãã

        # CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĮĴgĽĨÍĮâÑçĻĹŁĦ h ĤĦĽĞĬŁĦĽã w IJŁĻĴŁĻĽ
        # NIJĹİĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĽĦĬÌgĤĨĮĴŁĻĽ w ĽĞİĴĬĦĽ


# CIJİİIJĮ ĞĢĢŁĹĞĢÑ ĢĬĞĽĽ
ĢĬĞĽĽ AĢĢŁĹĞĢÑf

    # CĞĬĢŁĬĞĻĦĽ ĞĮ ĞĢĢŁĹĞĢÑ
    # ÍĨĦĮ ĴĹĦĤĨĢĻĨIJĮĽ ĞĮĤ ÍĹIJŁĮĤ ĻĹŁĻÎ ĦĞĬŁĦĽ
    ÎĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd ĴĹĦĤĨĢĻĨIJĮĽd Ñãf

        # GĦĻ GIJİĴĞĹĨĽIJĮ ĹĦĽŁĬĻĽ
        ĢIJİĴĞĹĨĽIJĮĽ ł ĽĦĬÌgĢIJİĴĞĹĦâĴĹĦĤĨĢĻĨIJĮĽd Ñã

        # CĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĞĢĢŁĹĞĢÑ ł ĮĴgİĦĞĮâĢIJİĴĞĹĨĽIJĮĽã

        # AĤĤ ĞĢĢŁİŁĬĞĻĦĤ ĽŁİ IJÌ İĞĻĢĨĨĮÍ ĦĞĬŁĦĽ ĞĮĤ ĽĞİĴĬĦ ĢIJŁĮĻ
        ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĽŁİ κł ĮĴgĽŁİâĢIJİĴĞĹĨĽIJĮĽã
        ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ κł ĬĦĮâĢIJİĴĞĹĨĽIJĮĽã

        # RĦĻŁĹĮ ĞĢĢŁĹĞĢÑ
        ĹĦĻŁĹĮ ĞĢĢŁĹĞĢÑ

    # CĞĬĢŁĬĞĻĦĽ ĞĢĢŁİŁĬĞĻĦĤ ĞĢĢŁĹĞĢÑ
    ÎĦÌ ĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦĤâĽĦĬÌãf

        # CĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĽŁİ w ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ

        # RĦĻŁĹĮ ĻĤĦ ĤĞĻĞ ĞĮĤ ĹĦÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽĦĽ
        ĹĦĻŁĹĮ ĞĢĢŁĹĞĢÑ

    # RĦĽĦĻ ĽĞĹĨĞĠĬĦĽ ÌIJĹ ĞĢĢŁİŁĬĞĻĦĤ ĞĢĢŁĹĞĢÑ
    ÎĦÌ ĮĦŃçĴĞĽĽâĽĦĬÌãf
        ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĽŁİ ł fl
        ĽĦĬÌgĞĢĢŁİŁĬĞĻĦĤçĢIJŁĮĻ ł fl
```

```
ſı AĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ ÌIJĹ ĢĬĞĽĽĨÌÌĨĢĞĻĨIJĮ İIJĤĦĬ
ĢĬĞĽĽ AĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâAĢĢŁĹĞĢÑãf

    ſı NIJ ĨĮĨĻĨĞĬĨĨĢĞĻĨIJĮ ĨĽ ĮĦĦĤĦĤ
    ĤĦÌ ĨĮĨĻâĽĦĬÌd Ñãf
        ĴĞĽĽ

    ſı CIJİĴĞĹĦĽ ĴĹĦĤĨĢĻĨIJĮĽ ĻIJ ĻĨĦ ÍĹIJŁĮĤ ĻĹŁĻĨ ŁĞĬŁĦĽ
    ĤĦÌ ĢIJİĴĞĹĦâĽĦĬÌd ĴĹĦĤĨĢĻĨIJĮĽd Ñãf
        ĨÌ ĬĦĮâÑgĽĨĞĴĦã łł ğf
            Ñ ł ĮĴgĞĹÍİĞŇâÑd ĞŇĨĽłĝã
        ĹĦĻŁĹĮ ĴĹĦĤĨĢĻĨIJĮĽ łł Ñ


ſı AĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ ÌIJĹ ĹĦÍĹĦĽĽĨIJĮ İIJĤĦĬ
ĢĬĞĽĽ AĢĢŁĹĞĢÑçRĦÍĹĦĽĽĨIJĮâAĢĢŁĹĞĢÑãf

    ĤĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ſı CĹĦĞĻĦ ĴĹĦĢĨĽĨIJĮ ĴĹIJĴĦĹĻÑ
        ĽĦĬÌgĴĹĦĢĨĽĨIJĮ ł NIJĮĦ

    ſı CĞĬĢŁĬĞĻĦĽ ĴĹĦĢĨĽĨIJĮ ŁĞĬŁĦ
    ſı ĠĞĽĦĤ IJĮ ĴĞĽĽĦĤ ĨĮ ÍĹIJŁĮĤ ĻĹŁĻĨ ŁĞĬŁĦĽ
    ĤĦÌ ĨĮĨĻâĽĦĬÌd Ñd ĹĦĨĮĨĻłFĞĬĽĦãf
        ĨÌ ĽĦĬÌgĴĹĦĢĨĽĨIJĮ ĨĽ NIJĮĦ IJĹ ĹĦĨĮĨĻf
            ĽĦĬÌgĴĹĦĢĨĽĨIJĮ ł ĮĴgĽĻĤâÑã w ğĥſl

    ſı CIJİĴĞĹĦĽ ĴĹĦĤĨĢĻĨIJĮĽ ĻIJ ĻĨĦ ÍĹIJŁĮĤ ĻĹŁĻĨ ŁĞĬŁĦĽ
    ĤĦÌ ĢIJİĴĞĹĦâĽĦĬÌd ĴĹĦĤĨĢĻĨIJĮĽd Ñãf
        ĹĦĻŁĹĮ ĮĴgĞĠĽIJĬŁĻĦâĴĹĦĤĨĢĻĨIJĮĽ h Ñã l· ĽĦĬÌgĴĹĦĢĨĽĨIJĮ


ſı MIJĤĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJĤĦĬf

    ĤĦÌ ççĨĮĨĻççâĽĦĬÌãf
        ſı CĹĦĞĻĦ Ğ ĬĨĽĻ IJÌ ĮĦĻŃIJĹĨ IJĠÏĦĢĻĽ
        ĽĦĬÌgĬĞÑĦĹĽ ł äã
        ſı SIJÌĻİĞŇ ĢĬĞĽĽĨÌĨĦĹďĽ IJŁĻĴŁĻ IJĠÏĦĢĻ
        ĽĦĬÌgĽIJÌĻİĞÑçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻĴŁĻ ł NIJĮĦ

    ſı AĤĤ IJĠÏĦĢĻĽ ĻIJ ĻĨĦ İIJĤĦĬ
    ĤĦÌ ĞĤĤâĽĦĬÌd ĬĞÑĦĹãf
        ĽĦĬÌgĬĞÑĦĹĽgĞĴĴĦĮĤâĬĞÑĦĹã
```

```
ƒ SĦĻ ĬIJĽĽd IJĴĻĨĬĬĨŅĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ
ÂĦÌ ĽĦĻâĽĦĬÌd ǻd ĬIJĽĽd IJĴĻĨĬĬĨŅĦĹd ĞĢĢŁĹĞĢÑãƒ
    ĽĦĬÌgĬIJĽĽ ł ĬIJĽĽ
    ĽĦĬÌgIJĴĻĨĬĬĨŅĦĹ ł IJĴĻĨĬĬĨŅĦĹ
    ĽĦĬÌgĞĢĢŁĹĞĢÑ ł ĞĢĢŁĹĞĢÑ

ƒ FĨĮĞĬĨŅĦ ĻĤĦ ĬIJÂĦĬ
ÂĦÌ ÌĨĮĞĬĨŅĦâĽĦĬÌãƒ

    ƒ CĹĦĞĻĦ ĞĮĤ ĽĦĻ ĻĤĦ ĨĮĴŁĻ ĬĞÑĦĹ
    ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ ł LĞÑĦĹçIĮĴŁĻâã

    ƒ CIJŁĮĻ ĞĬĬ ĻĤĦ IJĠĬĦĢĻĽ
    ĬĞÑĦĹçĢIJŁĮĻ ł ĬĦĮâĽĦĬÌgĬĞÑĦĹĽã

    ƒ IĮĨĻĬĞĬĨŅĦ Ğ ĬĨĽĻ ĢIJĮĻĞĨĮĨĮĬ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽƒ
    ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ ł äã

    ƒ IĻĦĹĞĻĦ ĻĤĦ IJĠĬĦĢĻĽ
    ÌIJĹ Ĩ ĨĮ ĹĞĮĹĦâĬĞÑĦĹçĢIJŁĮĻãƒ

        ƒ IÌ ĨĻďĽ ĻĤĦ ÌĨĹĽĻ ĬĞÑĦĹd
        ƒ ĻĤĦ ĴĹĦĹĨIJŁĽ ĬĞÑĦĹ IJĠĬĦĢĻ ĨĽ ĻĤĦ ĨĮĴŁĻ ĬĞÑĦĹ
        ĨÌ Ĩ łł ƒƒ
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĴĹĦĹ ł ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹ
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĮĦŇĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝã

        ƒ AĬĬ ĬĞÑĦĹĽ ĦŇĢĦĴĻ ÌIJĹ ĻĤĦ ÌĨĹĽĻ ĞĮĤ ĻĤĦ ĬĞĽĻ
        ĦĬĨÌ Ĩ l· ĬĞÑĦĹçĢIJŁĮĻ h ĝƒ
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĴĹĦĹ ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝã
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĮĦŇĻ ł ĽĦĬÌgĬĞÑĦĹĽäĨкĝã

        ƒ TĤĦ ĬĞĽĻ ĬĞÑĦĹ h ĻĤĦ ĮĦŇĻ IJĠĬĦĢĻ ĨĽ ĻĤĦ ĬIJĽĽ
        ƒ AĬĽIJ ĬĦĻďĽ ĽĞĽĦ ĞĽĨĤĦ ĻĤĦ ĹĦÌĦĹĦĮĢĦ ĻIJ ĻĤĦ ĬĞĽĻ IJĠĬĦĢĻ
        ƒ ŃĤIJĽĦ IJŁĻĴŁĻ ĨĽ ĻĤĦ ÌIJĤĦĬďĽ IJŁĻĴŁĻ
        ĦĬĽĦƒ
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĴĹĦĹ ł ĽĦĬÌgĬĞÑĦĹĽäĨhĝã
            ĽĦĬÌgĬĞÑĦĹĽäĨãgĮĦŇĻ ł ĽĦĬÌgĬIJĽĽ
            ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĽĞĻĨIJĮ ł ĽĦĬÌgĬĞÑĦĹĽäĨã

        ƒ IÌ ĬĞÑĦĹ ĢIJĮĻĞĨĮĽ ĞĮ ĞĻĻĹĨĠŁĻĦ ĢĞĬĬĦĤ ðŃĦĨĬĬĴĻĽðd
        ƒ ĨĻďĽ Ğ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹ h
        ƒ ĞĤĤ ĨĻ ĻIJ ĻĤĦ ĬĨĽĻ IJÌ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
        ƒ WĦ ĤIJĮďĻ ĮĦĦĤ ĻIJ ĢĤĦĢĨ ÌIJĹ ĠĨĞĽĦĽ h
        ƒ ĢĤĦĢĨĨĮĬ ÌIJĹ ŃĦĨĬĬĴĻĽ ĨĽ ĦĮIJŁĬĬ
        ĨÌ ĬĞĽĞĻĻĹâĽĦĬÌgĬĞÑĦĹĽäĨãd dŃĦĨĬĬĴĻĽďãƒ
            ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽgĞĴĴĦĮĤâĽĦĬÌgĬĞÑĦĹĽäĨãã
```

```
        ƒ UĴĤĞĻĦ ĬIJĹĹ IJĠÏĦĢĻ ŃĨĻÎ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĹ
        ĽĦĬÌgĬIJĹĹgĹĦİĦİĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĹâ
            ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĹ
        ã

    ƒ IÌ IJŁĻĴŁĻ ĞĢĻĨĿĞĻĨIJĮ ĨĹ SIJÌĻİĞÑ ĞĮĤ
    ƒ ĬIJĹĹ ÌŁĮĢĻĨIJĮ ĨĹ CĞĻĦÍIJĹĨĢĞĬ CĹIJĹĹhEĮĻĹIJĴÑ
    ƒ ĢĹĦĞĻĦ ĞĮ IJĠÏĦĢĻ IJÌ ĢIJİĠĨĮĦĤ ĞĢĻĨĿĞĻĨIJĮ
    ƒ ĞĮĤ ĬIJĹĹ ÌŁĮĢĻĨIJĮ GIJĮĻĞĨĮĨĮÍ
    ƒ ÌĞĹĻĦĹ ÍĹĞĤĨĦĮĻ ĢĞĬĢŁĬĞĻĨIJĮ
    ĨÌ ĨĹĨĮĹĻĞĮĢĦâĽĦĬÌgĬĞÑĦĹĹähĝād AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞÑã ĞĮĤ ċ
       ĨĹĨĮĹĻĞĮĢĦâĽĦĬÌgĬIJĹĹd LIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑãƒ
        ƒ CĹĦĞĻĦ ĞĮ IJĠÏĦĢĻ IJÌ ĢIJİĠĨĮĦĤ ĞĢĻĨĿĞĻĨIJĮ
        ƒ ĞĮĤ ĬIJĹĹ ÌŁĮĢĻĨIJĮĹ
        ĽĦĬÌgĹIJÌĻİĞÑçĢĬĞĹĹĨÌĨĦĹçIJŁĻĴŁĻ ł ċ
            AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞÑçLIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑâã

ƒ TĹĞĨĮ ĻĨĦ İIJĤĦĬ
ĤĦÌ ĻĹĞĨĮâĹĦĬÌd Xd Ñd ấd ĦĴIJĢĨĹłĝd ĠĞĻĢĨçĹĨÑĦłNIJĮĦd
          ĴĹĨĮçĦĿĦĹÑłĝd ĿĞĬĨĤĞĻĨIJĮçĤĞĻĞłNIJĮĦãƒ

    ƒ IĮĨĻĨĞĬĨÑĦ ĞĢĢŁĹĞĢÑ IJĠÏĦĢĻ
    ĽĦĬÌgĞĢĢŁĹĞĢÑgĨĮĨĻâÑã

    ƒ DĦÌĞŁĬĻ ĿĞĬŁĦ ĨÌ ĠĞĻĢĨ ĹĨÑĦ ĨĹ IJIJĻ ĠĦĨĮÍ ĹĦĻ
    ĻĹĞĨĮçĹĻĦĴĹ ł ĝ

    ƒ IÌ ĻĨĦĹĦ ĨĹ ĿĞĬĨĤĞĻĨIJĮ ĤĞĻĞ ĴĞĹĹĦĤd
    ƒ ĹĦĻ ĤĦÌĞŁĬĻ IJŁİĠĦĹ IJÌ ĹĻĦĴĹ ÌIJĹ ĿĞĬĨĤĞĻĨIJĮ ĞĹ ŃĦĬĬ
    ĨÌ ĿĞĬĨĤĞĻĨIJĮçĤĞĻĞ ĨĹ IJIJĻ NIJĮĦf
        ĿĞĬĨĤĞĻĨIJĮçĹĻĦĴĹ ł ĝ

        ƒ FIJĹ ĠĦĻĻĦĹ ĹĦĞĤĞĠĨĬĨĻÑ
        XçĿĞĬd ÑçĿĞĬ ł ĿĞĬĨĤĞĻĨIJĮçĤĞĻĞ

    ƒ CĞĬĢŁĬĞĻĦ IJŁİĠĦĹ IJÌ ĹĻĦĴĹ
    ĨÌ ĠĞĻĢĨçĹĨÑĦ ĨĹ IJIJĻ NIJĮĦf
        ĻĹĞĨĮçĹĻĦĴĹ ł ĬĦĮâXã ww ĠĞĻĢĨçĹĨÑĦ
        ƒ DĨĿĨĤĨĮÍ ĹIJŁĮĤĹ ĤIJŃĮg IÌ ĻĨĦĹĦ ĞĹĦ ĹIJİĦ ĹĦİĞĨĮĨĮÍ
        ƒ ĤĞĻĞ ĠŁĻ IJIJĻ Ğ ÌŁĬĬ ĠĞĻĢĨd ĻĨĨĹ ŃIJĮďĻ ĨĮĢĬŁĤĦ ĨĻ
        ƒ AĤĤ ẀĝẀ ĻIJ ĨĮĢĬŁĤĦ ĻĨĨĹ IJIJĻ ÌŁĬĬ ĠĞĻĢĨ
        ĨÌ ĻĹĞĨĮçĹĻĦĴĹ ấ ĠĞĻĢĨçĹĨÑĦ l· ĬĦĮâXãƒ
            ĻĹĞĨĮçĹĻĦĴĹ κł ĝ

        ĨÌ ĿĞĬĨĤĞĻĨIJĮçĤĞĻĞ ĨĹ IJIJĻ NIJĮĦƒ
            ĿĞĬĨĤĞĻĨIJĮçĹĻĦĴĹ ł ĬĦĮâXçĿĞĬã ww ĠĞĻĢĨçĹĨÑĦ
```

```
        ƒ DĨĿĨĤĨĮĺ ĹIJŁĮĤĿ ĤIJŃĮg IÌ ĻĨĦĹĦ ĞĹĦ ĿIJİĦ ĹĦİĞĨĮĨĮĺ
        ƒ ĤĞĻĞ ĠŁĻ ĮIJĹ ÌŁĬĬ ĠĞĻĢĨd ĻĨĨĿ ŃIJĮďĻ ĨĮĢĬŁĤĦ ĨĻ
        ƒ AĤĤ ẀĝẀ ĻIJ ĨĮĢĬŁĤĦ ĻĨĨĿ ĮIJĻ ÌŁĬĬ ĠĞĻĢĨ
        ĨÌ ĿĞĬĨĤĞĻĨIJĮçĹĻĦĴĿ ắ ĠĞĻĢĨçĹĨŅĦ l· ĬĦĮâXçĿĞĬãf
            ĿĞĬĨĤĞĻĨIJĮçĹĻĦĴĿ κł ĝ

ƒ MĞĨĮ ĻĹĞĨĮĨĮĺ ĬIJIJĴ
ÌIJĹ ĦĴIJĢĨ ĨĮ ĹĞĮĺĦâĝd ĦĴIJĢĨĿκĝãf

    ƒ PĹĨĮĻ ĦĴIJĢĨ ĮŁĠĦĹ
    ĴĹĨĮĻâÌďĦĴIJĢĨf ăĦĴIJĢĨåďã

    ƒ RĦĹĦĻ ĞĢĢŁĠŁĞĻĦĤ ĿĞĬŁĦĹ ĨĮ ĬIJĹĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĹ
    ĹĦĬÌgĬIJĹĹgĮĦŃçĴĞĹĹâã
    ĹĦĬÌgĞĢĢŁĹĞĢÑgĮĦŃçĴĞĹĹâã

    ƒ IĻĦĹĞĻĦ IJĿĦĹ ĹĻĦĴĿ
    ÌIJĹ ĹĻĦĴ ĨĮ ĹĞĮĺĦâĻĹĞĨĮçĹĻĦĴĿãf

        ƒ IÌ ĠĞĻĢĨ ĹĨŅĦ ĨĿ ĮIJĻ ĹĦĻ h
        ƒ ĻĹĞĨĮ ŁĹĨĮĺ IJĮĦ ĹĻĦĴ ĞĮĤ ÌŁĬĬ ĤĞĻĞĹĦĻ
        ĨÌ ĠĞĻĢĨçĹĨŅĦ ĨĿ NIJĮĦf
            ĠĞĻĢĨçX ł X
            ĠĞĻĢĨçÑ ł Ñ

        ƒ OĻĨĦĹŃĨĹĦ ĹĬĨĢĦ Ğ ĠĞĻĢĨ
        ĦĬĹĦf
            ĠĞĻĢĨçX ł XäĹĻĦĴắĠĞĻĢĨçĹĨŅĦfâĹĻĦĴκĝãắĠĞĻĢĨçĹĨŅĦã
            ĠĞĻĢĨçÑ ł ÑäĹĻĦĴắĠĞĻĢĨçĹĨŅĦfâĹĻĦĴκĝãắĠĞĻĢĨçĹĨŅĦã

        ƒ PĦĹÌIJĹĬ ĻĨĦ ÌIJĹŃĞĹĤ ĴĞĹĹ
        IJŁĻĴŁĻ ł ĹĦĬÌgÌIJĹŃĞĹĤâĠĞĻĢĨçXd ĻĹĞĨĮĨĮĺłTĹŁĦã

        ƒ CĞĬĢŁĬĞĻĦ ĬIJĹĹ
        ĤĞĻĞçĬIJĹĹd ĹĦĬŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ ł ċ
            ĹĦĬÌgĬIJĹĹgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ĠĞĻĢĨçÑd
                                ĨĮĢĬŁĤĦçĹĦĬŁĬĞĹĨŅĞĻĨIJĮłTĹŁĦã
        ĬIJĹĹ ł ĤĞĻĞçĬIJĹĹ κ ĹĦĬŁĬĞĹĨŅĞĻĨIJĮçĬIJĹĹ

        ƒ GĦĻ ĴĹĦĤĨĢĻĨIJĮĹ ĞĮĤ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĴĹĦĤĨĢĻĨIJĮĹ ł ĹĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĿĞĻĨIJĮgĴĹĦĤĨĢĻĨIJĮĹâ
                            IJŁĻĴŁĻã
        ĞĢĢŁĹĞĢÑ ł ĹĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦĤĨĢĻĨIJĮĹd
                                            ĠĞĻĢĨçÑã

        ƒ PĦĹÌIJĹĬ ĠĞĢĨŃĞĹĤ ĴĞĹĹ
        ĹĦĬÌgĠĞĢĨŃĞĹĤâIJŁĻĴŁĻd ĠĞĻĢĨçÑã
```

```
    # OĴĻĨĬĨŅĦ âŁĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĽã
    ĽĦĬÌgIJĴĻĨĬĨŅĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĽâã
    ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽf
        ĽĦĬÌgIJĴĻĨĬĨŅĦĹgŁĴĤĞĻĦçĴĞĹĞĬĽâĬĞÑĦĹã
    ĽĦĬÌgIJĴĻĨĬĨŅĦĹgĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞĬĽâã

    # PĹĨĮĻ Ğ ĽŁĬĬĞĹÑ
    ĨÌ ĮIJĻ ĽĻĦĴ ķ ĴĹĨĮĻçĦŁĦĹÑ IJĹ ĽĻĦĴ łł ĻĹĞĨĮçĽĻĦĴĽ h ĝf
        ĴĹĨĮĻâÌďĽĻĦĴf ăĽĻĦĴåd ď κ
              ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĽĽf ăĬIJĽĽfgġÌå âď κ
              ÌďĤĞĻĞçĬIJĽĽf ăĤĞĻĞçĬIJĽĽfgġÌåd ď κ
              ÌďĹĦÍçĬIJĽĽf ăĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽfgġÌåãd ď κ
              ÌďĬĹf ăĽĦĬÌgIJĴĻĨĬĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

# GĦĻ ĞĮĤ ĴĹĨĮĻ ĦĴIJĢÎ ĬIJĽĽ ĞĮĤ ĞĢĢŁĹĞĢÑ
ĦĴIJĢÎçĤĞĻĞçĬIJĽĽd ĦĴIJĢÎçĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ ł ċ
    ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦçĞĢĢŁĬŁĬĞĻĦĤâ
        ĨĮĢĬŁĤĦçĹĦÍŁĬĞĹĨŅĞĻĨIJĮłTĹŁĦã
ĦĴIJĢÎçĬIJĽĽ ł ĦĴIJĢÎçĤĞĻĞçĬIJĽĽ κ ĦĴIJĢÎçĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽ
ĦĴIJĢÎçĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦçĞĢĢŁĬŁĬĞĻĦĤâã

ĴĹĨĮĻâÌďĻĹĞĨĮĨĮÍd ď κ
      ÌďĞĢĢf ăĦĴIJĢÎçĞĢĢŁĹĞĢÑfgġÌåd ď κ
      ÌďĬIJĽĽf ăĦĴIJĢÎçĬIJĽĽfgġÌå âď κ
      ÌďĤĞĻĞçĬIJĽĽf ăĦĴIJĢÎçĤĞĻĞçĬIJĽĽfgġÌåd ď κ
      ÌďĹĦÍçĬIJĽĽf ăĦĴIJĢÎçĹĦÍŁĬĞĹĨŅĞĻĨIJĮçĬIJĽĽfgġÌåãd ď κ
      ÌďĬĹf ăĽĦĬÌgIJĴĻĨĬĨŅĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

# IÌ ĻĤĦĹĦ ĨĽ ĻĤĦ ŁĞĬĨĤĞĻĨIJĮ ĤĞĻĞ
ĨÌ ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞ ĨĽ ĮIJĻ NIJĮĦf

    # RĦĽĦĻ ĞĢĢŁĬŁĬĞĻĦĤ ŁĞĬŁĦĽ ĨĮ ĬIJĽĽ
    # ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĽ
    ĽĦĬÌgĬIJĽĽgĮĦŃçĴĞĽĽâã
    ĽĦĬÌgĞĢĢŁĹĞĢÑgĮĦŃçĴĞĽĽâã

    # IĻĦĹĞĻĦ IJŁĦĹ ĽĻĦĴĽ
    ÌIJĹ ĽĻĦĴ ĨĮ ĹĞĮĬĦâŁĞĬĨĤĞĻĨIJĮçĽĻĦĴĽãf

        # IÌ ĠĞĻĢĤ ĽĨŅĦ ĨĽ ĮIJĻ ĽĦĻ h
        # ĻĹĞĨĮ ŁĽĨĮĬ IJĮĦ ĽĻĦĴ ĞĮĤ ÌŁĬĬ ĤĞĻĞĽĦĻ
        ĨÌ ĠĞĻĢĤçĽĨŅĦ ĨĽ NIJĮĦf
            ĠĞĻĢĤçX ł XçŁĞĬ
            ĠĞĻĢĤçÑ ł ÑçŁĞĬ
```

```
            ƒ OĻÎĦĹŃĨĽĦ ĽĬĨĢĦ Ğ ĠĞĻĢÎ
            ĦĬĽĦf
                ĠĞĻĢÎçX ł XçŁĞĬä
                    ĽĻĦĴǻĠĞĻĢÎçĽĨŅĦfâĽĻĦĴκĝãǻĠĞĻĢÎçĽĨŅĦ
                ã
                ĠĞĻĢÎçÑ ł ÑçŁĞĬä
                    ĽĻĦĴǻĠĞĻĢÎçĽĨŅĦfâĽĻĦĴκĝãǻĠĞĻĢÎçĽĨŅĦ
                ã

            ƒ PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÂ ĴĞĽĽ
            IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÂâĠĞĻĢÎçXd ĻĹĞĨĮĨĮÍłFĞĬĽĦã

            ƒ CĞĬĢŁĬĞĻĦ ĻÎĦ ĬIJĽĽ
            ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ĠĞĻĢÎçÑã

            ƒ GĦĻ ĴĹĦÂĨĢĻĨIJĮĽ ĞĮÂ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
            ĴĹĦÂĨĢĻĨIJĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨŁĞĻĨIJĮgĴĹĦÂĨĢĻĨIJĮĽâ
                                IJŁĻĴŁĻã
            ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦÂĨĢĻĨIJĮĽd ĠĞĻĢÎçÑã

        ƒ GĦĻ ĞĮÂ ĴĹĨĮĻ ŁĞĬĨÂĞĻĨIJĮ ĬIJĽĽ ĞĮÂ ĞĢĢŁĹĞĢÑ
        ŁĞĬĨÂĞĻĨIJĮçĬIJĽĽ ł ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦÂâã
        ŁĞĬĨÂĞĻĨIJĮçĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦÂâã

        ƒ PĹĨĮĻ Ğ ĽŁİİĞĹÑ
        ĴĹĨĮĻâÌďŁĞĬĨÂĞĻĨIJĮd ď κ
              ÌďĞĢĢf ăŁĞĬĨÂĞĻĨIJĮçĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĽĽf ăŁĞĬĨÂĞĻĨIJĮçĬIJĽĽfgġÌåďã

ƒ PĦĹÌIJĹİĽ ÌIJĹŃĞĹÂ ĴĞĽĽ
ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd Xd ĻĹĞĨĮĨĮÍãf

    ƒ CĞĬĬ ÌIJĹŃĞĹÂ İĦĻÎIJÂ IJĮ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
    ƒ ĻÎĨĽ ŃĨĬĬ ĽĦĻ ĻÎĦ IJŁĻĴŁĻ ĴĹIJĴĦĹĻÑ ĻÎĞĻ
    ƒ ĻÎĦ ÌĨĹĽĻ ĬĞÑĦĹ ĨĮ ðĴĹĦŁð IJĠÏĦĢĻ ĨĽ ĦŃĴĦĢĻĨĮÍ
    ĽĦĬÌgĨĮĴŁĻçĬĞÑĦĹgÌIJĹŃĞĹÂXd ĻĹĞĨĮĨĮÍã

    ƒ CĞĬĬ ÌIJĹŃĞĹÂ İĦĻÎIJÂ IJÌ ĦŁĦĹÑ IJĠÏĦĢĻ ĨĮ Ğ ĢÎĞĨĮ
    ƒ PĞĽĽ IJŁĻĴŁĻ IJÌ ĻÎĦ ĴĹĦŁĨIJŁĽ IJĠÏĦĢĻ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
    ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĬĞÑĦĹĽf
        ĬĞÑĦĹgÌIJĹŃĞĹÂâĬĞÑĦĹgĴĹĦŁgIJŁĻĴŁĻd ĻĹĞĨĮĨĮÍã

    ƒ ðĬĞÑĦĹð ĨĽ ĮIJŃ ĻÎĦ ĬĞĽĻ IJĠÏĦĢĻ ÌĹIJİ ĻÎĦ ĬĨĽĻd
    ƒ ĹĦĻŁĹĮ ĨĻĽ IJŁĻĴŁĻ
    ĹĦĻŁĹĮ ĬĞÑĦĹgIJŁĻĴŁĻ
```

```
ƒ PĦĹÌIJĹİĽ ĠĞĢĪŃĞĹĤ ĴĞĽĽ
ÎĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd IJŁĻĴŁĻd Ñãf

    ƒ IÌ ĽIJÌĻİĞŇ ĢĬĞĽĽĨÌÌĨĦĹ
    ĨÌ ĽĦĬÌgĽIJÌĻİĞŇçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻĴŁĻ ĨĽ ĮIJĻ NIJĮĦf
        ƒ FĨĹĽĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻĤIJĤ
        ƒ IJĮ ĻĤĦ ĢIJİĠĨĮĦĤ ĞĢĻĨĽĞĻĨIJĮwĬIJĽĽ
        ƒ ĻĤĨĽ ŃĨĬĬ ĽĦĻ ĤĨĮĴŁĻĽ ĴĹIJĴĦĹĻŇ
        ĽĦĬÌgĽIJÌĻİĞŇçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻĴŁĻgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd Ñã

        ƒ SĨĮĢĦ ŃĦďĬĬ ĮIJĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻĤIJĤ IJÌ ĻĤĦ ĬĞĽĻ ĬĞŇĦĹ
        ƒ ŃĤĨĢĤ ĨĽ SIJÌĻİĞŇ ĞĢĻĨĽĞĻĨIJĮ
        ƒ ĞĽ ŃĦ ŁĽĦĤ ĢIJİĠĨĮĦĤ ĞĢĻĨĽĞĻĨIJĮwĬIJĽĽ
        ƒ IJĠİĦĢĻd ĬĦĻďĽ ĽĦĻ ĤĨĮĴŁĻĽ ĨĮ ĻĤĨĽ IJĠİĦĢĻ
        ĽĦĬÌgĬĞŇĦĹĽähĝāgĤĨĮĴŁĻĽ ł ċ
            ĽĦĬÌgĽIJÌĻİĞŇçĢĬĞĽĽĨÌÌĨĦĹçIJŁĻĴŁĻgĤĨĮĴŁĻĽ

        ƒ CĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻĤIJĤ ÍIJĨĮÍ ĻĤĹIJŁÍĤ
        ƒ ĞĬĬ ĻĤĦ IJĠİĦĢĻĽ ĠŁĻ ĬĞĽĻ
        ƒ ĨĮ ĹĦĻĦĹĽĦĤ IJĹĤĦĹ ĴĞĽĽĨĮÍ ĤĨĮĴŁĻĽ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
        ÌIJĹ ĬĞŇĦĹ ĨĮ ĹĦĻĦĹĽĦĤâĽĦĬÌgĬĞŇĦĹĽäfhĝāãf
            ĬĞŇĦĹgĠĞĢĪŃĞĹĤâĬĞŇĦĹgĮĦŇĻgĤĨĮĴŁĻĽã

        ĹĦĻŁĹĮ

    ƒ FĨĹĽĻ ĢĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻĤIJĤ IJĮ ĻĤĦ ĬIJĽĽ
    ƒ ĻĤĨĽ ŃĨĬĬ ĽĦĻ ĤĨĮĴŁĻĽ ĴĹIJĴĦĹĻŇ ĻĤĞĻ ĻĤĦ ĬĞĽĻ
    ƒ ĬĞŇĦĹ ŃĨĬĬ ĻĹŇ ĻIJ ĞĢĢĦĽĽ ĽĤIJĹĻĬŇ
    ĽĦĬÌgĬIJĽĽgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd Ñã

    ƒ CĞĬĬ ĠĞĢĪŃĞĹĤ İĦĻĤIJĤ ÍIJĨĮÍ ĻĤĹIJŁÍĤ ĞĬĬ ĻĤĦ IJĠİĦĢĻĽ
    ƒ ĨĮ ĹĦĻĦĹĽĦĤ IJĹĤĦĹ ĴĞĽĽĨĮÍ ĤĨĮĴŁĻĽ ĞĽ Ğ ĴĞĹĞİĦĻĦĹ
    ÌIJĹ ĬĞŇĦĹ ĨĮ ĹĦĻĦĹĽĦĤâĽĦĬÌgĬĞŇĦĹĽãf
        ĬĞŇĦĹgĠĞĢĪŃĞĹĤâĬĞŇĦĹgĮĦŇĻgĤĨĮĴŁĻĽã


ƒ LIJĞĤĽ Ğ MNIST ĤĞĻĞĽĦĻ
ÎĦÌ ĬIJĞĤçİĮĨĽĻçĤĞĻĞĽĦĻâĤĞĻĞĽĦĻd ĴĞĻĨãf

    ƒ SĢĞĮ ĞĬĬ ĻĤĦ ĤĨĹĦĢĻIJĹĨĦĽ ĞĮĤ ĢĹĦĞĻĦ Ğ ĬĨĽĻ IJÌ ĬĞĠĦĬĽ
    ĬĞĠĦĬĽ ł IJĽgĬĨĽĻĤĨĹâIJĽgĴĞĻĨgĬIJĨĮâĴĞĻĨd ĤĞĻĞĽĦĻãã

    ƒ CĹĦĞĻĦ ĬĨĽĻĽ ÌIJĹ ĽĞİĴĬĦĽ ĞĮĤ ĬĞĠĦĬĽ
    X ł āã
    Ñ ł āã
```

```
    # FIJĹ ĦĞĢÎ ĬĞĠĦĬ ÌIJĬĤĦĹ
    ÌIJĹ ĬĞĠĦĬ ĨĮ ĬĞĠĦĬĽf
        # AĮĤ ÌIJĹ ĦĞĢÎ ĨĬĞÍĦ ĨĮ ÍĨĿĦĮ ÌIJĬĤĦĹ
        ÌIJĹ ÌĨĬĦ ĨĮ IJĽgĬĨĽĻĤĬĹâIJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ĤĞĻĞĽĦĻd ĬĞĠĦĬããf
            # RĦĞĤ ĻÎĦ ĨĬĞÍĦ
            ĨĬĞÍĦ ł ĢĿğgĨĬĹĦĞĤâ
                        IJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ĤĞĻĞĽĦĻd ĬĞĠĦĬd ÌĨĬĦãd
                        ĢĿğgIMREADçUNCHANGEDã

            # AĮĤ ĞĴĴĦĮĤ ĨĻ ĞĮĤ Ğ ĬĞĠĦĬ ĻIJ ĻÎĦ ĬĨĽĻĽ
            XgĞĴĴĦĮĤâĨĬĞÍĦã
            ÑgĞĴĴĦĮĤâĬĞĠĦĬã

    # CIJĮĿĦĹĻ ĻÎĦ ĤĞĻĞ ĻIJ ĴĹIJĴĦĹ ĮŁĬĴÑ ĞĹĹĞÑĽ ĞĮĤ ĹĦĻŁĹĮ
    ĹĦĻŁĹĮ ĮĴgĞĹĹĞÑâXãd ĮĴgĞĹĹĞÑâÑãgĞĽĻÑĴĦâďŁĨĮĻíďã


# MNIST ĤĞĻĞĽĦĻ âĻĹĞĨĮ ĸ ĻĦĽĻã
ÂĦÌ ĢĹĦĞĻĦçĤĞĻĞçĬĮĨĽĻâĴĞĻÎãf

    # LIJĞĤ ĠIJĻÎ ĽĦĻĽ ĽĦĴĞĹĞĻĦĬÑ
    Xd Ñ ł ĬIJĞĤçĬĮĨĽĻçĤĞĻĞĽĦĻâďĻĹĞĨĮďd ĴĞĻÎã
    XçĻĦĽĻd ÑçĻĦĽĻ ł ĬIJĞĤçĬĮĨĽĻçĤĞĻĞĽĦĻâďĻĦĽĻďd ĴĞĻÎã

    # AĮĤ ĹĦĻŁĹĮ ĞĬĬ ĻÎĦ ĤĞĻĞ
    ĹĦĻŁĹĮ Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ


# CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ ł ĢĹĦĞĻĦçĤĞĻĞçĬĮĨĽĻâďÌĞĽÎĨIJĮçĬĮĨĽĻçĨĬĞÍĦĽďã

# SÎŁÌÌĬĦ ĻÎĦ ĻĹĞĨĮĨĮÍ ĤĞĻĞĽĦĻ
ĪĦÑĽ ł ĮĴgĞĹĹĞÑâĹĞĮÍĦâXgĽÎĞĴĦäflããã
ĮĴgĹĞĮĤIJĬgĽÎŁÌÌĬĦâĪĦÑĽã
X ł XäĪĦÑĽã
Ñ ł ÑäĪĦÑĽã

# SĢĞĬĦ ĞĮĤ ĹĦĽÎĞĴĦ ĽĞĬĴĬĦĽ
X ł âXgĹĦĽÎĞĴĦâXgĽÎĞĴĦäflãd hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġĝã h ĝĝìghã w ĝĝìgh
XçĻĦĽĻ ł âXçĻĦĽĻgĹĦĽÎĞĴĦâXçĻĦĽĻgĽÎĞĴĦäflãd hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġĝã h
             ĝĝìghã w ĝĝìgh

# IĮĽĻĞĮĻĨĞĻĦ ĻÎĦ ĬIJĤĦĬ
ĬIJĤĦĬ ł MIJĤĦĬâã
```

```
ƒ AĤĤ ĬĞÑĦĹĽ
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâXgĽĤĞĴĦäĝãd ĝğíãã
İIJĤĦĬgĞĤĤâAĢĻĨĿĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝğíãã
İIJĤĦĬgĞĤĤâAĢĻĨĿĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝflãã
İIJĤĦĬgĞĤĤâAĢĻĨĿĞĻĨIJĮçSIJĬĻİĞÑâãã

ƒ SĦĻ ĬIJĽĽd IJĴĻĨİĨĨĢĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĽ
İIJĤĦĬgĽĦĻâ
    ĬIJĽĽłLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâãd
    IJĴĻĨİĨĨĢĦĹłOĴĻĨİĨĨĢĦĹçAĤĞİâĤĦĢĞÑłĝĦhġãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâã
ã

ƒ FĨĮĞĬĨĢĦ ĻĨĦ İIJĤĦĬ
İIJĤĦĬgĬĨĮĞĬĨĢĦâã

ƒ TĹĞĨĮ ĻĨĦ İIJĤĦĬ
İIJĤĦĬgĻĹĞĨĮâXd Ñd ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞłâXçĻĦĽĻd ÑçĻĦĽĻãd
            ĦĴIJĢĴĽłĝfld ĠĞĻĢĴçĽĨĢĦłĝğíd ĴĹĨĮĻçĦŁĦĹÑłĝflflã
```

Supplementary Material: https://nnfs.io/ch19
Chapter code, further resources, and errata for this chapter.

Chapter 20

Model Evaluation

In Chapter 11, Testing or Out-of-Sample Data, we covered the differences between validation and testing data. With our model up to this point, we've validated during training, but currently have no great way to run a test on data or perform a prediction. To begin, we're going to add a new `HLǦĬŁǦĻH` method to the `MIJĤHĬ` class:

```
ń EĽǦĬŁǦĻHĽ ĻÎH ÌIJĤHĬ ŁĽĨĮÍ ĴǦĽĽHĤ ĨĮ ĤǦĻǦĽHĻ
ĤHÌ HĽǦĬŁǦĻHâĽHĬÌd XçĽǦĬd ÑçĽǦĬd ǻd ĠǦĻǦÎçĽĨĻNHłNIJĮHãf
```

This method takes in samples (*XçĽǦĬ*), target outputs (*ÑçĽǦĬ*), and an optional batch size. First, we calculate the number of steps given the length of the data and the *ĠǦĻǦÎçĽĨĻNH* argument. This is the same as in the `ĻĹǦĨĮ` method:

```
        # DHÌĞŁĬĻ ŁĞĬŁH ĨÌ ĠĞĻĢÎ ĹĨŅH ĨĹ ĮIJĻ ĠHĨĮÍ ĹHĻ
        ŁĞĬĨĤĞĻĨIJĮçĹĻHĴĹ ł ĝ

        # CĞĬĢŁĬĞĻH ĮŁĬĠHĹ IJÌ ĹĻHĴĹ
        ĨÌ ĠĞĻĢÎçĹĨŅH ĨĹ ĮIJĻ NIJĮHf
            ŁĞĬĨĤĞĻĨIJĮçĹĻHĴĹ ł ĬHĮâXçŁĞĬã ww ĠĞĻĢÎçĹĨŅH
            # DĨŁĬĤĨĮÍ ĹIJŁĮĤĹ ĤIJŃĮg IÌ ĻÎHĹH ĞĹH ĹIJĬH ĹHĬĞĨĮĨĮÍ
            # ĤĞĻĞd ĠŁĻ ĮIJĻ Ğ ÌŁĬĬ ĠĞĻĢÎd ĻÎĨĹ ŃIJĮďĻ ĨĮĢĬŁĤH ĨĻ
            # AĤĤ Ẁĝẁ ĻIJ ĨĮĢĬŁĤH ĻÎĨĹ ĮIJĻ ÌŁĬĬ ĠĞĻĢÎ
            ĨÌ ŁĞĬĨĤĞĻĨIJĮçĹĻHĴĹ ǻ ĠĞĻĢÎçĹĨŅH l· ĬHĮâXçŁĞĬãf
                ŁĞĬĨĤĞĻĨIJĮçĹĻHĴĹ κł ĝ
```

Then, we're going to move a chunk of code from the MIJĤHĬ class' ĻĹĞĨĮ method:

```
# MIJĤHĬ ĢĬĞĹĹ
ĢĬĞĹĹ MIJĤHĬf
    ggg
     ĤHÌ ĻĹĞĨĮâĹHĬÌd Xd Ñd ǻd HĴIJĢÎĹłĝd ĠĞĻĢÎçĹĨŅHłNIJĮHd
            ĴĹĨĮĻçHŁHĹÑłĝd ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞłNIJĮHãf
        ggg
            ggg
            # IÌ ĻÎHĹH ĨĹ ĻÎH ŁĞĬĨĤĞĻĨIJĮ ĤĞĻĞ
            ĨÌ ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞ ĨĹ ĮIJĻ NIJĮHf

                # RHĹHĻ ĞĢĢłĬłĬĞĻHĤ ŁĞĬłHĹ ĨĮ ĬIJĹĹ
                # ĞĮĤ ĞĢĢłĹĞĢÑ IJĠĬHĢĻĹ
                ĹHĬÌgĬIJĹĹgĮHŃçĴĞĹĹâã
                ĹHĬÌgĞĢĢłĹĞĢÑgĮHŃçĴĞĹĹâã

                # IĻHĹĞĻH IJŁHĹ ĹĻHĴĹ
                ÌIJĹ ĹĻHĴ ĨĮ ĹĞĮÍHâŁĞĬĨĤĞĻĨIJĮçĹĻHĴĹãf

                    # IÌ ĠĞĻĢÎ ĹĨŅH ĨĹ ĮIJĻ ĹHĻ h
                    # ĻĹĞĨĮ łĹĨĮÍ IJĮH ĹĻHĴ ĞĮĤ ÌŁĬĬ ĤĞĻĞĹHĻ
                    ĨÌ ĠĞĻĢÎçĹĨŅH ĨĹ NIJĮHf
                        ĠĞĻĢÎçX ł XçŁĞĬ
                        ĠĞĻĢÎçÑ ł ÑçŁĞĬ

                    # OĻÎHĹŃĨĹH ĹĬĨĢH Ğ ĠĞĻĢÎ
                    HĬĹHf
                        ĠĞĻĢÎçX ł XçŁĞĬä
                            ĹĻHĴǻĠĞĻĢÎçĹĨŅHfâĹĻHĴκĝãǻĠĞĻĢÎçĹĨŅH
                        ã
                        ĠĞĻĢÎçÑ ł ÑçŁĞĬä
                            ĹĻHĴǻĠĞĻĢÎçĹĨŅHfâĹĻHĴκĝãǻĠĞĻĢÎçĹĨŅH
                        ã
```

```
            ƒ PHĹÌIJĹİ ĻÎH ÌIJĹŃĞĹÂ ĴĞĽĽ
            IJŁĻĴŁĻ ł ĽHĬÌgÌIJĹŃĞĹÂâĠĞĻĢÎçXd ĻĹĞĨĮĨĮÍŁFĞĬĽHã

            ƒ CĞĬĢŁĬĞĻH ĻÎH ĬIJĽĽ
            ĽHĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻHâIJŁĻĴŁĻd ĠĞĻĢÎçÑã

            ƒ GHĻ ĴĹHÂĨĢĻĨIJĮĽ ĞĮÂ ĢĞĬĢŁĬĞĻH ĞĮ ĞĢĢŁĹĞĢÑ
            ĴĹHÂĨĢĻĨIJĮĽ ł ĽHĬÌgIJŁĻĴŁĻçĬĞÑHĹçĞĢĻĨĻĞĻĨIJĮgĴĹHÂĨĢĻĨIJĮĽâ
                                IJŁĻĴŁĻã
            ĽHĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻHâĴĹHÂĨĢĻĨIJĮĽd ĠĞĻĢÎçÑã

        ƒ GHĻ ĞĮÂ ĴĹĨĮĻ ĻĞĬĨÂĞĻĨIJĮ ĬIJĽĽ ĞĮÂ ĞĢĢŁĹĞĢÑ
        ĻĞĬĨÂĞĻĨIJĮçĬIJĽĽ ł ĽHĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻHçĞĢĢŁİŁĬĞĻHÂâã
        ĻĞĬĨÂĞĻĨIJĮçĞĢĢŁĹĞĢÑ ł ĽHĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻHçĞĢĢŁİŁĬĞĻHÂâã

        ƒ PĹĨĮĻ Ğ ĽŁİĬĞĹÑ
        ĴĹĨĮĻâÌďĻĞĬĨÂĞĻĨIJĮd ď κ
              ÌďĞĢĢf ăĻĞĬĨÂĞĻĨIJĮçĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĽĽf ăĻĞĬĨÂĞĻĨIJĮçĬIJĽĽfgġÌåďã
```

We'll move that code, along with the code parts for the number of steps calculation and resetting accumulated loss and accuracy, to the `HĻĞĬŁĞĻH` method, making it:

```
ƒ EĻĞĬŁĞĻHĽ ĻÎH ÌIJÂHĬ ŁĽĨĮÍ ĴĞĽĽHÂ ĨĮ ÂĞĻĞĽHĻ
ÂHÌ HĻĞĬŁĞĻHâĽHĬÌd XçĻĞĬd ÑçĻĞĬd ǻd ĠĞĻĢÎçĽĨĢHŁNIJĮHãf

    ƒ DHÌĞŁĬĻ ĻĞĬŁH ĨÌ ĠĞĻĢÎ ĽĨĢH ĨĽ ĮIJĻ ĠHĨĮÍ ĽHĻ
    ĻĞĬĨÂĞĻĨIJĮçĽĻHĴĽ ł ĝ

    ƒ CĞĬĢŁĬĞĻH ĮŁİĠHĹ IJÌ ĽĻHĴĽ
    ĨÌ ĠĞĻĢÎçĽĨĢH ĨĽ ĮIJĻ NIJĮHf
        ĻĞĬĨÂĞĻĨIJĮçĽĻHĴĽ ł ĬHĮâXçĻĞĬã ww ĠĞĻĢÎçĽĨĢH
        ƒ DĨĻĨÂĨĮÍ ĹIJŁĮÂĽ ÂIJŃĮg IÌ ĻÎHĹH ĞĹH ĽIJİH ĹHİĞĨĮĨĮÍ
        ƒ ÂĞĻĞd ĠŁĻ IJĻ Ğ ÌŁĬĬ ĠĞĻĢÎd ĻÎĨĽ ŃIJĮďĻ ĨĮĢĬŁÂH ĨĻ
        ƒ AÂÂ ẀĝẀ ĻIJ ĨĮĢĬŁÂH ĻÎĨĽ IJĻ ÌŁĬĬ İĨĮĨĠĞĻĢÎ
        ĨÌ ĻĞĬĨÂĞĻĨIJĮçĽĻHĴĽ ǻ ĠĞĻĢÎçĽĨĢH l· ĬHĮâXçĻĞĬãf
            ĻĞĬĨÂĞĻĨIJĮçĽĻHĴĽ κł ĝ

    ƒ RHĽHĻ ĞĢĢŁİŁĬĞĻHÂ ĻĞĬŁHĽ ĨĮ ĬIJĽĽ
    ƒ ĞĮÂ ĞĢĢŁĹĞĢÑ IJĠĬHĢĻĽ
    ĽHĬÌgĬIJĽĽgĮHŃçĴĞĽĽâã
    ĽHĬÌgĞĢĢŁĹĞĢÑgĮHŃçĴĞĽĽâã

    ƒ IĻHĹĞĻH IJĻHĹ ĽĻHĴĽ
    ÌIJĹ ĽĻHĴ ĨĮ ĹĞĮÍHâĻĞĬĨÂĞĻĨIJĮçĽĻHĴĽãf
```

```
        # IÌ ĠĞĻĢÎ ĽÎŅĦ ÎĽ ĮIJĻ ĽĦĻ h
        # ĻĹĞÎĮ ŁĽÎĮÍ IJĮĦ ĽĻĦĴ ĞĮÂ ÌŁĬÎ ÂĞĻĞĽĦĻ
        ÎÌ ĠĞĻĢÎçĽÎŅĦ ÎĽ NIJĮĦf
            ĠĞĻĢÎçX ł XçĽĞĬ
            ĠĞĻĢÎçÑ ł ÑçĽĞĬ

        # OĻÎĦĹŃÎĽĦ ĽÎÎĢĦ Ğ ĠĞĻĢÎ
        ĦĬĽĦf
            ĠĞĻĢÎçX ł XçĽĞĬä
                ĽĻĦĴåĠĞĻĢÎçĽÎŅĦfâĽĻĦĴкĝãåĠĞĻĢÎçĽÎŅĦ
            ã
            ĠĞĻĢÎçÑ ł ÑçĽĞĬä
                ĽĻĦĴåĠĞĻĢÎçĽÎŅĦfâĽĻĦĴкĝãåĠĞĻĢÎçĽÎŅĦ
            ã

        # PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÂ ĴĞĽĽ
        IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÂâĠĞĻĢÎçXd ĻĹĞÎĮÎĮÍŁFĞĬĽĦã

        # CĞĬĢŁĬĞĻĦ ĻÎĦ ĬIJĽĽ
        ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ĠĞĻĢÎçÑã

        # GĦĻ ĴĹĦÂÎĢĻÎIJĮĽ ĞĮÂ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĴĹĦÂÎĢĻÎIJĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻÎĽĞĻÎIJĮgĴĹĦÂÎĢĻÎIJĮĽâ
                            IJŁĻĴŁĻã
        ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦÂÎĢĻÎIJĮĽd ĠĞĻĢÎçÑã

    # GĦĻ ĞĮÂ ĴĹÎĮĻ ĽĞĬÎÂĞĻÎIJĮ ĬIJĽĽ ĞĮÂ ĞĢĢŁĹĞĢÑ
    ĽĞĬÎÂĞĻÎIJĮçĬIJĽĽ ł ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦÂâã
    ĽĞĬÎÂĞĻÎIJĮçĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦÂâã

    # PĹÎĮĻ Ğ ĽŁİĬĞĹÑ
    ĴĹÎĮĻâÌďĽĞĬÎÂĞĻÎIJĮd ď к
        ÌďĞĢĢf ăĽĞĬÎÂĞĻÎIJĮçĞĢĢŁĹĞĢÑfgġÌåd ď к
        ÌďĬIJĽĽf ăĽĞĬÎÂĞĻÎIJĮçĬIJĽĽfgġÌåďã
```

Now, where that block of code once was in the MIJÂĦĬ class' ĻĹĞÎĮ method, we can call the new ĦĽĞĬŁĞĻĦ method:

```
# MIJÂĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJÂĦĬf
    ggg
     ÂĦÌ ĻĹĞÎĮâĽĦĬÌd Xd Ñd åd ĦĴIJĢÎĽŁĝd ĠĞĻĢÎçĽÎŅĦŁNIJĮĦd
            ĴĹÎĮĻçĦĽĦĹÑŁĝd ĽĞĬÎÂĞĻÎIJĮçÂĞĻĞŁNIJĮĦãf
        ggg
            ggg
             # IÌ ĻÎĦĹĦ ÎĽ ĻÎĦ ĽĞĬÎÂĞĻÎIJĮ ÂĞĻĞ
            ÎÌ ĽĞĬÎÂĞĻÎIJĮçÂĞĻĞ ÎĽ ĮIJĻ NIJĮĦf
```

```
                    # EŁĞĬŁĞĻĦ ĻÎĦ ĬIJĤĦĬf
                    ĽĦĬÌgĦŁĞĬŁĞĻĦâǻŁĞĬĨĤĞĻĨIJĮçĤĞĻĞd
                                  ĠĞĻĢÎçĽĨŅĦłĠĞĻĢÎçĽĨŅĦã
```

If you're confused about the ǻŁĞĬĨĤĞĻĨIJĮçĤĞĻĞ part, the asterisk, called the **starred expression**, unpacks the ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞ list into singular values. For a simple example of how this works:

```
Ğ ł âĝd ğã

ĤĦÌ ĻĦĽĻâĮĝd Įğãƒ
    ĴĹĨĮĻâĮĝd Įğã

ĻĦĽĻâǻĞã


ńńń
ĝ ğ
```

Now that we have this separate ĦŁĞĬŁĞĻĦ method, we can evaluate the model whenever we please — either during training or on-demand, by passing the validation or testing data. First, we'll create and train a model as usual:

```
# CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ ł ĢĹĦĞĻĦçĤĞĻĞçĬĮĨĽĻâďÌĞĽÎĨIJĮçĬĮĨĽĻçĨĬĞÍĦĽďã

# SÎŁÌÌĬĦ ĻÎĦ ĻĹĞĨĮĨĮÍ ĤĞĻĞĽĦĻ
ĪĦÑĽ ł ĮĴgĞĹĹĞÑâĹĞĮÍĦâXgĽÎĞĴĦäflāãã
ĮĴgĹĞĮĤIJĬgĽÎŁÌÌĬĦâĪĦÑĽã
X ł XäĪĦÑĽã
Ñ ł ÑäĪĦÑĽã

# SĢĞĬĦ ĞĮĤ ĹĦĽÎĞĴĦ ĽĞĬĴĬĦĽ
X ł âXgĹĦĽÎĞĴĦâXgĽÎĞĴĦäflād hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã h ĝğìghã w ĝğìgh
XçĻĦĽĻ ł âXçĻĦĽĻgĹĦĽÎĞĴĦâXçĻĦĽĻgĽÎĞĴĦäflād hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã h
              ĝğìghã w ĝğìgh

# IĮĽĻĞĮĻĨĞĻĦ ĻÎĦ ĬIJĤĦĬ
ĬIJĤĦĬ ł MIJĤĦĬâã

# AĤĤ ĬĞÑĦĹĽ
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâXgĽÎĞĴĦäĝād ĝğíãã
ĬIJĤĦĬgĞĤĤâAĢĻĨŁĞĻĨIJĮçRĦLUâãã
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝğíãã
ĬIJĤĦĬgĞĤĤâAĢĻĨŁĞĻĨIJĮçRĦLUâãã
ĬIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝflãã
ĬIJĤĦĬgĞĤĤâAĢĻĨŁĞĻĨIJĮçSIJÌĻĬĞŇâãã
```

```
ń SĦĻ ĬIJĹĽd IJĴĻĨĬĨÑĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĽ
ĬIJĤĦĬgĽĦĻâ
    ĬIJĹĽłLIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâãd
    IJĴĻĨĬĨÑĦĹłOĴĻĨĬĨÑĦĹçAĤĞĬâĤĦĢĞÑłĝĦhġãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâã
ã

ń FĨĮĞĬĨÑĦ ĻĤĦ ĬIJĤĦĬ
ĬIJĤĦĬgÌĨĮĞĬĨÑĦâã

ń TĹĞĨĮ ĻĤĦ ĬIJĤĦĬ
ĬIJĤĦĬgĻĹĞĨĮâXd Ñd ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞłâXçĻĦĽĻd ÑçĻĦĽĻãd
            ĦĴIJĢĤĽłĝfld ĠĞĻĢĤçĽĨÑĦłĝĝíd ĴĹĨĮĻçĦŁĦĹÑłĝflflã
```

We can then add code to evaluate. Right now, we don't have any specific testing data besides what we've used for validation data, but we can use this, for now, to test this method:

```
ĬIJĤĦĬgĦŁĞĬłĞĻĦâXçĻĦĽĻd ÑçĻĦĽĻã
```

Running this, we get:

```
ńńń
ggg
ĦĴIJĢĤf ĝfl
ĽĻĦĴf fld ĞĢĢf flgíîĝd ĬIJĹĽf flgĝĥġ âĤĞĻĞçĬIJĹĽf flgĝĥġd ĹĦÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflĝîĝĥġĝĝíííĥĝìĝflĝ
ĽĻĦĴf ĝflfld ĞĢĢf flgííġd ĬIJĹĽf flgĝĥì âĤĞĻĞçĬIJĹĽf flgĝĥìd ĹĦÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflĝíìîġĝĥîíìĥîĥġĝĥì
ĽĻĦĴf ĝflfld ĞĢĢf flgîĝĝd ĬIJĹĽf flgĝĝì âĤĞĻĞçĬIJĹĽf flgĝĝìd ĹĦÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflĝíġĝĥìíĝflġĥìflîflîġ
ĽĻĦĴf ġflfld ĞĢĢf flgíîíd ĬIJĹĽf flgĝíĝ âĤĞĻĞçĬIJĹĽf flgĝíĝd ĹĦÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflĝíĝĝĝĥĥflìġîíĥġĝĥĥ
ĽĻĦĴf ġflfld ĞĢĢf flgîĝġd ĬIJĹĽf flgĝîî âĤĞĻĞçĬIJĹĽf flgĝîîd ĹĦÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflĝìîîflġĝíìġîġĝĝíĥĥ
ĽĻĦĴf ġĥíd ĞĢĢf flgîĝìd ĬIJĹĽf flgĝîĝ âĤĞĻĞçĬIJĹĽf flgĝîĝd ĹĦÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflĝìĥììíĝĥíġîĥĝĝíĥ
ĻĹĞĨĮĨĮÍd ĞĢĢf flgíîġd ĬIJĹĽf flgĝîĝ âĤĞĻĞçĬIJĹĽf flgĝîĝd ĹĦÍçĬIJĹĽf flgflflflãd ĬĹf
flgflflflĝìĥììíĝĥíġîĥĝĝíĥ
ŁĞĬĨĤĞĻĨIJĮd ĞĢĢf flgíìġd ĬIJĹĽf flgġĥġ
ŁĞĬĨĤĞĻĨIJĮd ĞĢĢf flgíìġd ĬIJĹĽf flgġĥġ
```

The validation accuracy and loss are repeated twice and show the same values at the end since we're validating during the training and evaluating right after on the same data. You'll often train a model, tweak its hyperparameters, train it all over again, and so on, using training and validation data passed into the training method. Then, whenever you find the model and hyperparameters that appear to perform the best, you'll use that model on testing data and, in the future, to make predictions in production.

Next, we can also run evaluation on the training data:

```
ĮIJĤĦĬgĦL·ĞĬŁĞĻĦâXd Ñã
```

Running this prints:

```
ńńń
L·ĞĬĨĤĞĻĨIJĮd ĞĢĢf flgíîĥd ĬIJĹĹf flgğíĥ
```

“Validation” here means that we evaluated the model, but we have done this using the training data. We compare that to the result of training on this data which we have just performed:

```
ĻĹĞĨĮĨĮÍd ĞĢĢf flgíîġd ĬIJĹĹf flgğîĝ âĤĞĻĞçĬIJĹĹf flgğîĝd ĹĦÍçĬIJĹĹf flgflflflãd ĬĹf
flgflflflĝìĥìììíĝħíġîĥĝġíĥ
```

You may notice that, despite using the same dataset, there is some difference between accuracy and loss values. This difference comes from the fact that the model prints accuracy and loss accumulated during the epoch, while the model was still learning; thus, mean accuracy and loss differ from the evaluation on the training data that has been run after the last epoch of training. **Running evaluation on the training data at the end of the training process will return the final accuracy and loss.**

In the next chapter, we will add the ability to save and load our models; we’ll also construct a way to retrieve and set a model’s parameters.

Supplementary Material: https://nnfs.io/ch20
Chapter code, further resources, and errata for this chapter.

Chapter 21

Saving and Loading Models and Their Parameters

Retrieving Parameters

There are situations where we'd like to take a closer look into model parameters to see if we have dead or exploding neurons. To retrieve these parameters, we will iterate over the trainable layers, take their parameters, and put them into a list. The only trainable layer type that we have here is the *Dense* layer. Let's add a method to the `LǦÑĦĹçDĦĮĹĦ` class to retrieve parameters:

```
# DHIĻĽH ĬĞÑHĹ
ĢĬĞĽĽ LĞÑHĹçDHIĻĽHf
    ggg
    # RHĻĹĨHŁH ĬĞÑHĹ ĴĞĹĞİHĻHĹĽ
    ÂHÌ ÍHĻçĴĞĹĞİHĻHĹĽâĽHĬÌãf
        ĹHĻŁĹĮ ĽHĬÌgŃHĨÍÎĻĽd ĽHĬÌgĠĨĞĽHĽ
```

Within the `MIJÂHĬ` class, we'll add `ÍHĻçĴĞĹĞİHĻHĹĽ` method, which will iterate over the trainable layers of the model, run their `ÍHĻçĴĞĹĞİHĻHĹĽ` method, and append returned weights and biases to a list:

```
# MIJÂHĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJÂHĬf
    ggg
    # RHĻĹĨHŁHĽ ĞĮÂ ĹHĻŁĹĮĽ ĴĞĹĞİHĻHĹĽ IJÌ ĻĹĞĨĮĞĠĬH ĬĞÑHĹĽ
    ÂHÌ ÍHĻçĴĞĹĞİHĻHĹĽâĽHĬÌãf

        # CĹHĞĻH Ğ ĬĨĽĻ ÌIJĹ ĴĞĹĞİHĻHĹĽ
        ĴĞĹĞİHĻHĹĽ ł ää

        # IĻHĹĞĠĬH ĻĹĞĨĮĞĠĬH ĬĞÑHĹĽ ĞĮÂ ÍHĻ ĻÎHĨĹ ĴĞĹĞİHĻHĹĽ
        ÌIJĹ ĬĞÑHĹ ĨĮ ĽHĬÌgĻĹĞĨĮĞĠĬHçĬĞÑHĹĽf
            ĴĞĹĞİHĻHĹĽgĞĴĴHĮÂâĬĞÑHĹgÍHĻçĴĞĹĞİHĻHĹĽâãã

        # RHĻŁĹĮ Ğ ĬĨĽĻ
        ĹHĻŁĹĮ ĴĞĹĞİHĻHĹĽ
```

Now, after training a model, we can grab the parameters by running:

```
ĴĞĹĞİHĻHĹĽ ł ÌIJÂHĬgÍHĻçĴĞĹĞİHĻHĹĽâã
```

For example:

```
ﬁ CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ ł ĢĹĦĞĻĦçĤĞĻĞçİĮĨĽĻâďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽďã

ﬁ SÎŁÌÌĬĦ ĻÎĦ ĻĹĞĨĮĨĮÍ ĤĞĻĞĽĦĻ
ĪĦÑĽ ł ĮĴgĞĹĹĞÑâĹĞĮÍĦâXgĽÎĞĴĦäﬂāãã
ĮĴgĹĞĮĤIJİgĽÎŁÌÌĬĦâĪĦÑĽã
X ł XäĪĦÑĽā
Ñ ł ÑäĪĦÑĽā

ﬁ SĢĞĬĦ ĞĮĤ ĹĦĽÎĞĴĦ ĽĞİĴĬĦĽ
X ł âXgĹĦĽÎĞĴĦâXgĽÎĞĴĦäﬂād hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã h ĝğìghãã w ĝğìghã
XçĻĦĽĻ ł âXçĻĦĽĻgĹĦĽÎĞĴĦâXçĻĦĽĻgĽÎĞĴĦäﬂād hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã h
              ĝğìghã w ĝğìghã

ﬁ IĮĽĻĞĮĻĨĞĻĦ ĻÎĦ İIJĤĦĬ
İIJĤĦĬ ł MIJĤĦĬâã

ﬁ AĤĤ ĬĞÑĦĹĽ
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâXgĽÎĞĴĦäĝād ĝğíãã
İIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝğíãã
İIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝﬂãã
İIJĤĦĬgĞĤĤâAĢĻĨĽĞĻĨIJĮçSIJÌĻİĞÑâãã

ﬁ SĦĻ ĬIJĽĽd IJĴĻĨİĨĨĢĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠÏĦĢĻĽ
İIJĤĦĬgĽĦĻâ
    ĬIJĽĽŁLIJĽĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĽĽĦĮĻĹIJĴÑâãd
    IJĴĻĨİĨĢĦĹŁOĴĻĨİĨĢĦĹçAĤĞİâĤĦĢĞÑŁĝĦhġãd
    ĞĢĢŁĹĞĢÑŁAĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâã
ã

ﬁ FĨĮĞĬĨĢĦ ĻÎĦ İIJĤĦĬ
İIJĤĦĬgÌĨĮĞĬĨĢĦâã

ﬁ TĹĞĨĮ ĻÎĦ İIJĤĦĬ
İIJĤĦĬgĻĹĞĨĮâXd Ñd ĹĞĬĨĤĞĻĨIJĮçĤĞĻĞŁâXçĻĦĽĻd ÑçĻĦĽĻãd
            ĦĴIJĢÎĽŁĝﬂd ĠĞĻĢÎçĽĨĢĦŁĝĝíd ĴĹĨĮĻçĦĹĦĹÑŁĝﬂﬂã

ﬁ RĦĻĹĨĦĹĦ ĞĮĤ ĴĹĨĮĻ ĴĞĹĞİĦĻĦĹĽ
ĴĞĹĞİĦĻĦĹĽ ł İIJĤĦĬgÍĦĻçĴĞĹĞİĦĻĦĹĽâã
ĴĹĨĮĻâĴĞĹĞİĦĻĦĹĽã
```

This will look *something* like (we trim the output to save space):

```
ääĞĹĹĞÑâää flgflġĥġíħġğd  flgflflìîġìĝìd hflgflġĝġġğĝd gggd  flgflġğħìġğĥd
         hflgflflîġĥĝflìd  flgflĝíìğġîġād
        ä flgflġğíîġíġd  flgflflħîĝğġîd hflgflġġğġflîħd gggd  flgflğġħğìĥĥd
         hflgflflîflġħflğd  flgflflîìììğĥād
        ä flgflğĝíîflğğd hflgflĝġħğġìġd hflgflĝġġğíĝîd gggd  flgflĝġğflġġĥd
         hflgflğflíġġğìd  flgflğġîîĝĥìād
        gggd
        ä flgflĝġħîġì d hflgflğíħîflğìd hflgflğĝîííflîd gggd  flgflĝġĥîğîĥd
         hflgflğġġĥíğġd  flgflflîġĥħġġād
        ähflgflflflîflĝġîd  flgflĝflíğĝíğd hflgflħflĝġíflħd gggd  flgflflìflġġĥġd
         hflgflflġîflîġ d  flgflflġĝĝĥìĝād
        ä flgflġħħflflíğd hflgflflíflîħflìd hflgflğìġìĝġĝd gggd  flgflğĝĝħĥíğd
         hflgflĝìĝflĥííd  flgflĝĥìíġĝġāād ĤĻÑĴĦłÌĬIJĞĻġğãd ĞĹĹĞÑâäähğĝġĥflĥìġìĦhflğd
ĥġġflflîflğĝġĦhflġd  ğgîĝġflìġġíĦhflğd
         hĝgflġġğġħîĝĦhflğd hîgĥğíğğğġîĦhflġd hĝgġíĝflîìğíĦhflğd
        gggd
          flgflġĝĥíĥîĝd hflgflĝħĝġflííd hflgflĝġġġflġ d  flgflflìflíġîğd  flgflğíġìğî d
          flgflflġġħğììd hflgflflflíĥġíġd  flgflflĝħġíĝîāād ĤĻÑĴĦłÌĬIJĞĻġğãād
âĞĹĹĞÑâäähflgflflĝîħĥììd hflgflflġġĥġğîd hflgflĝġħğíĥĝd gggd  flgflflġîìflğíd
          flgflflflğìíĝħd  flgflflġğììĥĥād
        ä flgflġġġííğîd hflgflîĝîìíflġd  flgflğíîìġĥğd gggd hflgĝĝîğflğħġd
          flgflġíflíğîħd hflgflflĥġħĝġħād
        ä flgflġĝġħġġġd hflgflġħġìĥğîd  flgflġîìġġflĥd gggd hflgĝġĥħġħîíd
         hflgflíğĥîĝîìd hflgflğġħìğííād
        gggd
        ä flgflġġîĥíĥħd  flgflġîflğĥîìd  flgflflğíííġ d gggd hflgĝflflĝħííğd
         hflgĝĝġĥħĥġğd  flgflĥíħħġġġād
        ähflgflflíĥìííîd hflgflğħĝğħìħd hflgflĝflĥflííìĝd gggd hflgflflĥĥĝġğíd
         hflgflĝġġğĝĝĝd hflgflflîĝħġíğād
        ähflgğflġġġflíĥd hflgflĝġíġħîíd hflgflîġğĝĝĥğd gggd  flgflğĝĝġġĥħd
         hflgflìħğĥflġ d  flgflġħflflħĝĥāād ĤĻÑĴĦłÌĬIJĞĻġğãd ĞĹĹĞÑâ�hflgflĝflġġġġ d
hflgflflĝĥíġĝĝd  flgflğğħíĥíìd hflgflğġĥğîíĥd hflgflğĝĝĝĝğħd
         hflgflflìììħĝĝd  flgflflìîĥflğíd hflgflflħğğíìğd  flgflħîĝíìġĥd hflgflflìġġġìiāād
       ĤĻÑĴĦłÌĬIJĞĻġğãāā
```

Setting Parameters

If we have a method to get parameters, we will likely also want to have a method that will set parameters. We'll do this similar to how we setup the ÍĦĻçĴĞĹĞİĦĻĦĹĽ method, starting with the LĞÑĦĹçDĦĮĽĦ class:

```
# DĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĹĽ LĞÑĦĹçDĦĮĽĦf
    ggg
    # SĦĻ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĹĦĹ ĨĮ Ğ ĬĞÑĦĹ ĨĮĹĻĞĮĢĦ
    ÂĦÌ ĽĦĻçĴĞĹĞİĦĻĦĹĽâĹĦĬÌd ŃĦĨÍÎĻĽd ĠĨĞĹĦĹãf
        ĽĦĬÌgŃĦĨÍÎĻĽ ł ŃĦĨÍÎĻĽ
        ĽĦĬÌgĠĨĞĹĦĹ ł ĠĨĞĹĦĹ
```

Then we can update the MIJÂĦĬ class:

```
# MIJÂĦĬ ĢĬĞĹĽ
ĢĬĞĹĽ MIJÂĦĬf
    ggg
    # UĴÂĞĻĦĹ ĻÎĦ ĬIJÂĦĬ ŃĨĻÎ ĮĦŃ ĴĞĹĞİĦĻĦĹĽ
    ÂĦÌ ĽĦĻçĴĞĹĞİĦĻĦĹĽâĹĦĬÌd ĴĞĹĞİĦĻĦĹĽãf

        # IĻĦĹĞĻĦ IJĽĦĹ ĻÎĦ ĴĞĹĞİĦĻĦĹĽ ĞĮĤ ĬĞÑĦĹĽ
        # ĞĮĤ ŁĴÂĞĻĦ ĦĞĢÎ ĬĞÑĦĹĽ ŃĨĻÎ ĦĞĢÎ ĽĦĻ IJÌ ĻÎĦ ĴĞĹĞİĦĻĦĹĽ
        ÌIJĹ ĴĞĹĞİĦĻĦĹçĽĦĻd ĬĞÑĦĹ ĨĮ ŅĨĴâĴĞĹĞİĦĻĦĹĽd
                                        ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽãf
            ĬĞÑĦĹgĽĦĻçĴĞĹĞİĦĻĦĹĽâåĴĞĹĞİĦĻĦĹçĽĦĻã
```

We are also iterating over the trainable layers here, but what we are doing next needs a bit more explanation. First, the ŅĨĴâã function takes in iterables, like lists, and returns a new iterable with pairwise combinations of all the iterables passed in as parameters. In other words (and using our example), ŅĨĴâã takes a list of parameters and a list of layers and returns an iterator containing tuples of 0th elements of both lists, then the 1st elements of both lists, the 2nd elements from both lists, and so on. This way, we can iterate over parameters and the layer they belong to at the same time. As our parameters are a tuple of weights and biases, we will unpack them with a starred expression so that our LĞÑĦĹçDĦĮĽĦ method can take them as separate parameters. This approach gives us flexibility if we'd like to use layers with different numbers of parameter groups.

One difference that presents itself now is that this allows us to have a model that never needed an optimizer. If we don't train a model but, instead, load already trained parameters into it, we won't optimize anything. To account for this, we'll visit the ÌĨĮĞĬĨŅĦ method of the MIJĤĦĬ class, changing:

```
ħ MIJĤĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJĤĦĬf
    ggg
    ħ FĨĮĞĬĨŅĦ ĻĨĦ ĬIJĤĦĬ
    ĤĦÌ ÌĨĮĞĬĨŅĦâĽĦĬÌãf
        ggg
            ħ UĴĤĞĻĦ ĬIJĽĽ IJĠÏĦĢĻ ŃĨĻĨ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
            ĽĦĬÌgĬIJĽĽgĹĦĬĦĬĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽâ
                ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ
            ã
```

To (we added an ĨÌ statement to set a list of trainable layers to the loss function, only if this loss object exists):

```
ħ MIJĤĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJĤĦĬf
    ggg
    ħ FĨĮĞĬĨŅĦ ĻĨĦ ĬIJĤĦĬ
    ĤĦÌ ÌĨĮĞĬĨŅĦâĽĦĬÌãf
        ggg
            ħ UĴĤĞĻĦ ĬIJĽĽ IJĠÏĦĢĻ ŃĨĻĨ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
            ĨÌ ĽĦĬÌgĬIJĽĽ ĨĽ ĮIJĻ NIJĮĦf
                ĽĦĬÌgĬIJĽĽgĹĦĬĦĬĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽâ
                    ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽ
                ã
```

Next, we'll change the MIJĤĦĬ class' ĽĦĻ method to allow us to pass in only given parameters. We'll assign default values and add ĨÌ statements to use parameters only when they're present. To do that, we'll change:

```
    ħ SĦĻ ĬIJĽĽd IJĴĻĨĬĨŅĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ
    ĤĦÌ ĽĦĻâĽĦĬÌd ǵd ĬIJĽĽd IJĴĻĨĬĨŅĦĹd ĞĢĢŁĹĞĢÑãf
        ĽĦĬÌgĬIJĽĽ ł ĬIJĽĽ
        ĽĦĬÌgIJĴĻĨĬĨŅĦĹ ł IJĴĻĨĬĨŅĦĹ
        ĽĦĬÌgĞĢĢŁĹĞĢÑ ł ĞĢĢŁĹĞĢÑ
```

To:

```
# SHL ĬIJĹĹd IJĴĻĨĬĨŅHĹ ĞĮĤ ĞĢĢŁĹĞĢÑ
ĤHÌ ĹHĻâĹHĬÌd ǻd ĬIJĹĹłNIJĮHd IJĴĻĨĬĨŅHĹłNIJĮHd ĞĢĢŁĹĞĢÑłNIJĮHãf

    ĨÌ ĬIJĹĹ ĨĹ ĮIJĻ NIJĮHf
        ĹHĬÌgĬIJĹĹ ł ĬIJĹĹ

    ĨÌ IJĴĻĨĬĨŅHĹ ĨĹ ĮIJĻ NIJĮHf
        ĹHĬÌgIJĴĻĨĬĨŅHĹ ł IJĴĻĨĬĨŅHĹ

    ĨÌ ĞĢĢŁĹĞĢÑ ĨĹ ĮIJĻ NIJĮHf
        ĹHĬÌgĞĢĢŁĹĞĢÑ ł ĞĢĢŁĹĞĢÑ
```

We can now train a model, retrieve its parameters, create a new model, and set its parameters with those retrieved from the previously-trained model:

```
# CĹHĞĻH ĤĞĻĞĹHĻ
Xd Ñd XçĻHĹĻd ÑçĻHĹĻ ł ĢĹHĞĻHçĤĞĻĞçĬĮĨĹĻâďÌĞĹĨĨIJĮçĬĮĨĹĻçĨĬĞÍHĹďã

# SĨŁÌÌĬH ĻĨH ĻĹĞĨĮĨĮÍ ĤĞĻĞĹHĻ
ĨHÑĹ ł ĮĴgĞĹĹĞÑâĹĞĮÍHâXgĹĨĞĴHäflāãã
ĮĴgĹĞĮĤIJĬgĹĨŁÌÌĬHâĨHÑĹã
X ł XäĨHÑĹã
Ñ ł ÑäĨHÑĹã

# SĢĞĬH ĞĮĤ ĹHĹĨĞĴH ĹĞĬĴĬHĹ
X ł âXgĹHĹĨĞĴHâXgĹĨĞĴHäflād hĝãgĞĹĻÑĴHâĮĴgÌĬIJĞĻġğã h ĝğìghĥã w ĝğìghĥ
XçĻHĹĻ ł âXçĻHĹĻgĹHĹĨĞĴHâXçĻHĹĻgĹĨĞĴHäflād hĝãgĞĹĻÑĴHâĮĴgÌĬIJĞĻġğã h
              ĝğìghĥã w ĝğìghĥ

# IĮĹĻĞĮĻĨĞĻH ĻĨH ĬIJĤHĬ
ĬIJĤHĬ ł MIJĤHĬâã

# AĤĤ ĬĞÑHĹĹ
ĬIJĤHĬgĞĤĤâLĞÑHĹçDHĮĹHâXgĹĨĞĴHäĝād ĝğíãã
ĬIJĤHĬgĞĤĤâAĢĻĨĹĞĻĨIJĮçRHLUâãã
ĬIJĤHĬgĞĤĤâLĞÑHĹçDHĮĹHâĝğíd ĝğíãã
ĬIJĤHĬgĞĤĤâAĢĻĨĹĞĻĨIJĮçRHLUâãã
ĬIJĤHĬgĞĤĤâLĞÑHĹçDHĮĹHâĝğíd ĝflãã
ĬIJĤHĬgĞĤĤâAĢĻĨĹĞĻĨIJĮçSIJÌĻĬĞÑâãã

# SHĻ ĬIJĹĹd IJĴĻĨĬĨŅHĹ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬHĢĻĹ
ĬIJĤHĬgĹHĻâ
    ĬIJĹĹłLIJĹĹçCĞĻHÍIJĹĨĢĞĬCĹIJĹĹHĮĻĹIJĴÑâãd
    IJĴĻĨĬĨŅHĹłOĴĻĨĬĨŅHĹçAĤĞĬâĤHĢĞÑłĝHhǵãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçCĞĻHÍIJĹĨĢĞĬâã
ã
```

```
ﬠ FÕĮĞĬĨŅĦ ĻÎĦ İIJÂĦĬ
İIJÂĦĬgÌĨĮĞĬĨŅĦâã

ﬠ TĹĞĨĮ ĻÎĦ İIJÂĦĬ
İIJÂĦĬgĻĹĞĨĮâXd Ñd ĿĞĬĨÂĞĻĨIJĮçÂĞĻĞłâXçĻĦĹĻd ÑçĻĦĹĻãd
            ĦĴIJĢÎĹłĝﬁd ĠĞĻĢÎçĹĨŅĦłĝĞíd ĴĹĨĮĻçĦĿĦĹÑłĝﬁﬁã

ﬠ RĦĻĹĨĦĿĦ İIJÂĦĬ ĴĞĹĞİĦĻĦĹĹ
ĴĞĹĞİĦĻĦĹĹ ł İIJÂĦĬgÍĦĻçĴĞĹĞİĦĻĦĹĹâã

ﬠ NĦŃ İIJÂĦĬ

ﬠ IĮĹĻĞĮĻĨĞĻĦ ĻÎĦ İIJÂĦĬ
İIJÂĦĬ ł MIJÂĦĬâã

ﬠ AÂÂ ĬĞÑĦĹĹ
İIJÂĦĬgĞÂÂâLĞÑĦĹçDĦĮĹĦâXgĹÎĞĴĦäĝad ĝĞíãã
İIJÂĦĬgĞÂÂâAĢĻĨĿĞĻĨIJĮçRĦLUâãã
İIJÂĦĬgĞÂÂâLĞÑĦĹçDĦĮĹĦâĝĞíd ĝĞíãã
İIJÂĦĬgĞÂÂâAĢĻĨĿĞĻĨIJĮçRĦLUâãã
İIJÂĦĬgĞÂÂâLĞÑĦĹçDĦĮĹĦâĝĞíd ĝﬁãã
İIJÂĦĬgĞÂÂâAĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŃâãã

ﬠ SĦĻ ĬIJĹĹ ĞĮÂ ĞĢĢŁĹĞĢÑ IJĠÏĦĢĻĹ
ﬠ WĦ ÂIJ ĮIJĻ ĹĦĻ IJĴĻĨİĨĨĨĢĦĹ IJĠÏĦĢĻ ĻÎĨĹ ĻĨİĦ h ĻÎĦĹĦďĹ ĮIJ ĮĦĦÂ ĻIJ ÂIJ ĨĻ
ﬠ ĞĹ ŃĦ ŃIJĮďĻ ĻĹĞĨĮ ĻÎĦ İIJÂĦĬ
İIJÂĦĬgĹĦĻâ
    ĬIJĹĹłLIJĹĹçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĹĦĮĻĹIJĴÑâãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâã
ã

ﬠ FÕĮĞĬĨŅĦ ĻÎĦ İIJÂĦĬ
İIJÂĦĬgÌĨĮĞĬĨŅĦâã

ﬠ SĦĻ İIJÂĦĬ ŃĨĻÎ ĴĞĹĞİĦĻĦĹĹ ĨĮĹĻĦĞÂ IJÌ ĻĹĞĨĮĨĮÍ ĨĻ
İIJÂĦĬgĹĦĻçĴĞĹĞİĦĻĦĹĹâĴĞĹĞİĦĻĦĹĹã

ﬠ EĿĞĬŁĞĻĦ ĻÎĦ İIJÂĦĬ
İIJÂĦĬgĦĿĞĬŁĞĻĦâXçĻĦĹĻd ÑçĻĦĹĻã


ńńń
âİIJÂĦĬ ĻĹĞĨĮĨĮÍ IJŁĻĴŁĻ ĹĦİIJĿĦÂã
ĿĞĬĨÂĞĻĨIJĮd ĞĢĢf ﬁgíìġd ĬIJĹĹf ﬁgġĥġ
ĿĞĬĨÂĞĻĨIJĮd ĞĢĢf ﬁgíìġd ĬIJĹĹf ﬁgġĥġ
```

Saving Parameters

We'll extend this further now by actually saving the parameters into a file. To do this, we'll add a `ĽĞŁÇĴĞĹĞİĦĻĦĹĽ` method in the `MIJĤĦĬ` class. We'll use Python's built-in *pickle* module to serialize any Python object. Serialization is a process of turning an object, which can be of any abstract form, into a binary representation — a set of bytes that can be, for example, saved into a file. This serialized form contains all the information needed to recreate the object later. *Pickle* can either return the bytes of the serialized object or save them directly to a file. We'll make use of the latter ability, so let's import *pickle*:

```
ĨİĴIJĹĻ ĴĨĢĪĬĦ
```

Then we'll add a new method to the `MIJĤĦĬ` class. Before having *pickle* save our parameters into a file, we'll need to create a file-handler by opening a file in binary-write mode. We will then pass this handler along to the data into `ĴĨĢĪĬĦgĤŁİĴâã`. To create the file, we need a filename that we'll save the data into; we'll pass it in as a parameter:

```
ħ MIJĤĦĬ ĢĬĞĽĽ
ĢĬĞĽĽ MIJĤĦĬf
    ggg
    ħ SĞŁĦĽ ĻĨĦ ĴĞĹĞİĦĻĦĹĽ ĻIJ Ğ ÌĨĬĦ
    ĤĦÌ ĽĞŁĦÇĴĞĹĞİĦĻĦĹĽâĽĦĬÌd ĴĞĻĨãf

        ħ OĴĦĮ Ğ ÌĨĬĦ ĨĮ ĻĨĦ ĠĨĮĞĹÑhŃĹĨĻĦ İIJĤĦ
        ħ ĞĮĤ ĽĞŁĦ ĴĞĹĞİĦĻĦĹĽ ĻIJ ĨĻ
        ŃĨĻĨ IJĴĦĮâĴĞĻĨd ďŃĠďã ĞĽ Ìf
            ĴĨĢĪĬĦgĤŁİĴâĽĦĬÌgÍĦĻçĴĞĹĞİĦĻĦĹĽâãd Ìã
```

With this method, you can save the parameters of a trained model by running:

```
ÌIJĤĦĬgĽĞŁĦÇĴĞĹĞİĦĻĦĹĽâďÌĞĽĨĨIJĮçÌĮĨĽĻgĴĞĹÌĽďã
```

Loading Parameters

Presumably, if we are saving model parameters into a file, we would also like to have a way to load them from this file. Loading parameters is very similar to saving the parameters, just reversed. We'll open the file in a binary-read mode and have *pickle* read from it, deserializing parameters back into a list. Then we call the `ĽĦĻçĴĞĹĞİĦĻĦĹĽ` method that we created earlier and pass in the loaded parameters:

```
# LIJĞĤĽ ĻÎĦ ŃĦĨÍÎĻĽ ĞĮĤ ŁĴĤĞĻĦĽ Ğ İIJĤĦĬ ĨĮĽĻĞĮĢĦ ŃĨĻÎ ĻÎĦİ
ĤĦÌ ĬIJĞĤçĴĞĹĞİĦĻĦĹĽâĽĦĬÌd ĴĞĻÎãf

    # OĴĦĮ ÌĨĬĦ ĨĮ ĻÎĦ ĠĨĮĞĹÑhĹĦĞĤ İIJĤĦd
    # ĬIJĞĤ ŃĦĨÍÎĻĽ ĞĮĤ ŁĴĤĞĻĦ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
    ŃĨĻÎ IJĴĦĮâĴĞĻÎd ďĹĠďã ĞĽ Ìf
        ĽĦĬÌgĽĦĻçĴĞĹĞİĦĻĦĹĽâĴĨĢĨĬĦgĬIJĞĤâÌãã
```

We set up a model, load in the parameters file (we did not train this model), and test the model to check if it works:

```
# CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ ł ĢĹĦĞĻĦçĤĞĻĞçİĮĨĽĻâďÌĞĽÎĨIJĮçİĮĨĽĻçĨİĞÍĦĽďã

# SÎŁÌÌĬĦ ĻÎĦ ĻĹĞĨĮĨĮÍ ĤĞĻĞĽĦĻ
ĪĦÑĽ ł ĮĴgĞĹĹĞÑâĹĞĮÍĦâXgĽÎĞĴĦäflããã
ĮĴgĹĞĮĤIJİgĽÎŁÌÌĬĦâĪĦÑĽã
X ł XäĪĦÑĽã
Ñ ł ÑäĪĦÑĽã

# SĢĞĬĦ ĞĮĤ ĹĦĽÎĞĴĦ ĽĞİĴĬĦĽ
X ł âXgĹĦĽÎĞĴĦâXgĽÎĞĴĦäflãd hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã h ĝğìghãã w ĝğìghã
XçĻĦĽĻ ł âXçĻĦĽĻgĹĦĽÎĞĴĦâXçĻĦĽĻgĽÎĞĴĦäflãd hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã h
            ĝğìghãã w ĝğìghã

# IĮĽĻĞĮĻĨĞĻĦ ĻÎĦ İIJĤĦĬ
İIJĤĦĬ ł MIJĤĦĬâã
```

```
ﬠ AĤĤ ĬĞÑĦĹĽ
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâXgĽĤĞĴĦäĝãd ĝğíãã
İIJĤĦĬgĞĤĤâAĢĻĨĿĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝğíãã
İIJĤĦĬgĞĤĤâAĢĻĨĿĞĻĨIJĮçRĦLUâãã
İIJĤĦĬgĞĤĤâLĞÑĦĹçDĦĮĽĦâĝğíd ĝﬠãã
İIJĤĦĬgĞĤĤâAĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇâãã

ﬠ SĦĻ ĬIJĹĽ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠÏĦĢĻĽ
ﬠ WĦ ĤIJ ĮIJĻ ĽĦĻ IJĴĻĨĬĨŅĦĹ IJĠÏĦĢĻ ĻĨĨĽ ĻĨĬĦ h ĻĨĦĹĦďĽ ĮIJ ĮĦĦĤ ĻIJ ĤIJ ĨĻ
ﬠ ĞĽ ŃĦ ŃIJĮďĻ ĻĹĞĨĮ ĻĨĦ ĬIJĤĦĬ
İIJĤĦĬgĽĦĻâ
    ĬIJĹĽłLIJĹĽçCĞĻĦÍIJĹĨĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâãd
    ĞĢĢŁĹĞĢÑłAĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâã
ã

ﬠ FĨĮĞĬĨŅĦ ĻĨĦ ĬIJĤĦĬ
İIJĤĦĬgÌĨĮĞĬĨŅĦâã

ﬠ SĦĻ ĬIJĤĦĬ ŃĨĻĨ ĴĞĹĞĬĦĻĦĹĽ ĨĮĽĻĦĞĤ IJÌ ĻĹĞĨĮĨĮÍ ĨĻ
İIJĤĦĬgĬIJĞĤçĴĞĹĞĬĦĻĦĹĽâďÌĞĽĨĨIJĮçĬĮĨĽĻgĴĞĹĬĽďã

ﬠ EŁĞĬŁĞĻĦ ĻĨĦ ĬIJĤĦĬ
İIJĤĦĬgĦŁĞĬŁĞĻĦâXçĻĦĽĻd ÑçĻĦĽĻã


ńńń
ŁĞĬĨĤĞĻĨIJĮd ĞĢĢf ﬂgíìǵd ĬIJĹĽf ﬂgġĥǵ
```

While we can save and load model parameter values, we still need to define the model. It must be the exact configuration as the model that we're importing parameters from. It would be easier if we could save the model itself.

Saving the Model

Why didn't we save the whole model in the first place? Saving just weights versus saving the whole model has different use cases along with pros and cons. With saved weights, you can, for example, initialize a model with those weights, trained from similar data, and then train that model to work with your specific data. This is called **transfer learning** and is outside of the scope of this book. Weights can be used to visualize the model (like in some animations that we have created for the purpose of this book, starting from chapter 6), identify dead neurons, implement more complicated models (like **reinforcement learning**, where weights collected from multiple models are committed to a single network), and so on. A file containing just weights is also much smaller than an entire model. A model initialized from weights loads faster and uses less memory, as the optimizer and related parts are not created. One downside of loading just weights and biases is that the initialized model does not contain the optimizer's state. It is possible to train the model further, but it's more optimal to load a full model if we intend to train it. When saving the full model, everything related to it is saved as well; this includes the optimizer's state (that allows us to easily continue the training) and model's structure.

We'll create another method in the `MIJÂHĤĬ` class that we'll use to save the entire model. The first thing we'll do is make a copy of the model since we're going to edit it before saving, and we may also want to save a model during the training process as a **checkpoint**.

```
ħ SĞL·ĦĽ ĻÎĦ ĬIJÂĦĬ
ÂĦÌ ĽĞL·Ħâ*ĽĦĬÌ*d *ĴĞĻÎ*ãf

    ħ MĞĪĦ Ğ ÂĦĦĴ ĢIJĴÑ IJÌ ĢŁĹĹĦĮĻ ĬIJÂĦĬ ĨĮĽĻĞĮĢĦ
    ĬIJÂĦĬ ł ĢIJĴÑgÂĦĦĴĢIJĴÑâĽĦĬÌã
```

We import the *copy* module to support this:

```
ĨĬĴIJĹĻ ĢIJĴÑ
```

The *copy* module offers two methods that allow us to copy the model — *copy* and *deepcopy*. While *copy* is faster, it only copies the first level of the object's properties, causing copies of our model objects to have some references common to the original model. For example, our model object has a list of layers — the list is the top-level property, and the layers themselves

are secondary — therefore, references to the layer objects will be shared by both the original and copied model objects. Due to these challenges with *copy*, we'll use the *deepcopy* method to recursively traverse all objects and create a full copy.

Next, we'll remove the accumulated loss and accuracy:

```
# RHLHL GQQLILIGLHH LGILHL IL ILLL GLH GQQLLGQN LGIHQLL
ILHHIgILLLgLHNcJGLLaa
ILHHIgGQQLLGQNgLHNcJGLLaa
```

Then remove any data in the input layer, and reset the gradients, if any exist:

```
# RHILLH HGLG ILLLI ILJLL IGNHL
# RHILLH HGLG ILLLI ILJLL IGNHL
# GLH LLGHIHLL ILLLI LIH ILLL LGIHQL
ILHHIgILJLLcIGNHLgccHIQLccgJLJJadLLLJLLd NLJLHa
ILHHIgILLLgccHIQLccgJLJJadHILJLLLd NLJLHa
```

Both ILHHIgILJLLcIGNHL and ILHHIgILLL are class instances. They're attributes of the MLHHI object but also objects themselves. One of the dunder properties (called "dunder" because of the double underscores) that exists for all classes is the ccHIQLcc property. It contains names and values for the class object's properties. We can then use the built-in JLJ method on these values, which means we remove them from that instance of the class' object. The JLJ method will wind up throwing an error if the key we pass as the first parameter doesn't exist, as the JLJ method wants to return the value of the key that it removes. We use the second parameter of the JLJ method — which is the default value that we want to return if the key doesn't exist — to prevent these errors. We will set this parameter to *None* — we do not intend to catch the removed values, and it doesn't really matter what the default value is. This way, we do not have to check if a given property exists, in times like when we'd like to delete it using the *del* statement, and some of them might not exist.

Next, we'll iterate over all the layers to remove their properties:

```
# FLL HGQI IGNHL LHILLH ILJLLd LLLJLL GLH HILJLLL JLLJJHLLIHL
ILL IGNHL ILJ ILHHIgIGNHLLf
    ILL JLLJJHLLN ILJ adILJLLLd dLLLJLLd dHILJLLLd
                    dHNHILLILLd dHGILGLHLdaf
        IGNHLgccHIQLccgJLJJaJLLJJHLLNd NLJLHa
```

With these things cleaned up, we can save the model object. To do that, we have to open a file in a binary-write mode, and call JILGILHgHLLJaa with the model object and the file handler as parameters:

```
# OJHL G LILIH ILJ LIH GILGLNhNLILH ILJHH GLH LGLH LIH ILJHHI
NILI LJHLaJGLId dNGda GL Lf
    JILGILHgHLLJaILJHHId La
```

This makes the full ĽĞĹŦ method:

```
ħ SĞĹŦĹ ĻÎŦ İIJÂŦŤ
ÂŦÌ ĽĞĹŦâĹŦŤÌd ĴĞĻÎãf

    ħ MĞĪŦ Ğ ÂŦŦĴ ĢIJĴÑ IJÌ ĢŁĹĹŦĮĻ İIJÂŦŤ ĨĮĹĻĞĮĢŦ
    İIJÂŦŤ ł ĢIJĴÑgÂŦŦĴĢIJĴÑâĹŦŤÌã

    ħ RŦĹŦĻ ĞĢĢŁİŁŤĞĻŦÂ ĹĞŤŁŦĹ ĨĮ ŤIJĹĹ ĞĮÂ ĞĢĢŁĹĞĢÑ IJĠÏŦĢĻĹ
    İIJÂŦŤgŤIJĹĹgĮŦŃçĴĞĹĹâã
    İIJÂŦŤgĞĢĢŁĹĞĢÑgĮŦŃçĴĞĹĹâã

    ħ RŦİIJĹŦ ÂĞĻĞ ÌĹIJİ ĻÎŦ ĨĮĴŁĻ ŤĞÑŦĹ
    ħ ĞĮÂ ÍĹĞÂĨŦĮĻĹ ÌĹIJİ ĻÎŦ ŤIJĹĹ IJĠÏŦĢĻ
    İIJÂŦŤgĨĮĴŁĻçŤĞÑŦĹgççÂĨĢĻççgĴIJĴâďIJŁĻĴŁĻďd NIJĮŦã
    İIJÂŦŤgŤIJĹĹgççÂĨĢĻççgĴIJĴâďÂĨĮĴŁĻĹďd NIJĮŦã

    ħ FIJĹ ŦĞĢÎ ŤĞÑŦĹ ĹŦİIJĹŦ ĨĮĴŁĻĹd IJŁĻĴŁĻ ĞĮÂ ÂĨĮĴŁĻĹ ĴĹIJĴŦĹĻĨŦĹ
    ÌIJĹ ŤĞÑŦĹ ĨĮ İIJÂŦŤgŤĞÑŦĹĹf
        ÌIJĹ ĴĹIJĴŦĹĻÑ ĨĮ äďĨĮĴŁĻĹďd ďIJŁĻĴŁĻďd ďÂĨĮĴŁĻĹďd
                        ďÂŃŦĨÍÎĻĹďd ďÂĠĨĞĹŦĹďãf
            ŤĞÑŦĹgççÂĨĢĻççgĴIJĴâĴĹIJĴŦĹĻÑd NIJĮŦã

    ħ OĴŦĮ Ğ ÌĨŤŦ ĨĮ ĻÎŦ ĠĨĮĞĹÑhŃĹĨĻŦ İIJÂŦ ĞĮÂ ĹĞĹŦ ĻÎŦ İIJÂŦŤ
    ŃĨĻÎ IJĴŦĮâĴĞĻÎd ďŃĠďã ĞĹ Ìf
        ĴĨĢĪŤŦgÂŁİĴâİIJÂŦŤd Ìã
```

This means we can train a model, then save it whenever we wish with:

```
İIJÂŦŤgĹĞĹŦâďÌĞĹÎĨIJĮçİĮĨĹĻgİIJÂŦŤďã
```

Loading the Model

Loading a model will ideally take place before a model object even exists. What we mean by this is we could load a model by calling a method of the `MIJĤĦĬ` class instead of the object:

```
ĬIJĤĦĬ ł MIJĤĦĬgĬIJĞĤâďÌĞĽÎÎIJIçĬIĮÎĽĻgĬIJĤĦĬďã
```

To achieve this, we're going to use the `đĽĻĞĻĨĢĬĦĻÎIJĤ` decorator. This decorator can be used with class methods to run them on uninitialized objects, where the *`ĽĦĬÌ`* does not exist (notice that it is missing the function definition). In our case, we're going to use it to immediately create a model object without first needing to instantiate a model object. Within this method, we'll open a file using the passed-in path, in binary-read mode, and use pickle to deserialize the saved model:

```
    ħ LIJĞĤĽ ĞIĮĤ ĹĦĻŁĹIĮĽ Ğ ĬIJĤĦĬ
    đĽĻĞĻĨĢĬĦĻÎIJĤ
    ĤĦÌ ĬIJĞĤâĴĞĻÎãf

        ħ OĴĦI ÌĨĬĦ ĨIĮ ĻÎĦ ĠĨIĮĞĹÑhĹĦĞĤ ĬIJĤĦd ĬIJĞĤ Ğ ĬIJĤĦĬ
        ŃĨĻÎ IJĴĦIâĴĞĻÎd ďĹĠďã ĞĽ Ìf
            ĬIJĤĦĬ ł ĴĨĢĬĬĦgĬIJĞĤâÌã

        ħ RĦĻŁĹI Ğ ĬIJĤĦĬ
        ĹĦĻŁĹI ĬIJĤĦĬ
```

Since we already have a saved model, let's create the data, and then load a model to see if it works:

```
ħ CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ ł ĢĹĦĞĻĦçĤĞĻĞçĬIĮĨĽĻâďÌĞĽÎÎIJIçĬIĮĨĽĻçĨĬĞĬĦĽďã

ħ SÎŁÌÌĬĦ ĻÎĦ ĻĹĞĨIĮĨIĮĬ ĤĞĻĞĽĦĻ
ĨĦÑĽ ł IĮĴgĞĹĹĞÑâĹĞIĮĬĦâXgĽÎĞĴĦäflãã
IĮĴgĹĞIĮĤIJĬgĽÎŁÌÌĬĦâĨĦÑĽã
X ł XäĨĦÑĽã
Ñ ł ÑäĨĦÑĽã
```

```
# SQĞĬH ĞĮĤ ĹHĹĬĞĴH ĹĞİĴĬHĹ
X ł âXgĹHĹĬĞĴHâXgĹĬĞĴHäflād hĝãgĞĹĻÑĴHâĮĴgÌĬIJĞĻġğã h ĝğìgĥã w ĝğìgĥ
XçĻHĹĻ ł âXçĻHĹĻgĹHĹĬĞĴHâXçĻHĹĻgĹĬĞĴHäflād hĝãgĞĹĻÑĴHâĮĴgÌĬIJĞĻġğã h
             ĝğìgĥã w ĝğìgĥ

# LIJĞĤ ĻĬH İIJĤHĬ
İIJĤHĬ ł MIJĤHĬgĬIJĞĤâďÌĞĹĬĨIJĮçİĮĨĹĻgİIJĤHĬďã

# EŁĞĬŁĞĻH ĻĬH İIJĤHĬ
İIJĤHĬgHŁĞĬŁĞĻHâXçĻHĹĻd ÑçĻHĹĻã


ńńń
ŁĞĬĨĤĞĻĨIJĮd ĞQQf flgíìǵd ĬIJĹĹf flgġĥǵ
```

Saving the full trained model is a common way of saving a model. It saves parameters (weights and biases) and instances of all the model's objects and the data they generated. That is going to be, for example, the optimizer state like cache, learning rate decay, full model structure, etc. Loading the model, in this case, is as easy as calling one method and the model is ready to use, whether we want to continue training it or use it for a prediction.

Supplementary Material: https://nnfs.io/ch21
Chapter code, further resources, and errata for this chapter.

Chapter 22

Prediction / Inference

While we often spend most of our time focusing on training and testing models, the whole reason we're doing any of this is to have a model that takes new inputs and produces desired outputs. This will typically involve many attempts to train the best model possible, save that model, and load that saved model to do inference, or prediction.

In the case of Fashion MNIST classification, we'd like to load a trained model, show it never-before-seen images, and have it predict the correct classification. To do this, we'll add a new `ĴĹĦÂĨĢĻ` method to the `MIJÂĦĬ` class:

```
ħ PĹĦÂĨĢĻĹ IJĮ ĻÎĦ ĹĞİĴĬĦĹ
ÂĦĬ ĴĹĦÂĨĢĻâĹĦĬĬd Xd ǻd ĠĞĻĢÎçĹĨŅĦłNIJĮĦãf
```

Note that we predict *X* with a possible *ĠĞĻĢÎçĹĨŅĦ*. This means all predictions, including predictions on just one sample, will still be fed in as a list of samples in the form of a NumPy array, whose first dimension is the list samples, and second is sample data. For example, if we would like to predict on a single image, we still need to create a NumPy array mimicking a list containing a single sample — with a shape of *(1, 784)* where *1* is this single sample, and *784* is

the number of features in a sample (pixels per image). Similar to the ĦŁĞĬŁĞĻĦ method, we'll calculate the number of steps we plan to take:

```
# DĦÌĞŁĬĻ ŁĞĬŁĦ ÎÌ ĠĞĻĢÎ ĹÎŅĦ ÎĹ ĮIJĻ ĠĦÎĮÍ ĹĦĻ
ĴĹĦĤÎĢĻÎIJĮçĹĻĦĴĹ ł ĝ

# CĞĬĢŁĬĞĻĦ ĮŁĬĠĦĹ IJÌ ĹĻĦĴĹ
ÎÌ ĠĞĻĢÎçĹÎŅĦ ÎĹ ĮIJĻ NIJĮĦf
    ĴĹĦĤÎĢĻÎIJĮçĹĻĦĴĹ ł ĬĦĮâXã ww ĠĞĻĢÎçĹÎŅĦ
    # DÎŁÎĤÎĮÍ ĹIJŁĮĤĹ ĤIJŃĮg IÌ ĻÎĦĹĦ ĞĹĦ ĹIJĬĦ ĹĦĬĞÎĮÎĮÍ
    # ĤĞĻĞd ĠŁĻ ĮIJĻ Ğ ÌŁĬĬ ĠĞĻĢÎd ĻÎÎĹ ŃIJĮďĻ ÎĮĢĬŁĤĦ ÎĻ
    # AĤĤ ẀĝẀ ĻIJ ÎĮĢĬŁĤĦ ĻÎÎĹ ĮIJĻ ÌŁĬĬ ĠĞĻĢÎ
    ÎÌ ĴĹĦĤÎĢĻÎIJĮçĹĻĦĴĹ ắ ĠĞĻĢÎçĹÎŅĦ l· ĬĦĮâXãf
        ĴĹĦĤÎĢĻÎIJĮçĹĻĦĴĹ κł ĝ
```

Then create a list that we'll populate with the predictions:

```
# MIJĤĦĬ IJŁĻĴŁĻĹ
IJŁĻĴŁĻ ł äā
```

We'll iterate over the batches, passing the samples for predictions forward through the network, and populating the IJŁĻĴŁĻ with the predictions:

```
# IĻĦĹĞĻĦ IJĹĦĹ ĹĻĦĴĹ
ÌIJĹ ĹĻĦĴ ÎĮ ĹĞĮÍĦâĴĹĦĤÎĢĻÎIJĮçĹĻĦĴĹãf

    # IÌ ĠĞĻĢÎ ĹÎŅĦ ÎĹ ĮIJĻ ĹĦĻ h
    # ĻĹĞÎĮ ŁĹÎĮÍ IJĮĦ ĹĻĦĴ ĞĮĤ ÌŁĬĬ ĤĞĻĞĹĦĻ
    ÎÌ ĠĞĻĢÎçĹÎŅĦ ÎĹ NIJĮĦf
        ĠĞĻĢÎçX ł X

    # OĻÎĦĹŃÎĹĦ ĹĬÎĢĦ Ğ ĠĞĻĢÎ
    ĦĬĹĦf
        ĠĞĻĢÎçX ł XäĹĻĦĴắĠĞĻĢÎçĹÎŅĦfâĹĻĦĴκĝāắĠĞĻĢÎçĹÎŅĦā

    # PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹĤ ĴĞĹĹ
    ĠĞĻĢÎçIJŁĻĴŁĻ ł ĹĦĬÌgÌIJĹŃĞĹĤâĠĞĻĢÎçXd ĻĹĞÎĮÎĮÍłFĞĬĹĦã

    # AĴĴĦĮĤ ĠĞĻĢÎ ĴĹĦĤÎĢĻÎIJĮ ĻIJ ĻÎĦ ĬÎĹĻ IJÌ ĴĹĦĤÎĢĻÎIJĮĹ
    IJŁĻĴŁĻgĞĴĴĦĮĤâĠĞĻĢÎçIJŁĻĴŁĻã
```

After running this, the IJŁĻĴŁĻ is a list of batch predictions. Each of them is a NumPy array, a partial result made by predicting on a batch of samples from the input data array. Any applications, or programs, that will make use of the inference output of our models, we expect to simply pass in a list of samples and get back a list of predictions (both in the form of a NumPy array as mentioned before). Since we're not focused on training, we're only using batches in prediction

to ensure our model can fit into memory, but we're going to get a return that's also in batches of predictions. We can see a simple example of this:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

IJŁĻĴŁĻ ł äā

Ġ ł ĮĴgĞĹĹĞÑâäägd ğād ägd ǵāāã
IJŁĻĴŁĻgĞĴĴĦĮĤâĠã
Ġ ł ĮĴgĞĹĹĞÑâäähd ħād äìd íāāã
IJŁĻĴŁĻgĞĴĴĦĮĤâĠã
Ġ ł ĮĴgĞĹĹĞÑâäaîd ĝflād äĝĝd ĝğāāã
IJŁĻĴŁĻgĞĴĴĦĮĤâĠã

ĴĹĨĮĻâIJŁĻĴŁĻã

ńńń
äĞĹĹĞÑâäägd ğād
       ägd ǵāāãd ĞĹĹĞÑâäähd ħād
       äìd íāāãd ĞĹĹĞÑâää îd ĝflād
       äĝĝd ĝğāāãā
```

In this example, we see an output with a batch size of 2 and 6 total samples. The output is a list of arrays, with each array housing a batch of predictions. Instead, we want just 1 list of predictions, no more batches. To achieve this, we're going to use NumPy's `ŁĽĻĞĢĪ` method:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

IJŁĻĴŁĻ ł äā

Ġ ł ĮĴgĞĹĹĞÑâäägd ğād ägd ǵāāã
IJŁĻĴŁĻgĞĴĴĦĮĤâĠã
Ġ ł ĮĴgĞĹĹĞÑâäähd ħād äìd íāāã
IJŁĻĴŁĻgĞĴĴĦĮĤâĠã
Ġ ł ĮĴgĞĹĹĞÑâäaîd ĝflād äĝĝd ĝğāāã
IJŁĻĴŁĻgĞĴĴĦĮĤâĠã

IJŁĻĴŁĻ ł ĮĴgŁĽĻĞĢĪâIJŁĻĴŁĻã

ĴĹĨĮĻâIJŁĻĴŁĻã

ńńń
ää ĝ  ğā
 ä ġ  ǵā
 ä ĥ  ħā
 ä ì  íā
 ä î ĝflā
 äĝĝ ĝğāā
```

It takes a list of objects and stacks them, if possible, creating a homologous array. This is a preferable form of the return from the ĴĹĦĤĨĢĻ method when we pass a list of samples. With plain Python, we might just add to the list each step:

```
IJŁĻĴŁĻ ł ääā

Ġ ł äägd ğād äġd ǵāā
IJŁĻĴŁĻ κł Ġ
Ġ ł äähd ħād äìd íāā
IJŁĻĴŁĻ κł Ġ
Ġ ł äaîd ĝflād äĝĝd ĝğāā
IJŁĻĴŁĻ κł Ġ

ĴĹĨĮĻâIJŁĻĴŁĻã


ńńń
äägd ğād äġd ǵād äĥd ħād äìd íād äîd ĝflād äĝĝd ĝğāā
```

We add results to a list and stack them at the end, instead of appending to the NumPy array each batch to avoid a performance penalty. Unlike plain Python, NumPy is written in *C* language and creates data objects in memory differently. That means that there is no easy way of adding data to the existing NumPy array, other than merging two arrays and saving the result as a new array. But this will lead to a performance penalty, since the further in predictions we are, the bigger the resulting array is. The fastest and most optimal way is to append NumPy arrays to a list and stack them vertically at once when we have collected all of the partial results. We'll add the ĮĴgŁĹĻĞĢĪ to the end of the outputs that we return:

```
ń SĻĞĢĪ ĞĮĤ ĹĦĻŁĹĮ ĹĦĹŁĬĻĹ
ĹĦĻŁĹĮ ĮĴgŁĹĻĞĢĪâIJŁĻĴŁĻã
```

Making our full ĴĹĦĤĨĢĻ method:

```
ƒ PĹĦĤĨĢĻĽ IJĮ ĻĨĦ ĽĞİĴĬĦĽ
ĤĦÌ ĴĹĦĤĨĢĻâĹĦĬÌd Xd ǻd ĠĞĻĢĨçĹĨŅĦłNIJĮĦãf

    ƒ DĦÌĞŁĬĻ ŁĞĬŁĦ ĨÌ ĠĞĻĢĨ ĽĨŅĦ ĨĽ ĮIJĻ ĠĦĨĮĺ ĽĦĻ
    ĴĹĦĤĨĢĻĨIJĮçĽĻĦĴĽ ł ĝ

    ƒ CĞĬĢŁĬĞĻĦ ĮŁĬĠĦĹ IJÌ ĽĻĦĴĽ
    ĨÌ ĠĞĻĢĨçĽĨŅĦ ĨĽ ĮIJĻ NIJĮĦf
        ĴĹĦĤĨĢĻĨIJĮçĽĻĦĴĽ ł ĬĦĮâXã ww ĠĞĻĢĨçĽĨŅĦ
        ƒ DĨŁĨĤĨĮĺ ĹIJŁĮĤĽ ĤIJŃĮg IÌ ĻĨĦĹĦ ĞĹĦ ĽIJĬĦ ĹĦĬĞĨĮĨĮĺ
        ƒ ĤĞĻĞd ĠŁĻ ĮIJĻ Ğ ÌŁĬĬ ĠĞĻĢĨd ĻĨĨĽ ŃIJĮďĻ ĨĮĢĬŁĤĦ ĨĻ
        ƒ AĤĤ ẀĝẀ ĻIJ ĨĮĢĬŁĤĦ ĻĨĨĽ IJĮĻ ÌŁĬĬ ĠĞĻĢĨ
        ĨÌ ĴĹĦĤĨĢĻĨIJĮçĽĻĦĴĽ ǻ ĠĞĻĢĨçĽĨŅĦ l· ĬĦĮâXãf
            ĴĹĦĤĨĢĻĨIJĮçĽĻĦĴĽ κł ĝ

    ƒ MIJĤĦĬ IJŁĻĴŁĻĽ
    IJŁĻĴŁĻ ł äã

    ƒ IĻĦĹĞĻĦ IJŁĦĹ ĽĻĦĴĽ
    ÌIJĹ ĽĻĦĴ ĨĮ ĹĞĮĺĦâĴĹĦĤĨĢĻĨIJĮçĽĻĦĴĽãf

        ƒ IÌ ĠĞĻĢĨ ĽĨŅĦ ĨĽ ĮIJĻ ĽĦĻ h
        ƒ ĻĹĞĨĮ ŁĽĨĮĺ IJĮĦ ĽĻĦĴ ĞĮĤ ÌŁĬĬ ĤĞĻĞĽĦĻ
        ĨÌ ĠĞĻĢĨçĽĨŅĦ ĨĽ NIJĮĦf
            ĠĞĻĢĨçX ł X

        ƒ OĻĨĦĹŃĨĽĦ ĽĬĨĢĦ Ğ ĠĞĻĢĨ
        ĦĬĽĦf
            ĠĞĻĢĨçX ł XäĽĻĦĴǻĠĞĻĢĨçĽĨŅĦfâĽĻĦĴκĝãǻĠĞĻĢĨçĽĨŅĦã

        ƒ PĦĹÌIJĹĬ ĻĨĦ ÌIJĹŃĞĹĤ ĴĞĽĽ
        ĠĞĻĢĨçIJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹĤâĠĞĻĢĨçXd ĻĹĞĨĮĨĮĺłFĞĬĽĦã

        ƒ AĴĴĦĮĤ ĠĞĻĢĨ ĴĹĦĤĨĢĻĨIJĮ ĻIJ ĻĨĦ ĬĨĽĻ IJÌ ĴĹĦĤĨĢĻĨIJĮĽ
        IJŁĻĴŁĻgĞĴĴĦĮĤâĠĞĻĢĨçIJŁĻĴŁĻã

    ƒ SĻĞĢĬ ĞĮĤ ĹĦĻŁĹĮ ĹĦĽŁĬĻĽ
    ĹĦĻŁĹĮ ĮĴgL·ĽĻĞĢĬâIJŁĻĴŁĻã
```

Now we can load the model and test the prediction functionality:

```
ƒ CĹĦĞĻĦ ĤĞĻĞĽĦĻ
Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ ł ĢĹĦĞĻĦçĤĞĻĞçĮĮĨĬĽĻâďÌĞĽĨĨIJĮçĮĮĨĬĽĻçĨĮĞĹĦĽďã

ƒ SĢĞĬĦ ĞĮĤ ĹĦĽĨĞĴĦ ĽĞĮĴĬĦĽ
XçĻĦĽĻ ł âXçĻĦĽĻgĹĦĽĨĞĴĦâXçĻĦĽĻgĽĨĞĴĦäflãd hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġĝã h
            ĝĝìgĥã w ĝĝìgĥ
```

```
# LIJĞĤ ĻÎĦ İIJÂĦĬ
İIJÂĦĬ ł MIJÂĦĬgĬIJĞĤâďÌĞĽÎÎĨIJIçİIĮĨĽĻgİIJÂĦĬďã

# PĹĦÂĨĢĻ IJI ĻÎĦ ÌĨĹĽĻ ĥ ĽĞİĴĬĦĽ ÌĹIJİ ĽĞĬĨÂĞĻĨIJI ÂĞĻĞĽĦĻ
# ĞIĮĤ ĴĹĨIĮĻ ĻÎĦ ĹĦĽŁĬĻ
ĢIJIÌĨÂĦIĢĦĽ ł İIJÂĦĬgĴĹĦÂĨĢĻâXçĻĦĽĻäfĥãã
ĴĹĨIĮĻâĢIJIÌĨÂĦIĢĦĽã


ńńń
ääîgħíğħíĝflĦhflĝ íggġgflĤħíĦhflĤ ĝgflìîġgíħĦhflġ ĝggħġġgflĤĦhflġ ìgħìflġĝĝìĦhflì
  ĥgĤîħġĤĥġĦhflí ğgîĝîìĝĤħĦhflğ ígħħħĝğgíĦhĝħ ħgíĝġĝĝíğĦhflħ ĝgíflĤħġîħĦhĝğā
 äìgìğîġìğġĦhflĝ ĝgflħĝġìííĦhflġ îggġĤĝîíĝĦhflġ îgflħġìĝĤġĦhflğ ġgġííîġġĤĦhflġ
  ğgflĤħĤħġîĦhflì ĝggġflĝíĤġĦhflĝ ħggflîĤíîħĦhĝğ ĤğğflġĤîííĦhflĤ ìgìíġflflġíĦhĝĝā
 äîgġġĝflíğflĦhflĝ ĥgĝíġĝĝħĝĦhflĤ ĝgġìğġĤĝíĦhflġ íggflħíğġĝĦhflġ ìgîìĤĝğĝħĦhflħ
  îgîħĝîflflĝĦhflì ĥgġĤġğíííĦhflğ ĝgîħğğġĝĝĦhĝġ ĝgġîîìíġìĦhflĤ ğgğîħġġîîĦhĝflā
 äîgííġflìğĝĦhflĝ ĝgğĤìĤìġîĦhflġ ğgĤìġíĤííĦhflġ ĝgġġğġìĝĝĦhflġ ğgĤĝĝġíġħĦhflħ
  ĥgħĝíġġìħĦhflì ĝgflĝĤħîğĝĦhflğ ğgíĤîĝflìíĦhĝġ ĥgĤĝħğflĝíĦhflħ ĝgġìġħíġflĦhĝflā
 äîgğíħîġħìĦhflĝ ìggìĝġîìíĦhflġ ĝgìĤìîìííĦhflġ ğgĝíħġìġîĦhflġ ĝgìîġĤĝğîĦhflĤ
  ĝgîğíğîflíĦhflĤ ħgħĤğĝĝĝĦhflğ ĥggĤġġĤġíĦhĝĝ ħgĤĝĤìĤħíĦhflĤ ìgğflğflğĝĝĦhflîāā
```

It looks like it's working! After spending so much time training and finding the best hyperparameters, a common issue people have is actually *using* the model. As a reminder, each of the subarrays in the output is a vector of confidences containing a confidence metric per class. The first thing that we need to do in this case is to gather the argmax values of these confidence vectors. Recall that we're using a softmax classifier, so this neural network is attempting to fit to one-hot vectors, where the correct class is represented by a 1, and the others by 0s. When doing inference, it is unlikely to achieve such a perfect result, but the index associated with the highest value in the output is what we determine the model is predicting; we're just using the argmax. We could write code to do this, but we've already done that in all of the activation function classes, where we added a `ĴĹĦÂĨĢĻĨIJIĻ` method:

```
# SIJÌĻİĞŇ ĞĢĻĨĽĞĻĨIJI
ĢĬĞĽĽ AĢĻĨĽĞĻĨIJIçSIJÌĻİĞŇf
    ggg
    # CĞĬĢŁĬĞĻĦ ĴĹĦÂĨĢĻĨIJIĻ ÌIJĹ IJŁĻĴŁĻĽ
    ÂĦÌ ĴĹĦÂĨĢĻĨIJIĻĽâĽĦĬÌd IJŁĻĴŁĻĽãf
        ĹĦĻŁĹĮ ĮĴgĞĹÍİĞŇâIJŁĻĴŁĻĽd ĞŇĨĽŁĝã
```

We've also set an attribute in our model with the output layer's activation function, which means we can generically acquire predictions by performing:

```
ń LIJĞĤ ĻÎĦ İIJĤĦĬ
İIJĤĦĬ ł MIJĤĦĬgĬIJĞĤâďÌĞĽÎÎIJIçİIĮÎĽĻgİIJĤĦĬďã

ń PĹĦĤÎĢĻ IJI ĻÎĦ ÌÎĹĽĻ ĥ ĽĞİĴĬĦĽ ÌĹIJİ ŁĞĬÎĤĞĻÎIJI ĤĞĻĞĽĦĻ ĞIĤ ĴĹÎIĮĻ ĻÎĦ
ĹĦĽŁĬĻ
ĢIJIÌÎĤĦIĢĦĽ ł İIJĤĦĬgĴĹĦĤÎĢĻâXçĻĦĽĻäfĥāã
ĴĹĦĤÎĢĻÎIJIĽ ł İIJĤĦĬgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻÎŁĞĻÎIJIgĴĹĦĤÎĢĻÎIJIĽâĢIJIÌÎĤĦIĢĦĽã
ĴĹÎIĮĻâĴĹĦĤÎĢĻÎIJIĽã

ń PĹÎIĮĻ ÌÎĹĽĻ ĥ ĬĞĠĦĬĽ
ĴĹÎIĮĻâÑçĻĦĽĻäfĥāã

ńńń
äfl fl fl fl flā
äfl fl fl fl flā
```

In this case, our model predicted all “class 0,” and our test labels were all class 0 as well. Since shuffling our testing data isn’t essential, we never shuffled them, so they’re going in the original order like our training data was. This explains why all these predictions are 0s.

In practice, we don’t care what class number something is, we want to know *what* it is. In this case, class numbers map directly to names, so we add the following dictionary to our code:

```
ÌĞĽÎÎIJIçİIĮÎĽĻçĬĞĠĦĬĽ ł ă
    flf dďThĽÎÎĹĻwĻIJĴďd
    ĝf dďTĹIJŁĽĦĹďd
    ğf dďPŁĬĬIJŁĦĹďd
    ġf dďDĹĦĽĽďd
    ǵf dďCIJĞĻďd
    ĥf dďSĞIĤĞĬďd
    ħf dďSÎÎĹĻďd
    ìf dďSIĦĞĪĦĹďd
    íf dďBĞÍďd
    îf dďAIĪĬĦ ĠIJIJĻď
å
```

Then we could get the string classification by performing:

```
ÌIJĹ ĴĹĦĤÎĢĻÎIJI ÎIĮ ĴĹĦĤÎĢĻÎIJIĮĽf
    ĴĹÎIĮĻâÌĞĽÎÎIJIçİIĮÎĽĻçĬĞĠĦĬĽäĴĹĦĤÎĢĻÎIJIĮāã

ńńń
ThĽÎÎĹĻwĻIJĴ
ThĽÎÎĹĻwĻIJĴ
ThĽÎÎĹĻwĻIJĴ
ThĽÎÎĹĻwĻIJĴ
ThĽÎÎĹĻwĻIJĴ
```

This is great, but we still have to *actually* predict something instead of the training data. When covering deep learning, the training steps often get all the focus; we want to see those accuracy and loss metrics look good! It works well to focus on training for tutorials that aim to show people how to *use* a framework, but one of the larger pain points we see is *applying* the models in production, or just running predictions on new data that was sourced from the wild (especially since outside data is rarely formatted to match your training data).

At the moment, we have a model trained on items of clothing, so we need some truly new samples. Luckily, you're probably a person who owns some clothes; if so, you can take photos of those to start with. If not, use the following sample photos:

https://nnfs.io/datasets/tshirt.png

Fig 20.01: Hand-made t-shirt image for the purpose of inference.

https://nnfs.io/datasets/pants.png

Fig 20.02: Hand-made pants image for the purpose of inference.

You can also try your hand at hand-drawing samples like these. Once you have new images/samples that you wish to use in production, you'll need to preprocess them in the same way the training samples were. Some of these changes are fairly difficult to forget, like the image resolution or number of color channels; we'd get an error if we didn't do those things. Let's start preprocessing our image by loading it in. We'll use the *cv2* package to read in the image:

```
ĨİĴIJĹĻ ĢĿğ

ĨİĞÍĦçĤĞĻĞ ł ĢĿğgĨİĹĦĞĤâďĻĽÎĨĹĻgĴĮÍď ĢĿğgIMREADçUNCHANGEDã
```

We can view the image:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĴĬĻgĨİĽÎIJŃâĢĿğgĢĿĻCIJĬIJĹâĨİĞÍĦçĤĞĻĞd ĢĿğgCOLORçBGRğRGBãã
ĴĬĻgĽÎIJŃâã
```

Fig 20.03: Hand-made t-shirt image loaded with Python.

Note that we're doing *cv2.cvtColor* because OpenCV uses BGR (blue, green, red pixel values) color format by default, but matplotlib uses RGB (red, green, blue), so we're converting the colormap to display the image.

The first thing we'll do is read this image as grayscale instead of RGB. This is in contrast to the Fashion MNIST images, which are grayscaled, and we have used `ĢĿğgIMREADçUNCHANGED` as a parameter to the `ĢĿğgĨİĹĦĞĤâã` to inform OpenCV that our intention is to read images grayscaled and unchanged. Here, we have a color image, and this parameter won't work as "unchanged" means containing all the colors; thus, we'll use `ĢĿğgIMREADçGRAYSCALE` to force grayscaling when we read in our image:

```
ĨİĴIJĹĻ ĢĿğ
ĨİĞÍĦçĤĞĻĞ ł ĢĿğgĨİĹĦĞĤâďĻĽÎĨĹĻgĴĮÍďd ĢĿğgIMREADçGRAYSCALEã
```

Then we can display it:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨġgĴÑĴĬIJĻ ĞĽ ĴĬĻ
ĴĬĻgĨİĽÎIJŃâĨİĞÍĦçĤĞĻĞd ĢİĞĴłďÍĹĞÑďã
ĴĬĻgĽÎIJŃâã
```

Note that we use a gray colormap with `ĴĬĻgĨİĽÎIJŃâã` by passing the `ďÍĹĞÑď` argument into the `ĢİĞĴ` parameter. The result is a grayscale image:

Fig 20.04: Grayscaled hand-made t-shirt image loaded with Python.

Next, we'll resize the image to be the same 28x28 resolution as our training data:

```
ĨİĞÍĦçĤĞĻĞ ł ĢĿğgĹĦĽĨŅĦâĨİĞÍĦçĤĞĻĞd âğíd ğíãã
```

We then display this resized image:

```
ĴĬĻgĨİĽÎIJŃâĨİĞÍĦçĤĞĻĞd ĢİĞĴłďÍĹĞÑďã
ĴĬĻgĽÎIJŃâã
```

Fig 20.05: Grayscaled and scaled down hand-made t-shirt image.

Next, we'll flatten and scale the image. While the scale operation is the same as for the training data, the flattening is a bit different; we don't have a list of images but a single image, and, as previously explained, a single image must be passed in as a list containing this single image. We flatten by applying gĹĦĹÎĞĴĦâĝd hĝã to the image. The ĝ argument represents the number of samples, and the hĝ flattens the image to a vector of length 784. This produces a 1x784 array with our one sample and 784 features (i.e., 28x28 pixels):

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ

ĨİĞÍĦçĤĞĻĞ ł âĨİĞÍĦçĤĞĻĞgĹĦĹÎĞĴĦâĝd hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġĝã h
                  ĝĝìghĥã w ĝĝìghĥ
```

Now we can load in our model and predict on this image data:

```
ń LIJĞĤ ĻÎĦ ÌIJĤĦĬ
ÌIJĤĦĬ ł MIJĤĦĬgĬIJĞĤâďÌĞĽÎĨIJĮçÌĮĨĽĻgÌIJĤĦĬďã

ń PĹĦĤĨĢĻ IJĮ ĻÎĦ ĨİĞÍĦ
ĢIJĮÌĨĤĦĮĢĦĽ ł ÌIJĤĦĬgĴĹĦĤĨĢĻâĨİĞÍĦçĤĞĻĞã

ń GĦĻ ĴĹĦĤĨĢĻĨIJĮ ĨĮĽĻĦĞĤ IJÌ ĢIJĮÌĨĤĦĮĢĦ ĬĦĽĦĬĽ
ĴĹĦĤĨĢĻĨIJĮĽ ł ÌIJĤĦĬgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĽĞĻĨIJĮgĴĹĦĤĨĢĻĨIJĮĽâĢIJĮÌĨĤĦĮĢĦĽã

ń GĦĻ ĬĞĠĦĬ ĮĞİĦ ÌĹIJİ ĬĞĠĦĬ ĨĮĤĦÑ
ĴĹĦĤĨĢĻĨIJĮ ł ÌĞĽÎĨIJĮçÌĮĨĽĻçĬĞĠĦĬĽäĴĹĦĤĨĢĻĨIJĮĽäflāā

ĴĹĨĮĻâĴĹĦĤĨĢĻĨIJĮã
```

Making our code up to this point that loads, preprocesses, and predicts:

```
ń LĞĠĦĬ ĨĮĤĦÑ ĻIJ ĬĞĠĦĬ ĮĞİĦ ĹĦĬĞĻĨIJĮ
ÌĞĽÎĨIJĮçÌĮĨĽĻçĬĞĠĦĬĽ ł ă
    flf dThĹÎĨĹĻwĻIJĴďd
    ĝf dTĹIJŁĽĦĹďd
    ğf dPŁĬĬIJŁĦĹďd
    ġf dDĹĦĽĽďd
    ģf dCIJĞĻďd
    ĥf dSĞĮĤĞĬďd
    ħf dSÎĨĹĻďd
    ìf dSĮĦĞĨĦĹďd
    íf dBĞÍďd
    îf dAĮĨĬĦ ĠIJIJĻď
å

ń RĦĞĤ ĞĮ ĨİĞÍĦ
ĨİĞÍĦçĤĞĻĞ ł ĢŁğgĨİĹĦĞĤâďĻĽÎĨĹĻgĴĮÍďd ĢŁğgIMREADçGRAYSCALEã
```

```
# RHĹĨŅH ĻIJ ĻÎH ĹĞİH ĹĨŅH ĞĹ FĞĹÎĨIJĮ MNIST ĨİĞÍHĹ
ĨİĞÍHçĤĞĻĞ ł ĢŁğgĹHĹĨŅHâĨİĞÍHçĤĞĻĞd âğíd ğíãã

# RHĹÎĞĴH ĞĮĤ ĹĢĞĬH ĴĨŇHĬ ĤĞĻĞ
ĨİĞÍHçĤĞĻĞ ł âĨİĞÍHçĤĞĻĞgĹHĹÎĞĴHâĝd hĝãgĞĹĻÑĴHâĮĴgÌĬIJĞĻġğã h
                ĝğìgĥã w ĝğìgĥ

# LIJĞĤ ĻÎH İIJĤHĬ
İIJĤHĬ ł MIJĤHĬgĬIJĞĤâdÌĞĹÎĨIJĮçİĮĨĹĻgİIJĤHĬdã

# PĹHĤĨĢĻ IJĮ ĻÎH ĨİĞÍH
ĴĹHĤĨĢĻĨIJĮĹ ł İIJĤHĬgĴĹHĤĨĢĻâĨİĞÍHçĤĞĻĞã

# GHĻ ĴĹHĤĨĢĻĨIJĮ ĨĮĹĻHĞĤ IJÌ ĢIJĮÌĨĤHĮĢH ĬHŁHĬĹ
ĴĹHĤĨĢĻĨIJĮĹ ł İIJĤHĬgIJŁĻĴŁĻçĬĞÑHĹçĞĢĻĨŁĞĻĨIJĮgĴĹHĤĨĢĻĨIJĮĹâĴĹHĤĨĢĻĨIJĮĹã

# GHĻ ĬĞĠHĬ ĮĞİH ÌĹIJİ ĬĞĠHĬ ĨĮĤHÑ
ĴĹHĤĨĢĻĨIJĮ ł ÌĞĹÎĨIJĮçİĮĨĹĻçĬĞĠHĬĹäĴĹHĤĨĢĻĨIJĮĹäflāā

ĴĹĨĮĻâĴĹHĤĨĢĻĨIJĮã
```

Note that we are using `ĴĹHĤĨĢĻĨIJĮĹäflā` as we passed in a single image in the form of a list, and the model returns a list containing a single prediction.

Only one problem…

```
ńńń
AĮĨĬH ĠIJIJĻ
```

What's wrong? Let's compare our currently-preprocessed image to the training data:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĹ ĴĬĻ

İĮĨĹĻçĨİĞÍH ł ĢŁğgĨİĹHĞĤâdÌĞĹÎĨIJĮçİĮĨĹĻçĨİĞÍHĹwĻĹĞĨĮwflwflflflgĴĮÍdd
                            ĢŁğgIMREADçUNCHANGEDã
ĴĬĻgĨİĹÎIJŃâİĮĨĹĻçĨİĞÍHd ĢİĞĴłdÍĹĞÑdã
ĴĬĻgĹÎIJŃâã
```

Fig 20.06: Example t-shirt image from the Fashion MNIST dataset

Now we compare this original and example training image to our's:

Fig 20.07: Grayscaled and scaled down hand-made t-shirt image.

The training data that we've used is color-inverted (i.e., the background is black instead of white, and so on). To invert our image before scaling, we can use pixel math directly instead of using OpenCV. We'll subtract all the pixel values from the maximum pixel value: 255. For example, a value of *0* will become *255 - 0 = 255*, and the value of *255* will become *255 - 255 = 0*.

```
ĨĬĞÍĦçĤĞĻĞ ł ğĥĥ h ĨĬĞÍĦçĤĞĻĞ
```

With this small change, our prediction code becomes:

```
ń ŘĦĞĤ ĞĮ ĨĬĞÍĦ
ĨĬĞÍĦçĤĞĻĞ ł ĢĽğgĨĬĹĦĞĤâďĻĽĨĨĹĻgĴĮÍďd ĢĽğgIMREADçGRAYSCALEã

ń ŘĦĽĨŅĦ ĻIJ ĻĨĦ ĽĞĬĦ ĽĨŅĦ ĞĽ FĞĽĨĨIJĮ MNIST ĨĬĞÍĦĽ
ĨĬĞÍĦçĤĞĻĞ ł ĢĽğgĹĦĽĨŅĦâĨĬĞÍĦçĤĞĻĞd âğíd ğíãã

ń IĮĽĦĹĻ ĨĬĞÍĦ ĢIJĬIJĹĽ
ĨĬĞÍĦçĤĞĻĞ ł ğĥĥ h ĨĬĞÍĦçĤĞĻĞ
```

```
# RHĽÎĞĴH ĞĮÂ ĽĢĞĬH ĴĨŇHĬ ÂĞĻĞ
ĨİĞÍHçÂĞĻĞ ł âĨİĞÍHçÂĞĻĞgĹHĽÎĞĴHâĝd hĝãgĞĽĻÑĴHâĮĴgÌĬIJĞĻġĝã h
                  ĝĝìghĥã w ĝĝìghĥ

# LIJĞÂ ĻÎH İIJÂHĬ
İIJÂHĬ ł MIJÂHĬgĬIJĞÂâďÌĞĽÎĨIJĮçİĮĨĽĻgİIJÂHĬďã

# PĹHÂĨĢĻ IJĮ ĻÎH ĨİĞÍH
ĢIJĮÌĨÂHĮĢHĽ ł İIJÂHĬgĴĹHÂĨĢĻâĨİĞÍHçÂĞĻĞã

# GHĻ ĴĹHÂĨĢĻĨIJĮ ĨĮĽĻHĞÂ IJÌ ĢIJĮÌĨÂHĮĢH ĬHL·HĬĽ
ĴĹHÂĨĢĻĨIJĮĽ ł İIJÂHĬgIJŁĻĴŁĻçĬĞÑHĹçĞĢĻĨL·ĞĻĨIJĮgĴĹHÂĨĢĻĨIJĮĽâĢIJĮÌĨÂHĮĢHĽã

# GHĻ ĬĞĠHĬ ĮĞİH ÌĹIJİ ĬĞĠHĬ ĨĮÂHH
ĴĹHÂĨĢĻĨIJĮ ł ÌĞĽÎĨIJĮçİĮĨĽĻçĬĞĠHĬĽäĴĹHÂĨĢĻĨIJĮĽäflāā

ĴĹĨĮĻâĴĹHÂĨĢĻĨIJĮã


ńńń
ThĽÎĨĹĻwĻIJĴ
```

Now it works! The reason it works now, and not work previously, is from how the *Dense* layers work — they learn feature (pixel in this case) values and the correlation between them. Contrast this with convolutional layers, which are being trained to find and understand features on images (not features as data input nodes, but actual characteristics/traits, such as lines and curves). Because pixel values were very different, the model incorrectly put its "guess" in this case. Convolutional layers may properly predict in this case, as-is.

Let's try the pants:

```
ĨİĴIJĹĻ İĞĻĴĬIJĻĬĨĠgĴÑĴĬIJĻ ĞĽ ĴĬĻ

ĨİĞÍHçÂĞĻĞ ł ĢL·ğgĨİĹHĞÂâďĴĞĮĻĽgĴĮÍďd ĢL·ğgIMREADçUNCHANGEDã
ĴĬĻgĨİĽÎIJŃâĢL·ğgĢL·ĻCIJĬIJĹâĨİĞÍHçÂĞĻĞd ĢL·ğgCOLORçBGRğRGBãã
ĴĬĻgĽÎIJŃâã
```

Fig 20.08: Hand-made pants image loaded with Python.

Now we'll preprocess:

```
# RHĞĤ ĞĮ ĨİĞÍH
ĨİĞÍHçĤĞĻĞ ł ĢĽğgĨİĹHĞĤâďĴĞĮĻĹgĴĮÍďd ĢĽğgIMREADçGRAYSCALEã

# RHĹĨŅH ĻIJ ĻĨH ĹĞİH ĹĨŅH ĞĹ FĞĹĨĨIJĮ MNIST ĨİĞÍHĹ
ĨİĞÍHçĤĞĻĞ ł ĢĽğgĹHĹĨŅHâĨİĞÍHçĤĞĻĞd âğíd ğíãã

# IĮĽHĹĻ ĨİĞÍH ĢIJĬIJĹĹ
ĨİĞÍHçĤĞĻĞ ł ğĥĥ h ĨİĞÍHçĤĞĻĞ
```

Let's see what we have:

```
ĴĬĮgĨİĹĨIJŃâĨİĞÍHçĤĞĻĞd ĢİĞĴłďÍĹĞÑďã
ĴĬĮgĹĨIJŃâã
```

Fig 20.09: Grayscaled and scaled down hand-made t-shirt image.

Making our code:

```
ƒ LĞĠĦĬ ĨĮĤĦŃ ĻIJ ĬĞĠĦĬ ĮĞİĦ ĹĦĬĞĻĨIJĮ
ÌĞĽÎĨIJĮçİĮĨĽĻçĬĞĠĦĬĽ ł ă
    flf ďThĽÎĨĹĻwĻIJĴďd
    ĝf ďTĹIJŁĽĦĹďd
    ğf ďPŁĬĬIJL·ĦĹďd
    ġf ďDĹĦĽĽďd
    ģf ďCIJĞĻďd
    ĥf ďSĞĮĤĞĬďd
    ħf ďSÎĨĹĻďd
    ìf ďSĮĦĞĪĦĹďd
    íf ďBĞÍďd
    îf ďAĮĪĬĦ ĠIJIJĻď
å

ƒ RĦĞĤ ĞĮ ĨİĞÍĦ
ĨİĞÍĦçĤĞĻĞ ł ĢL·ğgĨİĹĦĞĤâďĴĞĮĻĽgĴĮÍďd ĢL·ğgIMREADçGRAYSCALEã

ƒ RĦĽĨŅĦ ĻIJ ĻÎĦ ĽĞİĦ ĽĨŅĦ ĞĽ FĞĽÎĨIJĮ MNIST ĨİĞÍĦĽ
ĨİĞÍĦçĤĞĻĞ ł ĢL·ğgĹĦĽĨŅĦâĨİĞÍĦçĤĞĻĞd âğíd ğíãã

ƒ IĮL·ĦĹĻ ĨİĞÍĦ ĢIJĬIJĹĽ
ĨİĞÍĦçĤĞĻĞ ł ğĥĤ h ĨİĞÍĦçĤĞĻĞ

ƒ RĦĽÎĞĴĦ ĞĮĤ ĽĢĞĬĦ ĴĨŇĦĬ ĤĞĻĞ
ĨİĞÍĦçĤĞĻĞ ł âĨİĞÍĦçĤĞĻĞgĹĦĽÎĞĴĦâĝd hĝãgĞĽĻÑĴĦâĮĴgÌĬIJĞĻġğã h
                  ĝğìgĥã w ĝğìgĥ

ƒ LIJĞĤ ĻÎĦ İIJĤĦĬ
İIJĤĦĬ ł MIJĤĦĬgĬIJĞĤâďÌĞĽÎĨIJĮçİĮĨĽĻgİIJĤĦĬďã

ƒ PĹĦĤĨĢĻ IJĮ ĻÎĦ ĨİĞÍĦ
ĢIJĮÌĨĤĦĮĢĦĽ ł İIJĤĦĬgĴĹĦĤĨĢĻâĨİĞÍĦçĤĞĻĞã

ƒ GĦĻ ĴĹĦĤĨĢĻĨIJĮ ĨĮĽĻĦĞĤ IJÌ GIJĮÌĨĤĦĮĢĦ ĬĦL·ĦĬĽ
ĴĹĦĤĨĢĻĨIJĮĽ ł İIJĤĦĬgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨL·ĞĻĨIJĮgĴĹĦĤĨĢĻĨIJĮĽâĢIJĮÌĨĤĦĮĢĦĽã

ƒ GĦĻ ĬĞĠĦĬ ĮĞİĦ ÌĹIJİ ĬĞĠĦĬ ĨĮĤĦŃ
ĴĹĦĤĨĢĻĨIJĮ ł ÌĞĽÎĨIJĮçİĮĨĽĻçĬĞĠĦĬĽäĴĹĦĤĨĢĻĨIJĮĽäflāā

ĴĹĨĮĻâĴĹĦĤĨĢĻĨIJĮã

ńńń
TĹIJŁĽĦĹ
```

A success again! We have now coded in the last feature of our model, which closes the list of the topics that we covered in this book.

Full code:

```
ĨİĴIJĹĻ ĮŁİĴÑ ĞĽ ĮĴ
ĨİĴIJĹĻ ĮĮÌĽ
ĨİĴIJĹĻ IJĽ
ĨİĴIJĹĻ ĢŁğ
ĨİĴIJĹĻ ĴĨĢĪĬĦ
ĨİĴIJĹĻ ĢIJĴÑ

ĮĮÌĽgĨĮĨĻâã


ńĦĮĽĦ ĬĞÑĦĹ
ĢĬĞĽĽ LĞÑĦĹçDĦĮĽĦf

    ńLĞÑĦĹ ĨĮĨĻĨĞĬĨÑĢĞĻĨIJĮ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽd
                ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨÑĦĹçĬĝłfld ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨÑĦĹçĬğłfld
                ĠĨĞĽçĹĦÍŁĬĞĹĨÑĦĹçĬĝłfld ĠĨĞĽçĹĦÍŁĬĞĹĨÑĦĹçĬğłflãf
        ńIĮĨĻĨĞĬĨÑĦ ŃĦĨÍÎĻĽ ĞĮÂ ĠĨĞĽĦĽ
        ĽĦĬÌgŃĦĨÍÎĻĽ ł flgflĝ ǻ ĮĴgĹĞĮÂIJĮgĹĞĮÂĮâĮçĨĮĴŁĻĽd ĮçĮĦŁĹIJĮĽã
        ĽĦĬÌgĠĨĞĽĦĽ ł ĮĴgÑĦĹIJĽââĝd ĮçĮĦŁĹIJĮĽãã
        ńSĦĻ ĹĦÍŁĬĞĹĨÑĢĞĻĨIJĮ ĽĻĹĦĮÍĻÎ
        ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨÑĦĹçĬĝ ł ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨÑĦĹçĬĝ
        ĽĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨÑĦĹçĬğ ł ŃĦĨÍÎĻçĹĦÍŁĬĞĹĨÑĦĹçĬğ
        ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨÑĦĹçĬĝ ł ĠĨĞĽçĹĦÍŁĬĞĹĨÑĦĹçĬĝ
        ĽĦĬÌgĠĨĞĽçĹĦÍŁĬĞĹĨÑĦĹçĬğ ł ĠĨĞĽçĹĦÍŁĬĞĹĨÑĦĹçĬğ

    ńFIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ńRĦİĦİĠĦĹ ĨĮĴŁĻ ŁĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ńCĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ŁĞĬŁĦĽ ÌĹIJİ ĨĮĴŁĻĽd ŃĦĨÍÎĻĽ ĞĮÂ ĠĨĞĽĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgÂIJĻâĨĮĴŁĻĽd ĽĦĬÌgŃĦĨÍÎĻĽã κ ĽĦĬÌgĠĨĞĽĦĽ

    ńBĞĢĪŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ĠĞĢĪŃĞĹÂâĽĦĬÌd ÂŁĞĬŁĦĽãf
        ńGĹĞÂĨĦĮĻĽ IJĮ ĴĞĹĞĬĦĻĦĹĽ
        ĽĦĬÌgÂŃĦĨÍÎĻĽ ł ĮĴgÂIJĻâĽĦĬÌgĨĮĴŁĻĽgTd ÂŁĞĬŁĦĽã
        ĽĦĬÌgÂĠĨĞĽĦĽ ł ĮĴgĽŁĬâÂŁĞĬŁĦĽd ĞÑĨĽłfld ĬĦĦĴÂĨĬĽłTĹŁĦã
```

```
        ńı GĹĞĤĨĦĮĻĹ ĲĮ ĹĦÍŁĬĞĹĨŅĞĻĨĲĮ
        ńı Lĝ ĲĮ ŃĦĨÍÎĻĹ
        ĨÌ ĹĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ĤLĝ ł ĮĴgĲĮĦĹçĬĨĨĦâĹĦĬÌgŃĦĨÍÎĻĹã
            ĤLĝäĹĦĬÌgŃĦĨÍÎĻĹ l· flā ł hĝ
            ĹĦĬÌgĤŃĦĨÍÎĻĹ κł ĹĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń́ ĤLĝ
        ńı Lğ ĲĮ ŃĦĨÍÎĻĹ
        ĨÌ ĹĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĹĦĬÌgĤŃĦĨÍÎĻĹ κł ğ ń́ ĹĦĬÌgŃĦĨÍÎĻçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń́ ċ
                                ĹĦĬÌgŃĦĨÍÎĻĹ
        ńı Lĝ ĲĮ ĠĨĞĹĦĹ
        ĨÌ ĹĦĬÌgĠĨĞĹçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń flf
            ĤLĝ ł ĮĴgĲĮĦĹçĬĨĨĦâĹĦĬÌgĠĨĞĹĦĹã
            ĤLĝäĹĦĬÌgĠĨĞĹĦĹ l· flā ł hĝ
            ĹĦĬÌgĤĠĨĞĹĦĹ κł ĹĦĬÌgĠĨĞĹçĹĦÍŁĬĞĹĨŅĦĹçĬĝ ń́ ĤLĝ
        ńı Lğ ĲĮ ĠĨĞĹĦĹ
        ĨÌ ĹĦĬÌgĠĨĞĹçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń flf
            ĹĦĬÌgĤĠĨĞĹĦĹ κł ğ ń́ ĹĦĬÌgĠĨĞĹçĹĦÍŁĬĞĹĨŅĦĹçĬğ ń́ ċ
                                ĹĦĬÌgĠĨĞĹĦĹ

        ńı GĹĞĤĨĦĮĻ ĲĮ L·ĞĬŁĦĹ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĮĴgĤĲĻâĤL·ĞĬŁĦĹd ĹĦĬÌgŃĦĨÍÎĻĹgTã

    ńı RĦĻĹĨĦL·Ħ ĬĞÑĦĹ ĴĞĹĞĬĦĻĦĹĹ
    ĤĦÌ ÍĦĻçĴĞĹĞĬĦĻĦĹĹâĹĦĬÌãf
        ĹĦĻŁĹĮ ĹĦĬÌgŃĦĨÍÎĻĹd ĹĦĬÌgĠĨĞĹĦĹ

    ńı SĦĻ ŃĦĨÍÎĻĹ ĞĮĤ ĠĨĞĹĦĹ ĨĮ Ğ ĬĞÑĦĹ ĨĮĹĻĞĮĢĦ
    ĤĦÌ ĹĦĻçĴĞĹĞĬĦĻĦĹĹâĹĦĬÌd ŃĦĨÍÎĻĹd ĠĨĞĹĦĹãf
        ĹĦĬÌgŃĦĨÍÎĻĹ ł ŃĦĨÍÎĻĹ
        ĹĦĬÌgĠĨĞĹĦĹ ł ĠĨĞĹĦĹ


ńı DĹĲĴĲŁĻ
ĢĬĞĹĹ LĞÑĦĹçDĹĲĴĲŁĻf

    ńı IĮĨĻ
    ĤĦÌ ççĨĮĨĻççâĹĦĬÌd ĹĞĻĦãf
        ńı SĻĲĹĦ ĹĞĻĦd ŃĦ ĨĮL·ĦĹĻ ĨĻ ĞĹ ÌĲĹ ĦŃĞĬĴĬĦ ÌĲĹ ĤĹĲĴĲŁĻ
        ńı ĲÌ flgĝ ŃĦ ĮĦĦĤ ĹŁĢĢĦĹĹ ĹĞĻĦ ĲÌ flgî
        ĹĦĬÌgĹĞĻĦ ł ĝ h ĹĞĻĦ

    ńı FĲĹŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ÌĲĹŃĞĹĤâĹĦĬÌd ĨĮĴŁĻĹd ĻĹĞĨĮĨĮĺãf
        ńı SĞL·Ħ ĨĮĴŁĻ L·ĞĬŁĦĹ
        ĹĦĬÌgĨĮĴŁĻĹ ł ĨĮĴŁĻĹ
```

```
        # IÌ ĮIJĻ ĨĮ ĻÎĦ ĻĹĞĨĮĨĮÍ İIJĤĦ h ĹĦĻŁĹĮ ŁĞĬŁĦĽ
        ĨÌ ĮIJĻ ĻĹĞĨĮĨĮÍf
            ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽgĢIJĴÑâã
            ĹĦĻŁĹĮ

        # GĦĮĦĹĞĻĦ ĞĮÎ ĽĞŁĦ ĽĢĞĬĦÎ İĞĽĨ
        ĽĦĬÌgĠĨĮĞĹÑçİĞĽĨ ł ĮĴgĹĞĮÎIJİgĠĨĮIJİĨĞĬâĝd ĽĦĬÌgĹĞĻĦd
                            ĽĨĢĦłĨĮĴŁĻĽgĽĨĞĴĦã w ĽĦĬÌgĹĞĻĦ
        # AĴĴĬÑ İĞĽĨ ĻIJ IJŁĻĴŁĻ ŁĞĬŁĦĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽ ǻ ĽĦĬÌgĠĨĮĞĹÑçİĞĽĨ

    # BĞĢĨŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ĠĞĢĨŃĞĹÎâĽĦĬÌd ÎŁĞĬŁĦĽãf
        # GĹĞÎĨĦĮĻ IJĮ ŁĞĬŁĦĽ
        ĽĦĬÌgÎĨĮĴŁĻĽ ł ÎŁĞĬŁĦĽ ǻ ĽĦĬÌgĠĨĮĞĹÑçİĞĽĨ


# IĮĴŁĻ ðĬĞÑĦĹð
ĢĬĞĽĽ ĽÑĦĹçIĮĴŁĻf

    # FIJĹŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ÌIJĹŃĞĹÎâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽ


# RĦLU ĞĢĻĨŁĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨŁĞĻĨIJĮçRĦLUf

    # FIJĹŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ÌIJĹŃĞĹÎâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        # RĦİĦİĠĦĹ ĨĮĴŁĻ ŁĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        # CĞĬĢŁĬĞĻĦ IJŁĻĴŁĻ ŁĞĬŁĦĽ ÌĹIJİ ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĮĴgİĞŃĨİŁİâfld ĨĮĴŁĻĽã

    # BĞĢĨŃĞĹÎ ĴĞĽĽ
    ÎĦÌ ĠĞĢĨŃĞĹÎâĽĦĬÌd ÎŁĞĬŁĦĽãf
        # SĨĮĢĦ ŃĦ ĮĦĦÎ ĻIJ İIJÎĨÌÑ IJĹĨÍÍĨĮĞĬ ŁĞĹĨĞĠĬĦd
        # ĬĦĻďĽ İĞĪĦ Ğ ĢIJĴÑ IJÌ ŁĞĬŁĦĽ ÌĨĹĽĻ
        ĽĦĬÌgÎĨĮĴŁĻĽ ł ÎŁĞĬŁĦĽgĢIJĴÑâã

        # ZĦĹIJ ÍĹĞÎĨĦĮĻ ŃÎĦĹĦ ĨĮĴŁĻ ŁĞĬŁĦĽ ŃĦĹĦ ĮĦÍĞĻĨŁĦ
        ĽĦĬÌgÎĨĮĴŁĻĽäĽĦĬÌgĨĮĴŁĻĽ l·ł flã ł fl

    # CĞĬĢŁĬĞĻĦ ĴĹĦÎĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ÎĦÌ ĴĹĦÎĨĢĻĨIJĮĽâĽĦĬÌd IJŁĻĴŁĻĽãf
        ĹĦĻŁĹĮ IJŁĻĴŁĻĽ
```

```
ƒ SIJÌĻİĞŇ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĿĽ AĢĻĨĿĞĻĨIJĮçSIJÌĻİĞŇf

    ƒ FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ƒ RĦİĦİĠĦĹ ĨĮĴŁĻ ĿĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ

        ƒ GĦĻ ŁĮĮIJĹİĞĬĨŅĦĤ ĴĹIJĠĞĠĨĬĨĻĨĦĽ
        ĦŇĴçĿĞĬŁĦĽ ł ĮĴgĦŇĴâĨĮĴŁĻĽ h ĮĴgİĞŇâĨĮĴŁĻĽd ĞŇĨĽłĝd
                                                ĪĦĦĴĤĨİĽłTĹŁĦãã
        ƒ NIJĹİĞĬĨŅĦ ĻĤĦİ ÌIJĹ ĦĞĢÎ ĽĞİĴĬĦ
        ĴĹIJĠĞĠĨĬĨĻĨĦĽ ł ĦŇĴçĿĞĬŁĦĽ w ĮĴgĽŁİâĦŇĴçĿĞĬŁĦĽd ĞŇĨĽłĝd
                                                ĪĦĦĴĤĨİĽłTĹŁĦã

        ĽĦĬÌgIJŁĻĴŁĻ ł ĴĹIJĠĞĠĨĬĨĻĨĦĽ

    ƒ BĞĢĨŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĨŃĞĹĤâĽĦĬÌd ĤĿĞĬŁĦĽãf

        ƒ CĹĦĞĻĦ ŁĮĨĮĨĻĨĞĬĨŅĦĤ ĞĹĹĞŇ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĮĴgĦİĴĻÑçĬĨĨĦâĤĿĞĬŁĦĽã

        ƒ EĮŁİĦĹĞĻĦ IJŁĻĴŁĻĽ ĞĮĤ ÍĹĞĤĨĦĮĻĽ
        ÌIJĹ ĨĮĤĦŇd âĽĨĮÍĬĦçIJŁĻĴŁĻd ĽĨĮÍĬĦçĤĿĞĬŁĦĽã ĨĮ ċ
                ĦĮŁİĦĹĞĻĦâŅĨĴâĽĦĬÌgIJŁĻĴŁĻd ĤĿĞĬŁĦĽããf
            ƒ FĬĞĻĻĦĮ IJŁĻĴŁĻ ĞĹĹĞŇ
            ĽĨĮÍĬĦçIJŁĻĴŁĻ ł ĽĨĮÍĬĦçIJŁĻĴŁĻgĹĦĽĴĞĴĦâhĝd ĝã
            ƒ CĞĬĢŁĬĞĻĦ JĞĢIJĠĨĞĮ İĞĻĹĨŇ IJÌ ĻĤĦ IJŁĻĴŁĻ ĞĮĤ
            ÏĞĢIJĠĨĞĮçİĞĻĹĨŇ ł ĮĴgĤĬĞÍÌĬĞĻâĽĨĮÍĬĦçIJŁĻĴŁĻã h ċ
                                ĮĴgĤIJĻâĽĨĮÍĬĦçIJŁĻĴŁĻd ĽĨĮÍĬĦçIJŁĻĴŁĻgTã
            ƒ CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦhŃĨĽĦ ÍĹĞĤĨĦĮĻ
            ƒ ĞĮĤ ĞĤĤ ĨĻ ĻIJ ĻĤĦ ĞĹĹĞŇ IJÌ ĽĞİĴĬĦ ÍĹĞĤĨĦĮĻĽ
            ĽĦĬÌgĤĨĮĴŁĻĽäĨĮĤĦŇã ł ĮĴgĤIJĻâÏĞĢIJĠĨĞĮçİĞĻĹĨŇd
                                        ĽĨĮÍĬĦçĤĿĞĬŁĦĽã

    ƒ CĞĬĢŁĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ĤĦÌ ĴĹĦĤĨĢĻĨIJĮĽâĽĦĬÌd IJŁĻĴŁĻĽãf
        ĹĦĻŁĹĮ ĮĴgĞĹÍİĞŇâIJŁĻĴŁĻĽd ĞŇĨĽłĝã
```

```
ƒ SĨÍİIJĨĤ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçSĨÍİIJĨĤf

    ƒ FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ƒ SĞĿĦ ĨĮĴŁĻ ĞĮĤ ĢĞĬĢŁĬĞĻĦwĽĞĿĦ IJŁĻĴŁĻ
        ƒ IJÌ ĻĤĦ ĽĨÍİIJĨĤ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĝ w âĝ κ ĮĴgĦŃĴâhĨĮĴŁĻĽãã

    ƒ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĿĞĬŁĦĽãf
        ƒ DĦĹĨĿĞĻĨĿĦ h ĢĞĬĢŁĬĞĻĦĽ ÌĹIJİ IJŁĻĴŁĻ IJÌ ĻĤĦ ĽĨÍİIJĨĤ ÌŁĮĢĻĨIJĮ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĤĿĞĬŁĦĽ ắ âĝ h ĽĦĬÌgIJŁĻĴŁĻã ắ ĽĦĬÌgIJŁĻĴŁĻ

    ƒ CĞĬĢŁĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ĤĦÌ ĴĹĦĤĨĢĻĨIJĮĽâĽĦĬÌd IJŁĻĴŁĻĽãf
        ĹĦĻŁĹĮ âIJŁĻĴŁĻĽ ń flgĥã ắ ĝ


ƒ LĨĮĦĞĹ ĞĢĻĨĿĞĻĨIJĮ
ĢĬĞĽĽ AĢĻĨĿĞĻĨIJĮçLĨĮĦĞĹf

    ƒ FIJĹŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ÌIJĹŃĞĹĤâĽĦĬÌd ĨĮĴŁĻĽd ĻĹĞĨĮĨĮÍãf
        ƒ JŁĽĻ ĹĦİĦİĠĦĹ LĞĬŁĦĽ
        ĽĦĬÌgĨĮĴŁĻĽ ł ĨĮĴŁĻĽ
        ĽĦĬÌgIJŁĻĴŁĻ ł ĨĮĴŁĻĽ

    ƒ BĞĢĪŃĞĹĤ ĴĞĽĽ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĽĦĬÌd ĤĿĞĬŁĦĽãf
        ƒ ĤĦĹĨĿĞĻĨĿĦ ĨĽ ĝd ĝ ắ ĤĿĞĬŁĦĽ ł ĤĿĞĬŁĦĽ h ĻĤĦ ĢĤĞĨĮ ĹŁĬĦ
        ĽĦĬÌgĤĨĮĴŁĻĽ ł ĤĿĞĬŁĦĽgĢIJĴÑâã

    ƒ CĞĬĢŁĬĞĻĦ ĴĹĦĤĨĢĻĨIJĮĽ ÌIJĹ IJŁĻĴŁĻĽ
    ĤĦÌ ĴĹĦĤĨĢĻĨIJĮĽâĽĦĬÌd IJŁĻĴŁĻĽãf
        ĹĦĻŁĹĮ IJŁĻĴŁĻĽ
```

```
ƒ SGD IJĴĻĨĬĨŅĦĹ
ĢĬĞĽĹ OĴĻĨĬĨŅĦĹçSGDf

    ƒ IĮĨĻĨĞĬĨŅĦ IJĴĻĨĬĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮĺĽd
    ƒ ĬĦĞĹĮĨĮĺ ĹĞĻĦ IJÌ ĝg ĨĽ ĤĦÌĞŁĬĻ ÌIJĹ ĻĴĨĽ IJĴĻĨĬĨŅĦĹ
    ĤĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮĺçĹĞĻĦłĝgd ĤĦĢĞÑłflgd ĬIJĬĦĮĻŁĬłflgãf
        ĽĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĬĦĞĹĮĨĮĺçĹĞĻĦ
        ĽĦĬÌgĤĦĢĞÑ ł ĤĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĬIJĬĦĮĻŁĬ ł ĬIJĬĦĮĻŁĬ

    ƒ CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞĬĦĻĦĹ ŁĴĤĞĻĦĽ
    ĤĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞĬĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgĤĦĢĞÑf
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĽĦĬÌgĤĦĢĞÑ ắ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    ƒ UĴĤĞĻĦ ĴĞĹĞĬĦĻĦĹĽ
    ĤĦÌ ŁĴĤĞĻĦçĴĞĹĞĬĽâĽĦĬÌd ĬĞÑĦĹãf

        ƒ IÌ ŃĦ ŁĽĦ ĬIJĬĦĮĻŁĬ
        ĨÌ ĽĦĬÌgĬIJĬĦĮĻŁĬf

            ƒ IÌ ĬĞÑĦĹ ĤIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĬIJĬĦĮĻŁĬ ĞĹĹĞÑĽd ĢĹĦĞĻĦ ĻĨĦĬ
            ƒ ÌĨĬĬĦĤ ŃĨĻĨ ŅĦĹIJĽ
            ĨÌ ĮIJĻ ĨĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨĺĨĨĻçĬIJĬĦĮĻŁĬĽďãf
                ĬĞÑĦĹgŃĦĨĺĨĨĻçĬIJĬĦĮĻŁĬĽ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgŃĦĨĺĨĨĻĽã
                ƒ IÌ ĻĨĦĹĦ ĨĽ ĮIJ ĬIJĬĦĮĻŁĬ ĞĹĹĞÑ ÌIJĹ ŃĦĨĺĨĨĻĽ
                ƒ TĨĦ ĞĹĹĞÑ ĤIJĦĽĮďĻ ĦŃĨĽĻ ÌIJĹ ĠĨĞĽĦĽ ÑĦĻ ĦĨĻĨĦĹg
                ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻŁĬĽ ł ĮĴgŅĦĹIJĽçĬĨĨĦâĬĞÑĦĹgĠĨĞĽĦĽã

            ƒ BŁĨĬĤ ŃĦĨĺĨĨĻ ŁĴĤĞĻĦĽ ŃĨĻĨ ĬIJĬĦĮĻŁĬ h ĻĞĨĦ ĴĹĦŁĨIJŁĽ
            ƒ ŁĴĤĞĻĦĽ ĬŁĬĻĨĴĬĨĦĤ ĠÑ ĹĦĻĞĨĮ ÌĞĢĻIJĹ ĞĮĤ ŁĴĤĞĻĦ ŃĨĻĨ
            ƒ ĢŁĹĹĦĮĻ ĺĹĞĤĨĦĮĻĽ
            ŃĦĨĺĨĨĻçŁĴĤĞĻĦĽ ł ċ
                ĽĦĬÌgĬIJĬĦĮĻŁĬ ắ ĬĞÑĦĹgŃĦĨĺĨĨĻçĬIJĬĦĮĻŁĬĽ h ċ
                ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ĬĞÑĦĹgĤŃĦĨĺĨĨĻĽ
            ĬĞÑĦĹgŃĦĨĺĨĨĻçĬIJĬĦĮĻŁĬĽ ł ŃĦĨĺĨĨĻçŁĴĤĞĻĦĽ

            ƒ BŁĨĬĤ ĠĨĞĽ ŁĴĤĞĻĦĽ
            ĠĨĞĽçŁĴĤĞĻĦĽ ł ċ
                ĽĦĬÌgĬIJĬĦĮĻŁĬ ắ ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻŁĬĽ h ċ
                ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮĺçĹĞĻĦ ắ ĬĞÑĦĹgĤĠĨĞĽĦĽ
            ĬĞÑĦĹgĠĨĞĽçĬIJĬĦĮĻŁĬĽ ł ĠĨĞĽçŁĴĤĞĻĦĽ
```

```
        # VĞĮĨĬĬĞ SGD łĴĤĞĻĦĽ âĞĽ ĠĦÌIJĹĦ ĬIJİĦĮĻłİ łĴĤĞĻĦã
        ĦĬĽĦf
            ŃĦĨÍÎĻçłĴĤĞĻĦĽ ł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ấ ċ
                                ĬĞÑĦĹgĤŃĦĨÍÎĻĽ
            ĠĨĞĽçłĴĤĞĻĦĽ ł hĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ấ ċ
                               ĬĞÑĦĹgĤĠĨĞĽĦĽ

        # UĴĤĞĻĦ ŃĦĨÍÎĻĽ ĞĮĤ ĠĨĞĽĦĽ łĽĨĮÍ ĦĨĻÎĦĹ
        # ŁĞĮĨĬĬĞ IJĹ ĬIJİĦĮĻłİ łĴĤĞĻĦĽ
        ĬĞÑĦĹgŃĦĨÍÎĻĽ κł ŃĦĨÍÎĻçłĴĤĞĻĦĽ
        ĬĞÑĦĹgĠĨĞĽĦĽ κł ĠĨĞĽçłĴĤĞĻĦĽ

    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ łĴĤĞĻĦĽ
    ÂĦÌ ĴIJĽĻçłĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ


# AĤĞÍĹĞĤ IJĴĻĨĬĬŅĦĹ
ĢĬĞĽĽ OĴĻĨĬĬŅĦĹçAĤĞÍĹĞĤf

    # IĮĨĻĨĞĬĬŅĦ IJĴĻĨĬĬŅĦĹ h ĽĦĻ ĽĦĻĻĨĮÍĽ
    ÂĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłĝgd ÂĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìãf
        ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgÂĦĢĞÑ ł ÂĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ łĴĤĞĻĦĽ
    ÂĦÌ ĴĹĦçłĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgÂĦĢĞÑf
            ĽĦĬÌgĢłĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ấ ċ
                âĝg w âĝg κ ĽĦĬÌgÂĦĢĞÑ ấ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    # UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
    ÂĦÌ łĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ÂIJĦĽ ĮIJĻ ĢIJĮĻĞĨĮ ĢĞĢÎĦ ĞĹĹĞÑĽd
        # ĢĹĦĞĻĦ ĻÎĦİ ÌĨĬĬĦÂ ŃĨĻÎ ŅĦĹIJĽ
        ĨÌ ĮIJĻ ÎĞĽĞĻĻĹâĬĞÑĦĹd dŃĦĨÍÎĻçĢĞĢÎĦdãf
            ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨÍÎĻĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

        # UĴĤĞĻĦ ĢĞĢÎĦ ŃĨĻÎ ĽĶŁĞĹĦÂ ĢŁĹĹĦĮĻ ÍĹĞÂĨĬĦĮĻĽ
        ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ κł ĬĞÑĦĹgÂŃĦĨÍÎĻĽấấğ
        ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ κł ĬĞÑĦĹgÂĠĨĞĽĦĽấấğ
```

```
        # VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ ĸ ĮIJĹİĞĬĨŅĞĻĨIJĮ
        # ŃĨĻÎ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢÎĦ
        ĬĞÑĦĹgŃĦĨÍÎĻĽ ĸł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                         ĬĞÑĦĹgĤŃĦĨÍÎĻĽ w ċ
                         âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦã ĸ ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ ĸł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                        ĬĞÑĦĹgĤĠĨĞĽĦĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦã ĸ ĽĦĬÌgĦĴĽĨĬIJĮã

    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ÎĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ĸł ĝ


# RMSĴĹIJĴ IJĴĻĨĬĨŅĦĹ
ĢĬĞĽĽ OĴĻĨĬĨŅĦĹçRMSĴĹIJĴf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨĬĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮÍĽ
    ÎĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłflgflflĝd ÎĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìd
                ĹÎIJłflgîãf
        ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĤĦĢĞÑ ł ÎĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬÌgĹÎIJ ł ĹÎIJ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ÎĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgĤĦĢĞÑf
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                âĝg w âĝg ĸ ĽĦĬÌgĤĦĢĞÑ ắ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    # UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
    ÎĦÌ łĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ÎIJĦĽ ĮIJĮ ĢIJĮĻĞĨĮ ĢĞĢÎĦ ĞĹĹĞÑĽd
        # ĢĹĦĞĻĦ ĻÎĦİ ÌĨĬĬĦĤ ŃĨĻÎ ŅĦĹIJĽ
        ĨÌ ĮIJĮ ÎĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍÎĻçĢĞĢÎĦďãf
            ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨÍÎĻĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã

        # UĴĤĞĻĦ ĢĞĢÎĦ ŃĨĻÎ ĽĶŁĞĹĦĤ ĢŁĹĹĦĮĻ ÍĹĞĤĨĦĮĻĽ
        ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĽĦĬÌgĹÎIJ ắ ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ĸ ċ
            âĝ h ĽĦĬÌgĹÎIJã ắ ĬĞÑĦĹgĤŃĦĨÍÎĻĽắắĝ
        ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĽĦĬÌgĹÎIJ ắ ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ĸ ċ
            âĝ h ĽĦĬÌgĹÎIJã ắ ĬĞÑĦĹgĤĠĨĞĽĦĽắắĝ
```

```
        # VĞĮĨĬĬĞ SGD ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦ κ ĮIJĹİĞĬĨŅĞĻĨIJĮ
        # ŃĨĻÎ ĽĶŁĞĹĦ ĹIJIJĻĦĤ ĢĞĢÎĦ
        ĬĞÑĦĹgŃĦĨÍÎĻĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                         ĬĞÑĦĹgĤŃĦĨÍÎĻĽ w ċ
                         âĮĴgĽĶĹĻâĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã
        ĬĞÑĦĹgĠĨĞĽĦĽ κł hĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                        ĬĞÑĦĹgĤĠĨĞĽĦĽ w ċ
                        âĮĴgĽĶĹĻâĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦã κ ĽĦĬÌgĦĴĽĨĬIJĮã

    # CĞĬĬ IJĮĢĦ ĞÌĻĦĹ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ÎĦÌ ĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ κł ĝ


# AĤĞİ IJĴĻĨİĨŅĦĹ
ĢĬĞĽĽ OĴĻĨİĨŅĦĹçAĤĞİf

    # IĮĨĻĨĞĬĨŅĦ IJĴĻĨİĨŅĦĹ h ĽĦĻ ĽĦĻĻĨĮÍĽ
    ÎĦÌ ççĨĮĨĻççâĽĦĬÌd ĬĦĞĹĮĨĮÍçĹĞĻĦłflgflflĝd ÎĦĢĞÑłflgd ĦĴĽĨĬIJĮłĝĦhìd
                ĠĦĻĞçĝłflgîd ĠĦĻĞçğłflgîîîãf
        ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĬĦĞĹĮĨĮÍçĹĞĻĦ
        ĽĦĬÌgĤĦĢĞÑ ł ÎĦĢĞÑ
        ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽ ł fl
        ĽĦĬÌgĦĴĽĨĬIJĮ ł ĦĴĽĨĬIJĮ
        ĽĦĬÌgĠĦĻĞçĝ ł ĠĦĻĞçĝ
        ĽĦĬÌgĠĦĻĞçğ ł ĠĦĻĞçğ

    # CĞĬĬ IJĮĢĦ ĠĦÌIJĹĦ ĞĮÑ ĴĞĹĞİĦĻĦĹ ŁĴĤĞĻĦĽ
    ÎĦÌ ĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌãf
        ĨÌ ĽĦĬÌgĤĦĢĞÑf
            ĽĦĬÌgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦ ł ĽĦĬÌgĬĦĞĹĮĨĮÍçĹĞĻĦ ắ ċ
                âĝg w âĝg κ ĽĦĬÌgĤĦĢĞÑ ắ ĽĦĬÌgĨĻĦĹĞĻĨIJĮĽãã

    # UĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽ
    ÎĦÌ ŁĴĤĞĻĦçĴĞĹĞİĽâĽĦĬÌd ĬĞÑĦĹãf

        # IÌ ĬĞÑĦĹ ÂIJĦĽ ĮIJĮ ĢIJĮĻĞĨĮ ĢĞĢÎĦ ĞĹĹĞÑĽd
        # ĢĹĦĞĻĦ ĻÎĦİ ÌĨĬĬĦĤ ŃĨĻÎ ŅĦĹIJĽ
        ĨÌ ĮIJĮ ÎĞĽĞĻĻĹâĬĞÑĦĹd ďŃĦĨÍÎĻçĢĞĢÎĦďãf
            ĬĞÑĦĹgŃĦĨÍÎĻçĬIJİĦĮĻŁİĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨÍÎĻĽã
            ĬĞÑĦĹgŃĦĨÍÎĻçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgŃĦĨÍÎĻĽã
            ĬĞÑĦĹgĠĨĞĽçĬIJİĦĮĻŁİĽ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã
            ĬĞÑĦĹgĠĨĞĽçĢĞĢÎĦ ł ĮĴgŅĦĹIJĽçĬĨĬĦâĬĞÑĦĹgĠĨĞĽĦĽã
```

```
    fi UĴĤĞĻH İIJİHIĻŁİ  ŃÑĻÎ ĢŁĹĹHIĻ ÍĹĞĤÑHIĻĽ
    ĬĞÑHĹgŃHÑÍÍÎĻçİIJİHIĻŁİĽ ł ĽHĬÌgĠHĻĞçĝ ấ ċ
                              ĬĞÑHĹgŃHÑÍÍÎĻçİIJİHIĻŁİĽ κ ċ
                              âĝ h ĽHĬÌgĠHĻĞçĝã ấ ĬĞÑHĹgĤŃHÑÍÍÎĻĽ
    ĬĞÑHĹgĠÑĞĽçİIJİHIĻŁİĽ ł ĽHĬÌgĠHĻĞçĝ ấ ċ
                            ĬĞÑHĹgĠÑĞĽçİIJİHIĻŁİĽ κ ċ
                            âĝ h ĽHĬÌgĠHĻĞçĝã ấ ĬĞÑHĹgĤĠÑĞĽHĽ
    fi GHĻ ĢIJĹĹHĢĻHĤ İIJİHIĻŁİ
    fi ĽHĬÌgÑĻHĹĞĻÑIJI ÑĽ fl ĞĻ ÌÑĹĽĻ ĴĞĽĽ
    fi ĞIĤ ŃH IHHĤ ĻIJ ĽĻĞĹĻ ŃÑĻÎ ĝ ÎHĹH
    ŃHÑÍÍÎĻçİIJİHIĻŁİĽçĢIJĹĹHĢĻHĤ ł ĬĞÑHĹgŃHÑÍÍÎĻçİIJİHIĻŁİĽ w ċ
        âĝ h ĽHĬÌgĠHĻĞçĝ ấấ âĽHĬÌgÑĻHĹĞĻÑIJIĽ κ ĝãã
    ĠÑĞĽçİIJİHIĻŁİĽçĢIJĹĹHĢĻHĤ ł ĬĞÑHĹgĠÑĞĽçİIJİHIĻŁİĽ w ċ
        âĝ h ĽHĬÌgĠHĻĞçĝ ấấ âĽHĬÌgÑĻHĹĞĻÑIJIĽ κ ĝãã
    fi UĴĤĞĻH ĢĞĢÎH ŃÑĻÎ ĽĶŁĞĹHĤ ĢŁĹĹHIĻ ÍĹĞĤÑHIĻĽ
    ĬĞÑHĹgŃHÑÍÍÎĻçĢĞĢÎH ł ĽHĬÌgĠHĻĞçğ ấ ĬĞÑHĹgŃHÑÍÍÎĻçĢĞĢÎH κ ċ
        âĝ h ĽHĬÌgĠHĻĞçğã ấ ĬĞÑHĹgĤŃHÑÍÍÎĻĽấấğ
    ĬĞÑHĹgĠÑĞĽçĢĞĢÎH ł ĽHĬÌgĠHĻĞçğ ấ ĬĞÑHĹgĠÑĞĽçĢĞĢÎH κ ċ
        âĝ h ĽHĬÌgĠHĻĞçğã ấ ĬĞÑHĹgĤĠÑĞĽHĽấấğ
    fi GHĻ ĢIJĹĹHĢĻHĤ ĢĞĢÎH
    ŃHÑÍÍÎĻçĢĞĢÎHçĢIJĹĹHĢĻHĤ ł ĬĞÑHĹgŃHÑÍÍÎĻçĢĞĢÎH w ċ
        âĝ h ĽHĬÌgĠHĻĞçğ ấấ âĽHĬÌgÑĻHĹĞĻÑIJIĽ κ ĝãã
    ĠÑĞĽçĢĞĢÎHçĢIJĹĹHĢĻHĤ ł ĬĞÑHĹgĠÑĞĽçĢĞĢÎH w ċ
        âĝ h ĽHĬÌgĠHĻĞçğ ấấ âĽHĬÌgÑĻHĹĞĻÑIJIĽ κ ĝãã

    fi VĞIÑĬĬĞ SGD ĴĞĹĞİHĻHĹ ŁĴĤĞĻH κ IIJĹİĞĬÑŃĞĻÑIJI
    fi ŃÑĻÎ ĽĶŁĞĹH ĹIJJĻHĤ ĢĞĢÎH
    ĬĞÑHĹgŃHÑÍÍÎĻĽ κł hĽHĬÌgĢŁĹĹHIĻçĬHĞĹIÑIÍçĹĞĻH ấ ċ
                      ŃHÑÍÍÎĻçİIJİHIĻŁİĽçĢIJĹĹHĢĻHĤ w ċ
                      âIĴgĽĶĹĻâŃHÑÍÍÎĻçĢĞĢÎHçĢIJĹĹHĢĻHĤã κ
                          ĽHĬÌgHĴĽÑĬIJIã
    ĬĞÑHĹgĠÑĞĽHĽ κł hĽHĬÌgĢŁĹĹHIĻçĬHĞĹIÑIÍçĹĞĻH ấ ċ
                      ĠÑĞĽçİIJİHIĻŁİĽçĢIJĹĹHĢĻHĤ w ċ
                      âIĴgĽĶĹĻâĠÑĞĽçĢĞĢÎHçĢIJĹĹHĢĻHĤã κ
                          ĽHĬÌgHĴĽÑĬIJIã

fi CĞĬĬ IJIĢH ĞÌĻHĹ ĞIÑ ĴĞĹĞİHĻHĹ ŁĴĤĞĻHĽ
ĤHÌ ĴIJĽĻçŁĴĤĞĻHçĴĞĹĞİĽâĽHĬÌãf
    ĽHĬÌgÑĻHĹĞĻÑIJIĽ κł ĝ
```

```
# CIJIIIJI IIJLL GIGLL
GIGLL LIJLLf

    # RHILIGLIINGLIIJI IIJLL GGIGLIGLIIJI
    ÂHI LHILIGLIINGLIIJIcIIJLLâLHIIãf

        # fl GÑ ÂHIGLII
        LHILIGLIINGLIIJIcIIJLL ł fl

        # CGIGLIGLH LHILIGLIINGLIIJI IIJLL
        # ILHLGLH GII LLGIIGGIH IGÑHLL
        IIJL IGÑHL II LHIIgLLGIIGGIHcIGÑHLLf

            # Lĝ LHILIGLIINGLIIJI h NHIIIILL
            # GGIGLIGLH IJIIÑ NIHI IGGLIJL ILHGLHL LIGI fl
            II IGÑHLgNHIIILcLHILIGLIINHLcIĝ ń flf
                LHILIGLIINGLIIJIcIIJLL κł IGÑHLgNHIIILcLHILIGLIINHLcIĝ ấ ċ
                                          IJgLLIâIJgGGLâIGÑHLgNHIIILLãã

            # Lg LHILIGLIINGLIIJI h NHIIIILL
            II IGÑHLgNHIIILcLHILIGLIINHLcIg ń flf
                LHILIGLIINGLIIJIcIIJLL κł IGÑHLgNHIIILcLHILIGLIINHLcIg ấ ċ
                                          IJgLLIâIGÑHLgNHIIILL ấ ċ
                                                 IGÑHLgNHIIILLã

            # Lĝ LHILIGLIINGLIIJI h GIGLHL
            # GGIGLIGLH IJIIÑ NIHI IGGLIJL ILHGLHL LIGI fl
            II IGÑHLgGIGLcLHILIGLIINHLcIĝ ń flf
                LHILIGLIINGLIIJIcIIJLL κł IGÑHLgGIGLcLHILIGLIINHLcIĝ ấ ċ
                                          IJgLLIâIJgGGLâIGÑHLgGIGLHLãã

            # Lg LHILIGLIINGLIIJI h GIGLHL
            II IGÑHLgGIGLcLHILIGLIINHLcIg ń flf
                LHILIGLIINGLIIJIcIIJLL κł IGÑHLgGIGLcLHILIGLIINHLcIg ấ ċ
                                          IJgLLIâIGÑHLgGIGLHL ấ ċ
                                                 IGÑHLgGIGLHLã

        LHLLLI LHILIGLIINGLIIJIcIIJLL

    # SHLwLHIHIGHL LLGIIGGIH IGÑHLL
    ÂHI LHIHIGHLcLLGIIGGIHcIGÑHLLâLHIId LLGIIGGIHcIGÑHLLãf
        LHIIgLLGIIGGIHcIGÑHLL ł LLGIIGGIHcIGÑHLL
```

```
    fi CĞĬĢŁĬĞĻĦĽ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹÎŅĞĻĨIJĮ ĬIJĹĽĦĽ
    fi ÍÎŁĦĮ ĬIJÂĦĬ IJŁĻĴŁĻ ĞĮÂ ÍĹIJŁĮÂ ĻĹŁĻÎ ŁĞĬŁĦĽ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦâĽĦĬÌd IJŁĻĴŁĻd Ñd ős ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹÎŅĞĻĨIJĮłFĞĬĽĦãf

        fi CĞĬĢŁĬĞĻĦ ĽĞĬĴĬĦ ĬIJĹĽĦĽ
        ĽĞĬĴĬĦçĬIJĹĽĦĽ ł ĽĦĬÌgÌIJĹŃĞĹÂâIJŁĻĴŁĻd Ñã

        fi CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĹĽ
        ÂĞĻĞçĬIJĹĽ ł ĮĴgĬĦĞĮâĽĞĬĴĬĦçĬIJĹĽĦĽã

        fi AÂÂ ĞĢĢŁĬŁĬĞĻĦÂ ĽŁĬ IJÌ ĬIJĹĽĦĽ ĞĮÂ ĽĞĬĴĬĦ ĢIJŁĮĻ
        ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĽŁĬ κł ĮĴgĽŁĬâĽĞĬĴĬĦçĬIJĹĽĦĽã
        ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĢIJŁĮĻ κł ĬĦĮâĽĞĬĴĬĦçĬIJĹĽĦĽã

        fi IÌ ĬŁĽĻ ÂĞĻĞ ĬIJĹĽ h ĹĦĻŁĹĮ ĨĻ
        ĨÌ ĮIJĻ ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹÎŅĞĻĨIJĮf
            ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĽ

        fi RĦĻŁĹĮ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹÎŅĞĻĨIJĮ ĬIJĹĽĦĽ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĽd ĽĦĬÌgĹĦÍŁĬĞĹÎŅĞĻĨIJĮçĬIJĹĽâã

    fi CĞĬĢŁĬĞĻĦĽ ĞĢĢŁĬŁĬĞĻĦÂ ĬIJĹĽ
    ÂĦÌ ĢĞĬĢŁĬĞĻĦçĞĢĢŁĬŁĬĞĻĦÂâĽĦĬÌd ős ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹÎŅĞĻĨIJĮłFĞĬĽĦãf

        fi CĞĬĢŁĬĞĻĦ ĬĦĞĮ ĬIJĹĽ
        ÂĞĻĞçĬIJĹĽ ł ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĽŁĬ w ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĢIJŁĮĻ

        fi IÌ ĬŁĽĻ ÂĞĻĞ ĬIJĹĽ h ĹĦĻŁĹĮ ĨĻ
        ĨÌ ĮIJĻ ĨĮĢĬŁÂĦçĹĦÍŁĬĞĹÎŅĞĻĨIJĮf
            ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĽ

        fi RĦĻŁĹĮ ĻÎĦ ÂĞĻĞ ĞĮÂ ĹĦÍŁĬĞĹÎŅĞĻĨIJĮ ĬIJĹĽĦĽ
        ĹĦĻŁĹĮ ÂĞĻĞçĬIJĹĽd ĽĦĬÌgĹĦÍŁĬĞĹÎŅĞĻĨIJĮçĬIJĹĽâã

    fi RĦĽĦĻ ŁĞĹÎĞĠĬĦĽ ÌIJĹ ĞĢĢŁĬŁĬĞĻĦÂ ĬIJĹĽ
    ÂĦÌ ĮĦŃçĴĞĽĽâĽĦĬÌãf
        ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĽŁĬ ł fl
        ĽĦĬÌgĞĢĢŁĬŁĬĞĻĦÂçĢIJŁĮĻ ł fl


fi CĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ
ĢĬĞĽĽ LIJĹĽçCĞĻĦÍIJĹÎĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâĽIJĹĽãf

    fi FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ÑçĴĹĦÂd ÑçĻĹŁĦãf

        fi NŁĬĠĦĹ IJÌ ĽĞĬĴĬĦĽ ĨĮ Ğ ĠĞĻĢÎ
        ĽĞĬĴĬĦĽ ł ĬĦĮâÑçĴĹĦÂã
```

```
    ſ CĬĨĴ ÂĞĻĞ ĻIJ ĴĹĦĿĦĮĻ ÂĨĿĨĹĨIJĮ ĠÑ ſl
    ſ CĬĨĴ ĠIJĻÎ ĹĨÂĦĹ ĻIJ ĮIJĻ ÂĹĞÍ İĦĞĮ ĻIJŃĞĹÂĿ ĞĮÑ ĿĞĬŁĦ
    ÑçĴĹĦÂçĢĬĨĴĴĦÂ ł ĮĴgĢĬĨĴâÑçĴĹĦÂd ĝĦhìd ĝ h ĝĦhìã

    ſ PĹIJĠĞĠĨĬĨĻĨĦĹ ÌIJĹ ĻĞĹÍĦĻ ĿĞĬŁĦĹ h
    ſ IJĮĬÑ ĨÌ ĢĞĻĦÍIJĹĨĢĞĬ ĬĞĠĦĬĹ
    ĨÌ ĬĦĮâÑçĻĹŁĦgĹÎĞĴĦã łł ĝf
        ĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĹ ł ÑçĴĹĦÂçĢĬĨĴĴĦÂä
            ĹĞĮÍĦâĹĞİĴĬĦĹãd
            ÑçĻĹŁĦ
        ã

    ſ MĞĹĨ ĿĞĬŁĦĹ h IJĮĬÑ ÌIJĹ IJĮĦhÎIJĻ ĦĮĢIJÂĦÂ ĬĞĠĦĬĹ
    ĦĬĨÌ ĬĦĮâÑçĻĹŁĦgĹÎĞĴĦã łł ğf
        ĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĹ ł ĮĴgĹŁİâ
            ÑçĴĹĦÂçĢĬĨĴĴĦÂ ǻ ÑçĻĹŁĦd
            ĞŃĨĹłĝ
        ã

    ſ LIJĹĹĦĹ
    ĮĦÍĞĻĨĿĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĹ ł hĮĴgĬIJÍâĢIJĹĹĦĢĻçĢIJĮÌĨÂĦĮĢĦĹã
    ĹĦĻŁĹĮ ĮĦÍĞĻĨĿĦçĬIJÍçĬĨĨĦĬĨÎIJIJÂĹ

ſ BĞĢĪŃĞĹÂ ĴĞĹĹ
ÂĦÌ ĠĞĢĪŃĞĹÂâĹĦĬÌd ÂĿĞĬŁĦĹd ÑçĻĹŁĦãf

    ſ NŁİĠĦĹ IJÌ ĹĞİĴĬĦĹ
    ĹĞİĴĬĦĹ ł ĬĦĮâÂĿĞĬŁĦĹã
    ſ NŁİĠĦĹ IJÌ ĬĞĠĦĬĹ ĨĮ ĦĿĦĹÑ ĹĞİĴĬĦ
    ſ WĦďĬĬ ŁĹĦ ĻÎĦ ÌĨĹĹĻ ĹĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻÎĦİ
    ĬĞĠĦĬĹ ł ĬĦĮâÂĿĞĬŁĦĹäflãã

    ſ IÌ ĬĞĠĦĬĹ ĞĹĦ ĹĴĞĹĹĦd ĻŁĹĮ ĻÎĦİ ĨĮĻIJ IJĮĦhÎIJĻ ĿĦĢĻIJĹ
    ĨÌ ĬĦĮâÑçĻĹŁĦgĹÎĞĴĦã łł ĝf
        ÑçĻĹŁĦ ł ĮĴgĦÑĦâĬĞĠĦĬĹãäÑçĻĹŁĦã

    ſ CĞĬĢŁĬĞĻĦ ÍĹĞÂĨĦĮĻ
    ĹĦĬÌgÂĨĮĴŁĻĹ ł hÑçĻĹŁĦ w ÂĿĞĬŁĦĹ
    ſ NIJĹİĞĬĨŅĦ ÍĹĞÂĨĦĮĻ
    ĹĦĬÌgÂĨĮĴŁĻĹ ł ĹĦĬÌgÂĨĮĴŁĻĹ w ĹĞİĴĬĦĹ
```

```
ƒ SIJÌĻİĞŇ ĢĬĞĽĹÎÌÎĦĹ h ĢIJİĠÎĮĦÂ SIJÌĻİĞŇ ĞĢĻÎŁĞĻÎIJĮ
ƒ ĞĮÂ ĢĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ ÌIJĹ ÌĞĽĻĦĹ ĠĞĢĪŃĞĹÂ ĽĻĦĴ
ĢĬĞĽĽ AĢĻÎŁĞĻÎIJĮçSIJÌĻİĞŇçLIJĹĽçCĞĻĦÍIJĹÎĢĞĬCĹIJĹĽĦĮĻĹIJĴÑâãf

    ƒ BĞĢĪŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ĠĞĢĪŃĞĹÂâĽĦĬÌd ÂŁĞĬŁĦĽd ÑçĻĹŁĦãf

        ƒ NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâÂŁĞĬŁĦĽã

        ƒ IÌ ĬĞĠĦĬĽ ĞĹĦ IJĮĦhÎIJĻ ĦĮĢIJÂĦÂd
        ƒ ĻŁĹĮ ĻÎĦİ ÎĮĻIJ ÂÎĽĢĹĦĻĦ ŁĞĬŁĦĽ
        ÎÌ ĬĦĮâÑçĻĹŁĦgĽÎĞĴĦã łł ğf
            ÑçĻĹŁĦ ł ĮĴgĞĹÍİĞŇâÑçĻĹŁĦd ĞŇÎĽłĝã

        ƒ CIJĴÑ ĽIJ ŃĦ ĢĞĮ ĽĞÌĦĬÑ İIJÂÎÌÑ
        ĽĦĬÌgÂÎĮĴŁĻĽ ł ÂŁĞĬŁĦĽgĢIJĴÑâã
        ƒ CĞĬĢŁĬĞĻĦ ÍĹĞÂÎĦĮĻ
        ĽĦĬÌgÂÎĮĴŁĻĽäĹĞĮÍĦâĽĞİĴĬĦĽãd ÑçĻĹŁĦã hł ĝ
        ƒ NIJĹİĞĬÎŅĦ ÍĹĞÂÎĦĮĻ
        ĽĦĬÌgÂÎĮĴŁĻĽ ł ĽĦĬÌgÂÎĮĴŁĻĽ w ĽĞİĴĬĦĽ


ƒ BÎĮĞĹÑ ĢĹIJĹĽhĦĮĻĹIJĴÑ ĬIJĹĽ
ĢĬĞĽĽ LIJĹĽçBÎĮĞĹÑCĹIJĹĽĦĮĻĹIJĴÑâLIJĹĽãf

    ƒ FIJĹŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ÌIJĹŃĞĹÂâĽĦĬÌd ÑçĴĹĦÂd ÑçĻĹŁĦãf

        ƒ CĬÎĴ ÂĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ÂÎŁÎĽÎIJĮ ĠÑ fl
        ƒ CĬÎĴ ĠIJĻÎ ĽÎÂĦĽ ĻIJ ĮIJĻ ÂĹĞÍ İĦĞĮ ĻIJŃĞĹÂĽ ĞĮÑ ŁĞĬŁĦ
        ÑçĴĹĦÂçĢĬÎĴĴĦÂ ł ĮĴgĢĬÎĴâÑçĴĹĦÂd ĝĦhìd ĝ h ĝĦhìã

        ƒ CĞĬĢŁĬĞĻĦ ĽĞİĴĬĦhŃÎĽĦ ĬIJĹĽ
        ĽĞİĴĬĦçĬIJĹĽĦĽ ł hâÑçĻĹŁĦ ǻ ĮĴgĬIJÍâÑçĴĹĦÂçĢĬÎĴĴĦÂã κ
                          âĝ h ÑçĻĹŁĦã ǻ ĮĴgĬIJÍâĝ h ÑçĴĹĦÂçĢĬÎĴĴĦÂãã
        ĽĞİĴĬĦçĬIJĹĽĦĽ ł ĮĴgİĦĞĮâĽĞİĴĬĦçĬIJĹĽĦĽd ĞŇÎĽłhĝã

        ƒ RĦĻŁĹĮ ĬIJĹĽĦĽ
        ĹĦĻŁĹĮ ĽĞİĴĬĦçĬIJĹĽĦĽ

    ƒ BĞĢĪŃĞĹÂ ĴĞĽĽ
    ÂĦÌ ĠĞĢĪŃĞĹÂâĽĦĬÌd ÂŁĞĬŁĦĽd ÑçĻĹŁĦãf

        ƒ NŁİĠĦĹ IJÌ ĽĞİĴĬĦĽ
        ĽĞİĴĬĦĽ ł ĬĦĮâÂŁĞĬŁĦĽã
        ƒ NŁİĠĦĹ IJÌ IJŁĻĴŁĻĽ ÎĮ ĦŁĦĹŃ ĽĞİĴĬĦ
        ƒ WĦďĬĬ ŁĽĦ ĻÎĦ ÌÎĹĽĻ ĽĞİĴĬĦ ĻIJ ĢIJŁĮĻ ĻÎĦİ
        IJŁĻĴŁĻĽ ł ĬĦĮâÂŁĞĬŁĦĽäflãã
```

```
        # CĬĨĴ ĤĞĻĞ ĻIJ ĴĹĦŁĦĮĻ ĤĨŁĨĹĨIJĮ ĠÑ #
        # CĬĨĴ ĠIJĻĤ ĹĨĤĦĹ ĻIJ ĮIJĻ ĤĹĞÍ ĬĦĞĮ ĻIJŃĞĹĤĹ ĞĮÑ ŁĞĬŁĦ
        ĢĬĨĴĴĦĤçĤŁĞĬŁĦĹ ł ĮĴgĢĬĨĴâĤŁĞĬŁĦĹd ĝĦhìd ĝ h ĝĦhìã

        # CĞĬĢŁĬĞĻĦ ÍĹĞĤĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł hâÑçĻĹŁĦ w ĢĬĨĴĴĦĤçĤŁĞĬŁĦĹ h
                        âĝ h ÑçĻĹŁĦã w âĝ h ĢĬĨĴĴĦĤçĤŁĞĬŁĦĹãã w IJŁĻĴŁĻĹ
        # NIJĹĬĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĹĦĬÌgĤĨĮĴŁĻĹ w ĹĞĬĴĬĦĹ


# MĦĞĮ SĶŁĞĹĦĤ EĹĹIJĹ ĬIJĹĹ
ĢĬĞĹĹ LIJĹĹçMĦĞĮSĶŁĞĹĦĤEĹĹIJĹâLIJĹĹãf  # Lğ ĬIJĹĹ

    # FIJĹŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        # CĞĬĢŁĬĞĻĦ ĬIJĹĹ
        ĹĞĬĴĬĦçĬIJĹĹĦĹ ł ĮĴgĬĦĞĮââÑçĻĹŁĦ h ÑçĴĹĦĤãááğd ĞŃĨĹłhĝã

        # RĦĻŁĹĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ĹĞĬĴĬĦçĬIJĹĹĦĹ

    # BĞĢĪŃĞĹĤ ĴĞĹĹ
    ĤĦÌ ĠĞĢĪŃĞĹĤâĹĦĬÌd ĤĹĞĬŁĦĹd ÑçĻĹŁĦãf

        # NŁĬĠĦĹ IJÌ ĹĞĬĴĬĦĹ
        ĹĞĬĴĬĦĹ ł ĬĦĮâĤĹĞĬŁĦĹã
        # NŁĬĠĦĹ IJÌ IJŁĻĴŁĻĹ ĨĮ ĦĹĦĹÑ ĹĞĬĴĬĦ
        # WĦďĬĬ ŁĹĦ ĻĤĦ ÌĨĹĹĻ ĹĞĬĴĬĦ ĻIJ ĢIJŁĮĻ ĻĤĦĬ
        IJŁĻĴŁĻĹ ł ĬĦĮâĤĹĞĬŁĦĹäflãã

        # GĹĞĤĨĦĮĻ IJĮ ĹĞĬŁĦĹ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł hğ á âÑçĻĹŁĦ h ĤĹĞĬŁĦĹã w IJŁĻĴŁĻĹ
        # NIJĹĬĞĬĨŅĦ ÍĹĞĤĨĦĮĻ
        ĹĦĬÌgĤĨĮĴŁĻĹ ł ĹĦĬÌgĤĨĮĴŁĻĹ w ĹĞĬĴĬĦĹ


# MĦĞĮ AĠĹIJĬŁĻĦ EĹĹIJĹ ĬIJĹĹ
ĢĬĞĹĹ LIJĹĹçMĦĞĮAĠĹIJĬŁĻĦEĹĹIJĹâLIJĹĹãf  # Lĝ ĬIJĹĹ

    ĤĦÌ ÌIJĹŃĞĹĤâĹĦĬÌd ÑçĴĹĦĤd ÑçĻĹŁĦãf

        # CĞĬĢŁĬĞĻĦ ĬIJĹĹ
        ĹĞĬĴĬĦçĬIJĹĹĦĹ ł ĮĴgĬĦĞĮâĮĴgĞĠĹâÑçĻĹŁĦ h ÑçĴĹĦĤãd ĞŃĨĹłhĝã

        # RĦĻŁĹĮ ĬIJĹĹĦĹ
        ĹĦĻŁĹĮ ĹĞĬĴĬĦçĬIJĹĹĦĹ
```

```
    fi BĞĢĪŃĞĹĤ ĴĞĽĽ
    ÎHÌ ĠĞĢĪŃĞĹĤâĽHĬÌd ÎĹĞĬŁHĹd ÑçĻĹŁHãf

        fi NŁİĠHĹ IJÌ ĽĞİĴĬHĽ
        ĽĞİĴĬHĽ ł ĬHĮâĤĹĞĬŁHĽã
        fi NŁİĠHĹ IJÌ IJŁĻĴŁĻĽ ĨĮ HĹHĹÑ ĽĞİĴĬH
        fi WHďĬĬ ŁĽH ĻĤH ÌĨĹĽĻ ĽĞİĴĬH ĻIJ ĢIJŁĮĻ ĻĤHİ
        IJŁĻĴŁĻĽ ł ĬHĮâĤĹĞĬŁHĽäflãã

        fi CĞĬĢŁĬĞĻH ÍĹĞĤĨHĮĻ
        ĽHĬÌgĤĨĮĴŁĻĽ ł ĮĴgĽĨÍĮâÑçĻĹŁH h ĤĹĞĬŁHĽã w IJŁĻĴŁĻĽ
        fi NIJĹİĞĬĨŅH ÍĹĞĤĨHĮĻ
        ĽHĬÌgĤĨĮĴŁĻĽ ł ĽHĬÌgĤĨĮĴŁĻĽ w ĽĞİĴĬHĽ


fi CIJİİIJĮ ĞĢĢŁĹĞĢÑ ĢĬĞĽĽ
ĢĬĞĽĽ AĢĢŁĹĞĢÑf

    fi CĞĬĢŁĬĞĻHĽ ĞĮ ĞĢĢŁĹĞĢÑ
    fi ÍĨĹHĮ ĴĹHĤĨĢĻĨIJĮĽ ĞĮĤ ÍĹIJŁĮĤ ĻĹŁĻÎ ĹĞĬŁHĽ
    ÎHÌ ĢĞĬĢŁĬĞĻHâĽHĬÌd ĴĹHĤĨĢĻĨIJĮĽd Ñãf

        fi GHĻ ĢIJİĴĞĹĨĽIJĮ ĹHĽŁĬĻĽ
        ĢIJİĴĞĹĨĽIJĮĽ ł ĽHĬÌgĢIJİĴĞĹHâĴĹHĤĨĢĻĨIJĮĽd Ñã

        fi CĞĬĢŁĬĞĻH ĞĮ ĞĢĢŁĹĞĢÑ
        ĞĢĢŁĹĞĢÑ ł ĮĴgİHĞĮâĢIJİĴĞĹĨĽIJĮĽã

        fi AĤĤ ĞĢĢŁİŁĬĞĻHĤ ĽŁİ IJÌ İĞĻĢĨĨĮÍ ĹĞĬŁHĽ ĞĮĤ ĽĞİĴĬH ĢIJŁĮĻ
        ĽHĬÌgĞĢĢŁİŁĬĞĻHĤçĽŁİ κł ĮĴgĽŁİâĢIJİĴĞĹĨĽIJĮĽã
        ĽHĬÌgĞĢĢŁİŁĬĞĻHĤçĢIJŁĮĻ κł ĬHĮâĢIJİĴĞĹĨĽIJĮĽã

        fi RHĻŁĹĮ ĞĢĢŁĹĞĢÑ
        ĹHĻŁĹĮ ĞĢĢŁĹĞĢÑ

    fi CĞĬĢŁĬĞĻHĽ ĞĢĢŁİŁĬĞĻHĤ ĞĢĢŁĹĞĢÑ
    ÎHÌ ĢĞĬĢŁĬĞĻHçĞĢĢŁİŁĬĞĻHĤâĽHĬÌãf

        fi CĞĬĢŁĬĞĻH ĞĮ ĞĢĢŁĹĞĢÑ
        ĞĢĢŁĹĞĢÑ ł ĽHĬÌgĞĢĢŁİŁĬĞĻHĤçĽŁİ w ĽHĬÌgĞĢĢŁİŁĬĞĻHĤçĢIJŁĮĻ

        fi RHĻŁĹĮ ĻĤH ĤĞĻĞ ĞĮĤ ĹHÍŁĬĞĹĨŅĞĻĨIJĮ ĬIJĽĽHĽ
        ĹHĻŁĹĮ ĞĢĢŁĹĞĢÑ

    fi RHĽHĻ ĹĞĹĨĞĠĬHĽ ÌIJĹ ĞĢĢŁİŁĬĞĻHĤ ĞĢĢŁĹĞĢÑ
    ÎHÌ ĮHŃçĴĞĽĽâĽHĬÌãf
        ĽHĬÌgĞĢĢŁİŁĬĞĻHĤçĽŁİ ł fl
        ĽHĬÌgĞĢĢŁİŁĬĞĻHĤçĢIJŁĮĻ ł fl
```

```
ƒ AĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ ÌIJĹ ĢĬĞĹĹĨÌÌĨĢĞĻĨIJĮ İIJĤĦĬ
ĢĬĞĹĹ AĢĢŁĹĞĢÑçCĞĻĦÍIJĹĨĢĞĬâAĢĢŁĹĞĢÑãƒ

    ƒ NIJ ĨĮĨĻĨĞĬĨŅĞĻĨIJĮ ĨĹ ĮĦĦĤĦĤ
    ĤĦÌ ĨĮĨĻâĹĦĬÌd Ñãƒ
        ĴĞĹĹ

    ƒ CIJİĴĞĹĦĹ ĴĹĦĤĨĢĻĨIJĮĹ ĻIJ ĻĤĦ ÍĹIJŁĮĤ ĻĹŁĻĤ ŁĞĬŁĦĹ
    ĤĦÌ ĢIJİĴĞĹĦâĹĦĬÌd ĴĹĦĤĨĢĻĨIJĮĹd Ñãƒ
        ĨÌ ĬĦĮâÑgĹĤĞĴĦã łł ğƒ
            Ñ ł ĮĴgĞĹÍİĞŇâÑd ĞŇĨĹłĝã
        ĹĦĻŁĹĮ ĴĹĦĤĨĢĻĨIJĮĹ łł Ñ


ƒ AĢĢŁĹĞĢÑ ĢĞĬĢŁĬĞĻĨIJĮ ÌIJĹ ĹĦÍĹĦĹĹĨIJĮ İIJĤĦĬ
ĢĬĞĹĹ AĢĢŁĹĞĢÑçRĦÍĹĦĹĹĨIJĮâAĢĢŁĹĞĢÑãƒ

    ĤĦÌ ççĨĮĨĻççâĹĦĬÌãƒ
        ƒ CĹĦĞĻĦ ĴĹĦĢĨĹĨIJĮ ĴĹIJĴĦĹĻÑ
        ĹĦĬÌgĴĹĦĢĨĹĨIJĮ ł NIJĮĦ

    ƒ CĞĬĢŁĬĞĻĦĹ ĴĹĦĢĨĹĨIJĮ ŁĞĬŁĦ
    ƒ ĠĞĹĦĤ IJĮ ĴĞĹĹĦĤ ĨĮ ÍĹIJŁĮĤ ĻĹŁĻĤ ŁĞĬŁĦĹ
    ĤĦÌ ĨĮĨĻâĹĦĬÌd Ñd ĹĦĨĮĨĻłFĞĬĹĦãƒ
        ĨÌ ĹĦĬÌgĴĹĦĢĨĹĨIJĮ ĨĹ NIJĮĦ IJĹ ĹĦĨĮĨĻƒ
            ĹĦĬÌgĴĹĦĢĨĹĨIJĮ ł ĮĴgĹĻĤâÑã w ğĥƒl

    ƒ CIJİĴĞĹĦĹ ĴĹĦĤĨĢĻĨIJĮĹ ĻIJ ĻĤĦ ÍĹIJŁĮĤ ĻĹŁĻĤ ŁĞĬŁĦĹ
    ĤĦÌ ĢIJİĴĞĹĦâĹĦĬÌd ĴĹĦĤĨĢĻĨIJĮĹd Ñãƒ
        ĹĦĻŁĹĮ ĮĴgĞĠĹIJĬŁĻĦâĴĹĦĤĨĢĻĨIJĮĹ h Ñã l· ĹĦĬÌgĴĹĦĢĨĹĨIJĮ


ƒ MIJĤĦĬ ĢĬĞĹĹ
ĢĬĞĹĹ MIJĤĦĬƒ

    ĤĦÌ ççĨĮĨĻççâĹĦĬÌãƒ
        ƒ CĹĦĞĻĦ Ğ ĬĨĹĻ IJÌ ĮĦĻŃIJĹĨ IJĠÏĦĢĻĹ
        ĹĦĬÌgĬĞÑĦĹĹ ł äã
        ƒ SIJÌĻİĞÑ ĢĬĞĹĹĨÌÌĨĦĹďĹ IJŁĻĴŁĻ IJĠÏĦĢĻ
        ĹĦĬÌgĹIJÌĻİĞÑçĢĬĞĹĹĨÌÌĨĦĹçIJŁĻĴŁĻ ł NIJĮĦ

    ƒ AĤĤ IJĠÏĦĢĻĹ ĻIJ ĻĤĦ İIJĤĦĬ
    ĤĦÌ ĞĤĤâĹĦĬÌd ĬĞÑĦĹãƒ
        ĹĦĬÌgĬĞÑĦĹĹgĞĴĴĦĮĤâĬĞÑĦĹã
```

```
ƒ SĦĻ ĬIJĹĹd IJĴĻĨİĨŅĦĹ ĞĮĤ ĞĢĢŁĹĞĢÑ
ÂĦÌ ĹĦĻâĹĦĬÌd ǻd ĬIJĹĹłNIJĮĦd IJĴĻĨİĨŅĦĹłNIJĮĦd ĞĢĢŁĹĞĢÑłNIJĮĦãƒ

    ĨÌ ĬIJĹĹ ĨĹ ĮIJĻ NIJĮĦƒ
        ĹĦĬÌgĬIJĹĹ ł ĬIJĹĹ

    ĨÌ IJĴĻĨİĨŅĦĹ ĨĹ ĮIJĻ NIJĮĦƒ
        ĹĦĬÌgIJĴĻĨİĨŅĦĹ ł IJĴĻĨİĨŅĦĹ

    ĨÌ ĞĢĢŁĹĞĢÑ ĨĹ ĮIJĻ NIJĮĦƒ
        ĹĦĬÌgĞĢĢŁĹĞĢÑ ł ĞĢĢŁĹĞĢÑ

ƒ FĨĮĞĬĨŅĦ ĻĨĦ İIJĤĦĬ
ÂĦÌ ÌĨĮĞĬĨŅĦâĹĦĬÌãƒ

    ƒ CĹĦĞĻĦ ĞĮĤ ĹĦĻ ĻĨĦ ĨĮĴŁĻ ĬĞÑĦĹ
    ĹĦĬÌgĨĮĴŁĻçĬĞÑĦĹ ł LĞÑĦĹçIĮĴŁĻâã

    ƒ CIJŁĮĻ ĞĬĬ ĻĨĦ IJĠÏĦĢĻĹ
    ĬĞÑĦĹçĢIJŁĮĻ ł ĬĦĮâĹĦĬÌgĬĞÑĦĹĹã

    ƒ IĮĨĻĨĞĬĨŅĦ Ğ ĬĨĹĻ ĢIJĮĻĞĨĮĨĮÍ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĹƒ
    ĹĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĹ ł äã

    ƒ IĮĦĹĞĻĦ ĻĨĦ IJĠÏĦĢĻĹ
    ÌIJĹ Ĩ ĨĮ ĹĞĮÍĦâĬĞÑĦĹçĢIJŁĮĻãƒ

        ƒ IÌ ĨĻďĹ ĻĨĦ ÌĨĹĹĻ ĬĞÑĦĹd
        ƒ ĻĨĦ ĴĹĦĹĨIJŁĹ ĬĞÑĦĹ IJĠÏĦĢĻ ĨĹ ĻĨĦ ĨĮĴŁĻ ĬĞÑĦĹ
        ĨÌ Ĩ łł ƒlƒ
            ĹĦĬÌgĬĞÑĦĹĹäĨãgĴĹĦĹ ł ĹĦĬÌgĨĮĴŁĻçĬĞÑĦĹ
            ĹĦĬÌgĬĞÑĦĹĹäĨãgĮĦŇĻ ł ĹĦĬÌgĬĞÑĦĹĹäĨкĝã

        ƒ AĬĬ ĬĞÑĦĹĹ ĦŇĢĦĴĻ ÌIJĹ ĻĨĦ ÌĨĹĹĻ ĞĮĤ ĻĨĦ ĬĞĹĻ
        ĦĬĨÌ Ĩ l· ĬĞÑĦĹçĢIJŁĮĻ h ĝƒ
            ĹĦĬÌgĬĞÑĦĹĹäĨãgĴĹĦĹ ł ĹĦĬÌgĬĞÑĦĹĹäĨhĝã
            ĹĦĬÌgĬĞÑĦĹĹäĨãgĮĦŇĻ ł ĹĦĬÌgĬĞÑĦĹĹäĨкĝã

        ƒ TĨĦ ĬĞĹĻ ĬĞÑĦĹ h ĻĨĦ ĮĦŇĻ IJĠÏĦĢĻ ĨĹ ĻĨĦ ĬIJĹĹ
        ƒ AĬĹIJ ĬĦĻďĹ ĹĞĹĦ ĞĹĨÂĦ ĻĨĦ ĹĦÌĦĹĦĮĢĦ ĻIJ ĻĨĦ ĬĞĹĻ IJĠÏĦĢĻ
        ƒ ŃĨIJĹĦ IJŁĻĴŁĻ ĨĹ ĻĨĦ İIJĤĦĬďĹ IJŁĻĴŁĻ
        ĦĬĹĦƒ
            ĹĦĬÌgĬĞÑĦĹĹäĨãgĴĹĦĹ ł ĹĦĬÌgĬĞÑĦĹĹäĨhĝã
            ĹĦĬÌgĬĞÑĦĹĹäĨãgĮĦŇĻ ł ĹĦĬÌgĬIJĹĹ
            ĹĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨĹĞĻĨIJĮ ł ĹĦĬÌgĬĞÑĦĹĹäĨã
```

```
        # IÌ ĬĞÑĦĹ ǦIJĮĻĞĨĮĹ ĞĮ ĞĻĻĹĨĠŁĻĦ ǦĞĬĬĦÂ ðŃĦĨÍÎĻĹðd
        # ĨĻďĹ Ğ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹ h
        # ĞÂÂ ĨĻ ĻIJ ĻÎĦ ĬĨĹĻ IJÌ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĹ
        # WĦ ÂIJĮďĻ ĮĦĦÂ ĻIJ ǦÎĦǦĨ ÌIJĹ ĠĨĞĹĦĹ h
        # ǦÎĦǦĨĨĮÍ ÌIJĹ ŃĦĨÍÎĻĹ ĨĹ ĦĮIJŁÍÎ
        ĨÌ ÎĞĹĞĻĻĹâĹĦĬÌgĬĞÑĦĹĹäĨād dŃĦĨÍÎĻĹďãf
            ĹĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĹgĞĴĴĦĮÂâĹĦĬÌgĬĞÑĦĹĹäĨāã

        # UĴÂĞĻĦ ĬIJĹĹ IJĠĬĦǦĻ ŃĨĻÎ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĹ
        ĨÌ ĹĦĬÌgĬIJĹĹ ĨĹ ĮIJĻ NIJĮĦf
            ĹĦĬÌgĬIJĹĹgĹĦİĦİĠĦĹçĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĹâ
                ĹĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĹ
            ã

    # IÌ IJŁĻĴŁĻ ĞǦĻĨĹĞĻĨIJĮ ĨĹ SIJÌĻİĞŃ ĞĮÂ
    # ĬIJĹĹ ÌŁĮǦĻĨIJĮ ĨĹ CĞĻĦÍIJĹĨǦĞĬ CĹIJĹĹhEĮĻĹIJĴÑ
    # ǦĹĦĞĻĦ ĞĮ IJĠĬĦǦĻ IJÌ ǦIJİĠĨĮĦÂ ĞǦĻĨĹĞĻĨIJĮ
    # ĞĮÂ ĬIJĹĹ ÌŁĮǦĻĨIJĮ GIJĮĻĞĨĮĨĮÍ
    # ÌĞĹĻĦĹ ÍĹĞÂĨĦĮĻ ǦĞĬǦŁĬĞĻĨIJĮ
    ĨÌ ĨĹĨĮĹĻĞĮǦĦâĹĦĬÌgĬĞÑĦĹĹähĝād AǦĻĨĹĞĻĨIJĮçSIJÌĻİĞÑã ĞĮÂ ċ
       ĨĹĨĮĹĻĞĮǦĦâĹĦĬÌgĬIJĹĹd LIJĹĹçCĞĻĦÍIJĹĨǦĞĬCĹIJĹĹĦĮĻĹIJĴÑãf
        # CĹĦĞĻĦ ĞĮ IJĠĬĦǦĻ IJÌ ǦIJİĠĨĮĦÂ ĞǦĻĨĹĞĻĨIJĮ
        # ĞĮÂ ĬIJĹĹ ÌŁĮǦĻĨIJĮĹ
        ĹĦĬÌgĹIJÌĻİĞÑçǦĬĞĹĹĨÌĨĦĹçIJŁĻĴŁĻ ł ċ
            AǦĻĨĹĞĻĨIJĮçSIJÌĻİĞÑçLIJĹĹçCĞĻĦÍIJĹĨǦĞĬCĹIJĹĹĦĮĻĹIJĴÑâã

# TĹĞĨĮ ĻÎĦ İIJÂĦĬ
ÂĦÌ ĻĹĞĨĮâĹĦĬÌd Xd Ñd ấd ĦĴIJǦÎĹłĝd ĠĞĻǦÎçĹĨÑĦłNIJĮĦd
          ĴĹĨĮçĦĹĦĹÑłĝd ĹĞĬĨÂĞĻĨIJĮçÂĞĻĞłNIJĮĦãf

    # IĮĨĻĨĞĬĨÑĦ ĞǦǦŁĹĞǦÑ IJĠĬĦǦĻ
    ĹĦĬÌgĞǦǦŁĹĞǦÑgĨĮĨĻâÑã

    # DĦÌĞŁĬĻ ĹĞĬŁĦ ĨÌ ĠĞĻǦÎ ĹĨÑĦ ĨĹ ĮIJĻ ĠĦĨĮÍ ĹĦĻ
    ĻĹĞĨĮçĹĻĦĴĹ ł ĝ

    # CĞĬǦŁĬĞĻĦ ĮŁİĠĦĹ IJÌ ĹĻĦĴĹ
    ĨÌ ĠĞĻǦÎçĹĨÑĦ ĨĹ ĮIJĻ NIJĮĦf
        ĻĹĞĨĮçĹĻĦĴĹ ł ĬĦĮâXã ww ĠĞĻǦÎçĹĨÑĦ
        # DĨĹĨÂĨĮÍ ĹIJŁĮÂĹ ÂIJŃĮg IÌ ĻÎĦĹĦ ĞĹĦ ĹIJİĦ ĹĦİĞĨĮĨĮÍ
        # ÂĞĻĞ ĠŁĻ ĮIJĻ Ğ ÌŁĬĬ ĠĞĻǦÎd ĻÎĨĹ ŃIJĮďĻ ĨĮǦĬŁÂĦ ĨĻ
        # AÂÂ ẀĝẀ ĻIJ ĨĮǦĬŁÂĦ ĻÎĨĹ ĮIJĻ ÌŁĬĬ ĠĞĻǦÎ
        ĨÌ ĻĹĞĨĮçĹĻĦĴĹ ấ ĠĞĻǦÎçĹĨÑĦ l· ĬĦĮâXãf
            ĻĹĞĨĮçĹĻĦĴĹ κł ĝ
```

```
ƒ MĞĨĮ ĻĹĞĨĮĨĮÍ ĬIJIJĴ
ÌIJĹ ĦĴIJĢÎ ĨĮ ĹĞĮÍĦâĝd ĦĴIJĢÎĽĸĝãf

    ƒ PĹĨĮĻ ĦĴIJĢÎ ĮŁİĠĦĹ
    ĴĹĨĮĻâÌďĦĴIJĢÎf ăĦĴIJĢÎåďã

    ƒ RĦĽĦĻ ĞĢĢŁİŁĬĞĻĦĤ ŁĞĬŁĦĽ ĨĮ ĬIJĽĽ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠÌĦĢĻĽ
    ĽĦĬÌgĬIJĽĽgĮĦŃçĴĞĽĽâã
    ĽĦĬÌgĞĢĢŁĹĞĢÑgĮĦŃçĴĞĽĽâã

    ƒ IĻĦĹĞĻĦ IJŁĦĹ ĽĻĦĴĽ
    ÌIJĹ ĽĻĦĴ ĨĮ ĹĞĮÍĦâĻĹĞĨĮçĽĻĦĴĽãf

        ƒ IÌ ĠĞĻĢÎ ĽĨĢĦ ĨĽ ĮIJĻ ĽĦĻ h
        ƒ ĻĹĞĨĮ ŁĽĨĮÍ IJĮĦ ĽĻĦĴ ĞĮĤ ÌŁĬĬ ĤĞĻĞĽĦĻ
        ĨÌ ĠĞĻĢÎçĽĨĢĦ ĨĽ NIJĮĦf
            ĠĞĻĢÎçX ł X
            ĠĞĻĢÎçÑ ł Ñ

        ƒ OĻÎĦĹŃĨĽĦ ĽĬĨĢĦ ĞĠĞĻĢÎ
        ĦĬĽĦf
            ĠĞĻĢÎçX ł XäĽĻĦĴåĠĞĻĢÎçĽĨĢĦfâĽĻĦĴĸĝãåĠĞĻĢÎçĽĨĢĦã
            ĠĞĻĢÎçÑ ł ÑäĽĻĦĴåĠĞĻĢÎçĽĨĢĦfâĽĻĦĴĸĝãåĠĞĻĢÎçĽĨĢĦã

        ƒ PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹĤ ĴĞĽĽ
        IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹĤâĠĞĻĢÎçXd ĻĹĞĨĮĨĮÍłTĹŁĦã

        ƒ CĞĬĢŁĬĞĻĦ ĬIJĽĽ
        ĤĞĻĞçĬIJĽĽd ĹĦÍŁĬĞĹĨĢĞĻĨIJIçĬIJĽĽ ł ċ
            ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ĠĞĻĢÎçÑd
                                ĨĮĢĬŁĤĦçĹĦÍŁĬĞĹĨĢĞĻĨIJIłTĹŁĦã
        ĬIJĽĽ ł ĤĞĻĞçĬIJĽĽ ĸ ĹĦÍŁĬĞĹĨĢĞĻĨIJIçĬIJĽĽ

        ƒ GĦĻ ĴĹĦĤĨĢĻĨIJIĮĽ ĞĮĤ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĴĹĦĤĨĢĻĨIJIĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻĨŁĞĻĨIJIĮgĴĹĦĤĨĢĻĨIJIĮĽâ
                            IJŁĻĴŁĻã
        ĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦĤĨĢĻĨIJIĮĽd
                                            ĠĞĻĢÎçÑã

        ƒ PĦĹÌIJĹİ ĠĞĢĪŃĞĹĤ ĴĞĽĽ
        ĽĦĬÌgĠĞĢĪŃĞĹĤâIJŁĻĴŁĻd ĠĞĻĢÎçÑã

        ƒ OĴĻĨİĨĨĢĦ âŁĴĤĞĻĦ ĴĞĹĞİĦĻĦĹĽã
        ĽĦĬÌgIJĴĻĨİĨĨĢĦĹgĴĹĦçŁĴĤĞĻĦçĴĞĹĞİĽâã
        ÌIJĹ ĬĞÑĦĹ ĨĮ ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽf
            ĽĦĬÌgIJĴĻĨİĨĨĢĦĹgŁĴĤĞĻĦçĴĞĹĞİĽâĬĞÑĦĹã
        ĽĦĬÌgIJĴĻĨİĨĨĢĦĹgĴIJĽĻçŁĴĤĞĻĦçĴĞĹĞİĽâã
```

```
            ƒ PĹĨĮĻ Ğ ĹŁİİĞĹÑ
            ĨÌ ĮIJĻ ĹĻĦĴ ķ ĴĹĨĮĻçĦŁĦĹÑ IJĹ ĹĻĦĴ łł ĻĹĞĨĮçĹĻĦĴĹ h ĝf
                ĴĹĨĮĻâÌďĹĻĦĴf ăĹĻĦĴåd ď κ
                      ÌďĞĢĢf ăĞĢĢŁĹĞĢÑfgġÌåd ď κ
                      ÌďĬIJĹĹf ăĬIJĹĹfgġÌå âď κ
                      ÌďĤĞĻĞçĬIJĹĹf ăĤĞĻĞçĬIJĹĹfgġÌåd ď κ
                      ÌďĹĦÍçĬIJĹĹf ăĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĹĹfgġÌåãd ď κ
                      ÌďĬĹf ăĹĦĬÌgIJĴĻĨİĨĢĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

        ƒ GĦĻ ĞĮĤ ĴĹĨĮĻ ĦĴIJĢÎ ĬIJĹĹ ĞĮĤ ĞĢĢŁĹĞĢÑ
        ĦĴIJĢÎçĤĞĻĞçĬIJĹĹd ĦĴIJĢÎçĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĹĹ ł ċ
            ĹĦĬÌgĬIJĹĹgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦĤâ
                ĨĮĢĬŁĤĦçĹĦÍŁĬĞĹĨĢĞĻĨIJĮłTĹŁĦã
        ĦĴIJĢÎçĬIJĹĹ ł ĦĴIJĢÎçĤĞĻĞçĬIJĹĹ κ ĦĴIJĢÎçĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĹĹ
        ĦĴIJĢÎçĞĢĢŁĹĞĢÑ ł ĹĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦĤâã

        ĴĹĨĮĻâÌďĻĹĞĨĮĨĮÍd ď κ
              ÌďĞĢĢf ăĦĴIJĢÎçĞĢĢŁĹĞĢÑfgġÌåd ď κ
              ÌďĬIJĹĹf ăĦĴIJĢÎçĬIJĹĹfgġÌå âď κ
              ÌďĤĞĻĞçĬIJĹĹf ăĦĴIJĢÎçĤĞĻĞçĬIJĹĹfgġÌåd ď κ
              ÌďĹĦÍçĬIJĹĹf ăĦĴIJĢÎçĹĦÍŁĬĞĹĨĢĞĻĨIJĮçĬIJĹĹfgġÌåãd ď κ
              ÌďĬĹf ăĹĦĬÌgIJĴĻĨİĨĢĦĹgĢŁĹĹĦĮĻçĬĦĞĹĮĨĮÍçĹĞĻĦåďã

        ƒ IÌ ĻĨĦĹĦ ĨĹ ĻĨĦ ŁĞĬĨĤĞĻĨIJĮ ĤĞĻĞ
        ĨÌ ŁĞĬĨĤĞĻĨIJĮçĤĞĻĞ ĨĹ ĮIJĻ NIJĮĦf

            ƒ EŁĞĬŁĞĻĦ ĻĨĦ İIJĤĦĬf
            ĹĦĬÌgĦŁĞĬŁĞĻĦâáŁĞĬĨĤĞĻĨIJĮçĤĞĻĞd
                          ĠĞĻĢĨçĹĨĢĦłĠĞĻĢĨçĹĨĢĦã

ƒ EŁĞĬŁĞĻĦĹ ĻĨĦ İIJĤĦĬ ŁĹĨĮĹ ĴĞĹĹĦĤ ĨĮ ĤĞĻĞĹĦĻ
ĤĦÌ ĦŁĞĬŁĞĻĦâĹĦĬÌd XçŁĞĬd ÑçŁĞĬd ád ĠĞĻĢĨçĹĨĢĦłNIJĮĦãf

    ƒ DĦÌĞŁĬĻ ŁĞĬŁĦ ĨÌ ĠĞĻĢĨ ĹĨĢĦ ĨĹ ĮIJĻ ĠĦĨĮÍ ĹĦĻ
    ŁĞĬĨĤĞĻĨIJĮçĹĻĦĴĹ ł ĝ

    ƒ CĞĬĢŁĬĞĻĦ ĮŁİĠĦĹ IJÌ ĹĻĦĴĹ
    ĨÌ ĠĞĻĢĨçĹĨĢĦ ĨĹ ĮIJĻ NIJĮĦf
        ŁĞĬĨĤĞĻĨIJĮçĹĻĦĴĹ ł ĬĦĮâXçŁĞĬã ww ĠĞĻĢĨçĹĨĢĦ
        ƒ DĨŁĨĤĨĮÍ ĹIJŁĮĤĹ ĤIJŃĮg IÌ ĻĨĦĹĦ ĞĹĦ ĹIJİĦ ĹĦİĞĨĮĨĮÍ
        ƒ ĤĞĻĞ ĠŁĻ ĮIJĻ Ğ ÌŁĬĬ ĠĞĻĢĨd ĻĨĨĹ ŃIJĮďĻ ĨĮĢĬŁĤĦ ĨĻ
        ƒ AĤĤ ẀĝẀ ĻIJ ĨĮĢĬŁĤĦ ĻĨĨĹ ĮIJĻ ÌŁĬĬ ĠĞĻĢĨ
        ĨÌ ŁĞĬĨĤĞĻĨIJĮçĹĻĦĴĹ á ĠĞĻĢĨçĹĨĢĦ l· ĬĦĮâXçŁĞĬãf
            ŁĞĬĨĤĞĻĨIJĮçĹĻĦĴĹ κł ĝ

    ƒ RĦĹĦĻ ĞĢĢŁİŁĬĞĻĦĤ ŁĞĬŁĦĹ ĨĮ ĬIJĹĹ
    ƒ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĹ
    ĹĦĬÌgĬIJĹĹgĮĦŃçĴĞĹĹâã
    ĹĦĬÌgĞĢĢŁĹĞĢÑgĮĦŃçĴĞĹĹâã
```

```
    ƒ IĻĦĹĞĻĦ IJŁĦĹ ĹĻĦĴĽ
    ÌIJĹ ĽĻĦĴ ÎĮ ĹĞĮÍĦâŁĞĬÎĤĞĻÎIJIçĽĻĦĴĽãf

        ƒ IÌ ĠĞĻĢÎ ĽÎŅĦ ÎĽ ĮIJĻ ĽĦĻ h
        ƒ ĻĹĞÎĮ ŁĽÎĮÍ IJĮĦ ĽĻĦĴ ĞĮÂ ÌŁĬĬ ĤĞĻĞĽĦĻ
        ÎÌ ĠĞĻĢÎçĽÎŅĦ ÎĽ NIJĮĦf
            ĠĞĻĢÎçX ł XçŁĞĬ
            ĠĞĻĢÎçÑ ł ÑçŁĞĬ

        ƒ OĻÎĦĹŃÎĽĦ ĽĬÎĢĦ Ğ ĠĞĻĢÎ
        ĦĬĽĦf
            ĠĞĻĢÎçX ł XçŁĞĬä
                ĽĻĦĴǻĠĞĻĢÎçĽÎŅĦfâĽĻĦĴкĝãǻĠĞĻĢÎçĽÎŅĦ
            ā
            ĠĞĻĢÎçÑ ł ÑçŁĞĬä
                ĽĻĦĴǻĠĞĻĢÎçĽÎŅĦfâĽĻĦĴкĝãǻĠĞĻĢÎçĽÎŅĦ
            ā

        ƒ PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÂ ĴĞĽĽ
        IJŁĻĴŁĻ ł ĽĦĬÌgÌIJĹŃĞĹÂâĠĞĻĢÎçXd ĻĹĞÎĮÎĮÍłFĞĬĽĦã

        ƒ CĞĬĢŁĬĞĻĦ ĻÎĦ ĬIJĽĽ
        ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦâIJŁĻĴŁĻd ĠĞĻĢÎçÑã

        ƒ GĦĻ ĴĹĦÂÎĢĻÎIJĮĽ ĞĮÂ ĢĞĬĢŁĬĞĻĦ ĞĮ ĞĢĢŁĹĞĢÑ
        ĴĹĦÂÎĢĻÎIJĮĽ ł ĽĦĬÌgIJŁĻĴŁĻçĬĞÑĦĹçĞĢĻÎŁĞĻÎIJIgĴĹĦÂÎĢĻÎIJĮĽâ
                            IJŁĻĴŁĻã
        ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦâĴĹĦÂÎĢĻÎIJĮĽd ĠĞĻĢÎçÑã

    ƒ GĦĻ ĞĮÂ ĴĹÎĮĻ ŁĞĬÎÂĞĻÎIJĮ ĬIJĽĽ ĞĮÂ ĞĢĢŁĹĞĢÑ
    ŁĞĬÎÂĞĻÎIJIçĬIJĽĽ ł ĽĦĬÌgĬIJĽĽgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦÂâã
    ŁĞĬÎÂĞĻÎIJIçĞĢĢŁĹĞĢÑ ł ĽĦĬÌgĞĢĢŁĹĞĢÑgĢĞĬĢŁĬĞĻĦçĞĢĢŁİŁĬĞĻĦÂâã

    ƒ PĹÎĮĻ Ğ ĽŁİĬĞĹÑ
    ĴĹÎĮĻâÌďŁĞĬÎÂĞĻÎIJĮd ď к
          ÌďĞĢĢf ăŁĞĬÎÂĞĻÎIJIçĞĢĢŁĹĞĢÑfgġÌåd ď к
          ÌďĬIJĽĽf ăŁĞĬÎÂĞĻÎIJIçĬIJĽĽfgġÌåďã

ƒ PĹĦÂÎĢĻĽ IJĮ ĻÎĦ ĽĞİĴĬĦĽ
ÂĦÌ ĴĹĦÂÎĢĻâĽĦĬÌd Xd ǻd ĠĞĻĢÎçĽÎŅĦłNIJĮĦãf

    ƒ DĦÌĞŁĬĻ ŁĞĬŁĦ ÎÌ ĠĞĻĢÎ ĽÎŅĦ ÎĽ ĮIJĻ ĠĦÎĮÍ ĽĦĻ
    ĴĹĦÂÎĢĻÎIJIçĽĻĦĴĽ ł ĝ

    ƒ CĞĬĢŁĬĞĻĦ ĮŁİĠĦĹ IJÌ ĽĻĦĴĽ
    ÎÌ ĠĞĻĢÎçĽÎŅĦ ÎĽ ĮIJĻ NIJĮĦf
        ĴĹĦÂÎĢĻÎIJIçĽĻĦĴĽ ł ĬĦĮâXã ww ĠĞĻĢÎçĽÎŅĦ
```

```
        ƒ DÕĬ·ĨÂĨĨĮÍ ĹIJŁĮÂĹ ÂIJŃĮg IÌ ĻÎĦĹĦ ĞĹĦ ĹIJİĦ ĹĦİĞĨĮĨĮÍ
        ƒ ÂĞĻĞ ĠŁĻ ĮIJĻ Ğ ÌŁĬĬ ĠĞĻĢÎd ĻÎĨĹ ŃIJĮďĻ ĨĮĢĬŁÂĦ ĨĻ
        ƒ AÂÂ Ẁĝẁ ĻIJ ĨĮĢĬŁÂĦ ĻÎĨĹ ĮIJĻ ÌŁĬĬ ĠĞĻĢÎ
        ĨÌ ĴĹĦÂĨĞĻĨIJĮçĹĻĦĴĹ ǻ ĠĞĻĢÎçĹĨŅĦ l· ĬĦĮâXãf
            ĴĹĦÂĨĞĻĨIJĮçĹĻĦĴĹ κŁ ĝ

    ƒ MIJÂĦĬ IJŁĻĴŁĻĹ
    IJŁĻĴŁĻ Ł äā

    ƒ IĻĦĹĞĻĦ IJL·ĦĹ ĹĻĦĴĹ
    ÌIJĹ ĹĻĦĴ ĨĮ ĹĞĮÍĦâĴĹĦÂĨĞĻĨIJĮçĹĻĦĴĹãf

        ƒ IÌ ĠĞĻĢÎ ĹĨŅĦ ĨĹ ĮIJĻ ĹĦĻ h
        ƒ ĻĹĞĨĮ ŁĹĨĮÍ IJĮĦ ĹĻĦĴ ĞĮÂ ÌŁĬĬ ÂĞĻĞĹĦĻ
        ĨÌ ĠĞĻĢÎçĹĨŅĦ ĨĹ NIJĮĦf
            ĠĞĻĢÎçX Ł X

        ƒ OĻÎĦĹŃĨĹĦ ĹĬĨĢĦ Ğ ĠĞĻĢÎ
        ĦĬĹĦf
            ĠĞĻĢÎçX Ł XäĹĻĦĴǻĠĞĻĢÎçĹĨŅĦfâĹĻĦĴκĝāǻĠĞĻĢÎçĹĨŅĦā

        ƒ PĦĹÌIJĹİ ĻÎĦ ÌIJĹŃĞĹÂ ĴĞĹĹ
        ĠĞĻĢÎçIJŁĻĴŁĻ Ł ĹĦĬÌgÌIJĹŃĞĹÂâĠĞĻĢÎçXd ĻĹĞĨĮĨĮÍŁFĞĬĹĦã

        ƒ AĴĴĦĮÂ ĠĞĻĢÎ ĴĹĦÂĨĞĻĨIJĮ ĻIJ ĻÎĦ ĬĨĹĻ IJÌ ĴĹĦÂĨĞĻĨIJĮĹ
        IJŁĻĴŁĻgĞĴĴĦĮÂâĠĞĻĢÎçIJŁĻĴŁĻã

    ƒ SĻĞĢĪ ĞĮÂ ĹĦĻŁĹĮ ĹĦĹŁĬĻĹ
    ĹĦĻŁĹĮ ĮĴgL·ĹĻĞĢĪâIJŁĻĴŁĻã

ƒ PĦĹÌIJĹİĹ ÌIJĹŃĞĹÂ ĴĞĹĹ
ÂĦÌ ÌIJĹŃĞĹÂâĹĦĬÌd Xd ĻĹĞĨĮĨĮÍãf

    ƒ CĞĬĬ ÌIJĹŃĞĹÂ İĦĻÎIJÂ IJĮ ĻÎĦ ĨĮĴŁĻ ĬĞÑĦĹ
    ƒ ĻÎĨĹ ŃĨĬĬ ĹĦĻ ĻÎĦ IJŁĻĴŁĻ ĴĹIJĴĦĹĻÑ ĻÎĞĻ
    ƒ ĻÎĦ ÌĨĹĹĻ ĬĞÑĦĹ ĨĮ ðĴĹĦL·ð IJĠÏĦĢĻ ĨĹ ĦŃĴĦĢĻĨĮÍ
    ĹĦĬÌgĨĮĴŁĻçĬĞÑĦĹgÌIJĹŃĞĹÂâXd ĻĹĞĨĮĨĮÍã

    ƒ CĞĬĬ ÌIJĹŃĞĹÂ İĦĻÎIJÂ IJÌ ĦL·ĦĹÑ IJĠÏĦĢĻ ĨĮ Ğ ĢÎĞĨĮ
    ƒ PĞĹĹ IJŁĻĴŁĻ IJÌ ĻÎĦ ĴĹĦL·ĨIJŁĹ IJĠÏĦĢĻ ĞĹ Ğ ĴĞĹĞİĦĻĦĹ
    ÌIJĹ ĬĞÑĦĹ ĨĮ ĹĦĬÌgĬĞÑĦĹĹf
        ĬĞÑĦĹgÌIJĹŃĞĹÂâĬĞÑĦĹgĴĹĦL·gIJŁĻĴŁĻd ĻĹĞĨĮĨĮÍã

    ƒ ðĬĞÑĦĹð ĨĹ ĮIJŃ ĻÎĦ ĬĞĹĹ IJĠÏĦĢĻ ÌĹIJİ ĻÎĦ ĬĨĹĻd
    ƒ ĹĦĻŁĹĮ ĨĻĹ IJŁĻĴŁĻ
    ĹĦĻŁĹĮ ĬĞÑĦĹgIJŁĻĴŁĻ
```

```
# PHĹÌIJĹİĽ ĠĞĢĪŃĞĹÎ ĴĞĽĽ
ÎHÌ ĠĞĢĪŃĞĹÎâĽHĬÌd IJŁĻĴŁĻd Ñãf

    # IÌ ĽIJÌĻİĞŇ ĢĬĞĽĽĨÌÌĨHĹ
    ĨÌ ĽHĬÌgĽIJÌĻİĞŇçĢĬĞĽĽĨÌÌĨHĹçIJŁĻĴŁĻ ĨĽ ĮIJĻ NIJĮH f
        # FĨĹĽĻ ĢĞĬĬ ĠĞĢĪŃĞĹÎ İHĻÎIJÎ
        # IJĮ ĻÎH ĢIJİĠĨĮHÎ ĞĢĻĨĽĞĻĨIJĮwĬIJĽĽ
        # ĻÎĨĽ ŃĨĬĬ ĽHĻ ÎĨĮĴŁĻĽ ĴĹIJĴHĹĻŇ
        ĽHĬÌgĽIJÌĻİĞŇçĢĬĞĽĽĨÌÌĨHĹçIJŁĻĴŁĻgĠĞĢĪŃĞĹÎâIJŁĻĴŁĻd Ñã

        # SĨĮĢH ŃHďĬĬ ĮIJĻ ĢĞĬĬ ĠĞĢĪŃĞĹÎ İHĻÎIJÎ IJÌ ĻÎH ĬĞĽĻ ĬĞŇHĹ
        # ŃÎĨĢÎ ĨĽ SIJÌĻİĞŇ ĞĢĻĨĽĞĻĨIJĮ
        # ĞĽ ŃH ŁĽHÎ ĢIJİĠĨĮHÎ ĞĢĻĨĽĞĻĨIJĮwĬIJĽĽ
        # IJĠÏHĢĻd ĬHĻďĽ ĽHĻ ÎĨĮĴŁĻĽ ĨĮ ĻÎĨĽ IJĠÏHĢĻ
        ĽHĬÌgĬĞŇHĹĽähĝāgÎĨĮĴŁĻĽ ł ċ
            ĽHĬÌgĽIJÌĻİĞŇçĢĬĞĽĽĨÌÌĨHĹçIJŁĻĴŁĻgÎĨĮĴŁĻĽ

        # CĞĬĬ ĠĞĢĪŃĞĹÎ İHĻÎIJÎ ÍIJĨĮÍ ĻÎĹIJŁÍÎ
        # ĞĬĬ ĻÎH IJĠÏHĢĻĽ ĠŁĻ ĬĞĽĻ
        # ĨĮ ĹHŁHĹĽHÎ IJĹÎHĹ ĴĞĽĽĨĮÍ ÎĨĮĴŁĻĽ ĞĽ Ğ ĴĞĹĞİHĻHĹ
        ÌIJĹ ĬĞŇHĹ ĨĮ ĹHŁHĹĽHÎâĽHĬÌgĬĞŇHĹĽäfhĝāãf
            ĬĞŇHĹgĠĞĢĪŃĞĹÎâĬĞŇHĹgĮHŇĻgÎĨĮĴŁĻĽã

        ĹHĻŁĹĮ

    # FĨĹĽĻ ĢĞĬĬ ĠĞĢĪŃĞĹÎ İHĻÎIJÎ IJĮ ĻÎH ĬIJĽĽ
    # ĻÎĨĽ ŃĨĬĬ ĽHĻ ÎĨĮĴŁĻĽ ĴĹIJĴHĹĻŇ ĻÎĞĻ ĻÎH ĬĞĽĻ
    # ĬĞŇHĹ ŃĨĬĬ ĻĹŇ ĻIJ ĞĢHĽĽ ĽÎIJĹĻĬŇ
    ĽHĬÌgĬIJĽĽgĠĞĢĪŃĞĹÎâIJŁĻĴŁĻd Ñã

    # CĞĬĬ ĠĞĢĪŃĞĹÎ İHĻÎIJÎ ÍIJĨĮÍ ĻÎĹIJŁÍÎ ĞĬĬ ĻÎH IJĠÏHĢĻĽ
    # ĨĮ ĹHŁHĹĽHÎ IJĹÎHĹ ĴĞĽĽĨĮÍ ÎĨĮĴŁĻĽ ĞĽ Ğ ĴĞĹĞİHĻHĹ
    ÌIJĹ ĬĞŇHĹ ĨĮ ĹHŁHĹĽHÎâĽHĬÌgĬĞŇHĹĽãf
        ĬĞŇHĹgĠĞĢĪŃĞĹÎâĬĞŇHĹgĮHŇĻgÎĨĮĴŁĻĽã

# RHĻĹĨHŁHĽ ĞĮÎ ĹHĻŁĹĮĽ ĴĞĹĞİHĻHĹĽ IJÌ ĻĹĞĨĮĞĠĬH ĬĞŇHĹĽ
ÎHÌ ÍHĻçĴĞĹĞİHĻHĹĽâĽHĬÌãf

    # CĹHĞĻH Ğ ĬĨĽĻ ÌIJĹ ĴĞĹĞİHĻHĹĽ
    ĴĞĹĞİHĻHĹĽ ł ää

    # IĻHĹĞĠĬH ĻĹĞĨĮĞĠĬH ĬĞŇHĹĽ ĞĮÎ ÍHĻ ĻÎHĨĽ ĴĞĹĞİHĻHĹĽ
    ÌIJĹ ĬĞŇHĹ ĨĮ ĽHĬÌgĻĹĞĨĮĞĠĬHçĬĞŇHĹĽf
        ĴĞĹĞİHĻHĹĽgĞĴĴHĮÎâĬĞŇHĹgÍHĻçĴĞĹĞİHĻHĹĽâãã

    # RHĻŁĹĮ Ğ ĬĨĽĻ
    ĹHĻŁĹĮ ĴĞĹĞİHĻHĹĽ
```

```
ƒ UĴĤĞĻĦĽ ĻĨĦ ĬIJĤĦĬ ŃĨĻĨ ĮĦŃ ĴĞĹĞĬĦĻĦĹĽ
ĤĦÌ ĽĦĻçĴĞĹĞĬĦĻĦĹĽâĽĦĬÌd ĴĞĹĞĬĦĻĦĹĽãƒ

    ƒ IĻĦĹĞĻĦ IJĽĦĹ ĻĨĦ ĴĞĹĞĬĦĻĦĹĽ ĞĮĤ ĬĞÑĦĹĽ
    ƒ ĞĮĤ ŁĴĤĞĻĦ ĦĞĢĨ ĬĞÑĦĹĽ ŃĨĻĨ ĦĞĢĨ ĽĦĻ IJÌ ĻĨĦ ĴĞĹĞĬĦĻĦĹĽ
    ÌIJĹ ĴĞĹĞĬĦĻĦĹçĽĦĻd ĬĞÑĦĹ ĨĮ ŅĨĴâĴĞĹĞĬĦĻĦĹĽd
                                        ĽĦĬÌgĻĹĞĨĮĞĠĬĦçĬĞÑĦĹĽãƒ
        ĬĞÑĦĹgĽĦĻçĴĞĹĞĬĦĻĦĹĽâǻĴĞĹĞĬĦĻĦĹçĽĦĻã

ƒ SĞĽĦĽ ĻĨĦ ĴĞĹĞĬĦĻĦĹĽ ĻIJ Ğ ÌĨĬĦ
ĤĦÌ ĽĞĽĦçĴĞĹĞĬĦĻĦĹĽâĽĦĬÌd ĴĞĻĨãƒ

    ƒ OĴĦĮ Ğ ÌĨĬĦ ĨĮ ĻĨĦ ĠĨĮĞĹÑhŃĹĨĻĦ ĬIJĤĦ
    ƒ ĞĮĤ ĽĞĽĦ ĴĞĹĞĬĦĻĦĹĽ ĨĮĻIJ ĨĻ
    ŃĨĻĨ IJĴĦĮâĴĞĻĨd ďŃĠďã ĞĽ Ìƒ
        ĴĨĢĨĬĦgĤŁĬĴâĽĦĬÌgĬĦĻçĴĞĹĞĬĦĻĦĹĽâãd Ìã

ƒ LIJĞĤĽ ĻĨĦ ŃĦĨĬĬĻĽ ĞĮĤ ŁĴĤĞĻĦĽ Ğ ĬIJĤĦĬ ĨĮĽĻĞĮĢĦ ŃĨĻĨ ĻĨĦĬ
ĤĦÌ ĬIJĞĤçĴĞĹĞĬĦĻĦĹĽâĽĦĬÌd ĴĞĻĨãƒ

    ƒ OĴĦĮ ÌĨĬĦ ĨĮ ĻĨĦ ĠĨĮĞĹÑhĹĦĞĤ ĬIJĤĦd
    ƒ ĬIJĞĤ ŃĦĨĬĬĻĽ ĞĮĤ ŁĴĤĞĻĦ ĻĹĞĨĮĞĠĬĦ ĬĞÑĦĹĽ
    ŃĨĻĨ IJĴĦĮâĴĞĻĨd ďĹĠďã ĞĽ Ìƒ
        ĽĦĬÌgĽĦĻçĴĞĹĞĬĦĻĦĹĽâĴĨĢĨĬĦgĬIJĞĤâÌãã

ƒ SĞĽĦĽ ĻĨĦ ĬIJĤĦĬ
ĤĦÌ ĽĞĽĦâĽĦĬÌd ĴĞĻĨãƒ

    ƒ MĞĨĦ Ğ ĤĦĦĴ ĢIJĴÑ IJÌ ĢŁĹĹĦĮĻ ĬIJĤĦĬ ĨĮĽĻĞĮĢĦ
    ĬIJĤĦĬ ł ĢIJĴÑgĤĦĦĴĢIJĴÑâĽĦĬÌã

    ƒ RĦĽĦĻ ĞĢĢŁĬŁĬĞĻĦĤ ĽĞĬŁĦĽ ĨĮ ĬIJĽĽ ĞĮĤ ĞĢĢŁĹĞĢÑ IJĠĬĦĢĻĽ
    ĬIJĤĦĬgĬIJĽĽgĮĦŃçĴĞĽĽâã
    ĬIJĤĦĬgĞĢĢŁĹĞĢÑgĮĦŃçĴĞĽĽâã

    ƒ RĦĬIJĽĦ ĤĞĻĞ ÌĹIJĬ ĻĨĦ ĨĮĴŁĻ ĬĞÑĦĹ
    ƒ ĞĮĤ ÍĹĞĤĨĦĮĻĽ ÌĹIJĬ ĻĨĦ ĬIJĽĽ IJĠĬĦĢĻ
    ĬIJĤĦĬgĨĮĴŁĻçĬĞÑĦĹgççĤĨĢĻççgĴIJĴâďIJŁĻĴŁĻďd NIJĮĦã
    ĬIJĤĦĬgĬIJĽĽgççĤĨĢĻççgĴIJĴâďĤĨĮĴŁĻĽďd NIJĮĦã

    ƒ FIJĹ ĦĞĢĨ ĬĞÑĦĹ ĹĦĬIJĽĦ ĨĮĴŁĻĽd IJŁĻĴŁĻ ĞĮĤ ĤĨĮĴŁĻĽ ĴĹIJĴĦĹĻĨĦĽ
    ÌIJĹ ĬĞÑĦĹ ĨĮ ĬIJĤĦĬgĬĞÑĦĹĽƒ
        ÌIJĹ ĴĹIJĴĦĹĻÑ ĨĮ äďĨĮĴŁĻĽďd ďIJŁĻĴŁĻďd ďĤĨĮĴŁĻĽďd
                            ďĤŃĦĨĬĬĻĽďd ďĤĠĨĞĽĦĽďãƒ
            ĬĞÑĦĹgççĤĨĢĻççgĴIJĴâĴĹIJĴĦĹĻÑd NIJĮĦã

    ƒ OĴĦĮ Ğ ÌĨĬĦ ĨĮ ĻĨĦ ĠĨĮĞĹÑhŃĹĨĻĦ ĬIJĤĦ ĞĮĤ ĽĞĽĦ ĻĨĦ ĬIJĤĦĬ
    ŃĨĻĨ IJĴĦĮâĴĞĻĨd ďŃĠďã ĞĽ Ìƒ
        ĴĨĢĨĬĦgĤŁĬĴâĬIJĤĦĬd Ìã
```

```
    # LIJĞÂĽ ĞĮÂ ĹĦĻŁĹĮĽ Ğ İIJÂĦĬ
    đĽĻĞĻĨĢİĦĻÎIJÂ
    ÂĦÌ ĬIJĞÂâĴĞĻÎãf

        # OĴĦĮ ÌĨĬĦ ĨĮ ĻÎĦ ĠĨĮĞĹÑhĹĦĞÂ İIJÂĦd ĬIJĞÂ Ğ İIJÂĦĬ
        ŃĨĻÎ IJĴĦĮâĴĞĻÎd ďĹĠďã ĞĽ Ìf
            İIJÂĦĬ ł ĴĨĢĨĬĦgĬIJĞÂâÌã

        # RĦĻŁĹĮ Ğ İIJÂĦĬ
        ĹĦĻŁĹĮ İIJÂĦĬ


# LIJĞÂĽ Ğ MNIST ÂĞĻĞĽĦĻ
ÂĦÌ ĬIJĞÂçİĮĨĽĻçÂĞĻĞĽĦĻâÂĞĻĞĽĦĻd ĴĞĻÎãf

    # SĢĞĮ ĞĬĬ ĻÎĦ ÂĨĹĦĢĻIJĹĨĦĽ ĞĮÂ ĢĹĦĞĻĦ Ğ ĬĨĽĻ IJÌ ĬĞĠĦĬĽ
    ĬĞĠĦĬĽ ł IJĽgĬĨĽĻÂĨĹâIJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ÂĞĻĞĽĦĻãã

    # CĹĦĞĻĦ ĬĨĽĻĽ ÌIJĹ ĽĞİĴĬĦĽ ĞĮÂ ĬĞĠĦĬĽ
    X ł äã
    Ñ ł äã

    # FIJĹ ĦĞĢÎ ĬĞĠĦĬ ÌIJĬÂĦĹ
    ÌIJĹ ĬĞĠĦĬ ĨĮ ĬĞĠĦĬĽf
        # AĮÂ ÌIJĹ ĦĞĢÎ ĨİĞÍĦ ĨĮ ÍĨĽĦĮ ÌIJĬÂĦĹ
        ÌIJĹ ÌĨĬĦ ĨĮ IJĽgĬĨĽĻÂĨĹâIJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ÂĞĻĞĽĦĻd ĬĞĠĦĬãã f
            # RĦĞÂ ĻÎĦ ĨİĞÍĦ
            ĨİĞÍĦ ł ĢĽğgĨİĹĦĞÂâ
                        IJĽgĴĞĻÎgÏIJĨĮâĴĞĻÎd ÂĞĻĞĽĦĻd ĬĞĠĦĬd ÌĨĬĦãd
                        ĢĽğgIMREADçUNCHANGEDã

            # AĮÂ ĞĴĴĦĮÂ ĨĻ ĞĮÂ Ğ ĬĞĠĦĬ ĻIJ ĻÎĦ ĬĨĽĻĽ
            XgĞĴĴĦĮÂâĨİĞÍĦã
            ÑgĞĴĴĦĮÂâĬĞĠĦĬã

    # CIJĮĽĦĹĻ ĻÎĦ ÂĞĻĞ ĻIJ ĴĹIJĴĦĹ ĮŁİĴÑ ĞĹĹĞÑĽ ĞĮÂ ĹĦĻŁĹĮ
    ĹĦĻŁĹĮ ĮĴgĞĹĹĞÑâXãd ĮĴgĞĹĹĞÑâÑãgĞĽĻÑĴĦâďŁĨĮĻíďã


# MNIST ÂĞĻĞĽĦĻ âĻĹĞĨĮ κ ĻĦĽĻã
ÂĦÌ ĢĹĦĞĻĦçÂĞĻĞçİĮĨĽĻâĴĞĻÎãf

    # LIJĞÂ ĠIJĻÎ ĽĦĻĽ ĽĦĴĞĹĞĻĦĬÑ
    Xd Ñ ł ĬIJĞÂçİĮĨĽĻçÂĞĻĞĽĦĻâďĻĹĞĨĮďd ĴĞĻÎã
    XçĻĦĽĻd ÑçĻĦĽĻ ł ĬIJĞÂçİĮĨĽĻçÂĞĻĞĽĦĻâďĻĦĽĻďd ĴĞĻÎã

    # AĮÂ ĹĦĻŁĹĮ ĞĬĬ ĻÎĦ ÂĞĻĞ
    ĹĦĻŁĹĮ Xd Ñd XçĻĦĽĻd ÑçĻĦĽĻ
```

```
fi LĞĠHĬ ĨĮÂHŃ ĻIJ ĬĞĠHĬ ĮĞİH ĹHĬĞĻĨIJĮ
ÌĞĹÎĨIJĮçİĮĨĹĻçĬĞĠHĬĹ ł ă
    flf ďThĹÎĨĹĻwĻIJĴďd
    ĝf ďTĹIJŁĹHĹďd
    ğf ďPŁĬĬIJL·HĹďd
    ġf ďDĹHĹĹďd
    ģf ďCIJĞĻďd
    ĥf ďSĞĮÂĞĬďd
    ħf ďSÎĨĹĻďd
    ìf ďSĮHĞĨHĹďd
    íf ďBĞÍďd
    îf ďAĮĪĬH ĠIJIJĻď
å

fi RHĞÂ ĞĮ ĨİĞÍH
ĨİĞÍHçÂĞĻĞ ł ĢL·ğgĨİĹHĞÂâďĴĞĮĻĹgĴĮÍďd ĢL·ğgIMREADçGRAYSCALEã

fi RHĹĨŅH ĻIJ ĻÎH ĹĞİH ĹĨŅH ĞĹ FĞĹÎĨIJĮ MNIST ĨİĞÍHĹ
ĨİĞÍHçÂĞĻĞ ł ĢL·ğgĹHĹĨŅHâĨİĞÍHçÂĞĻĞd âğíd ğíãã

fi IĮL·HĹĻ ĨİĞÍH ĢIJĬIJĹĹ
ĨİĞÍHçÂĞĻĞ ł ğĥĥ h ĨİĞÍHçÂĞĻĞ

fi RHĹÎĞĴH ĞĮÂ ĹĢĞĬH ĴĨŃHĬ ÂĞĻĞ
ĨİĞÍHçÂĞĻĞ ł âĨİĞÍHçÂĞĻĞgĹHĹÎĞĴHâĝd hĝãgĞĹĻŃĴHâĮĴgÌĬIJĞĻġğã h
                ĝğìgĥã w ĝğìgĥ

fi LIJĞÂ ĻÎH İIJÂHĬ
İIJÂHĬ ł MIJÂHĬgĬIJĞÂâďÌĞĹÎĨIJĮçİĮĨĹĻgİIJÂHĬďã

fi PĹHÂĨĢĻ IJĮ ĻÎH ĨİĞÍH
ĢIJĮÌĨÂHĮĢHĹ ł İIJÂHĬgĴĹHÂĨĢĻâĨİĞÍHçÂĞĻĞã

fi GHĻ ĴĹHÂĨĢĻĨIJĮ ĨĮĹĻHĞÂ IJÌ ĢIJĮÌĨÂHĮĢH ĬHL·HĬĹ
ĴĹHÂĨĢĻĨIJĮĹ ł İIJÂHĬgIJŁĻĴŁĻçĬĞÑHĹçĞĢĻĨL·ĞĻĨIJĮgĴĹHÂĨĢĻĨIJĮĹâĢIJĮÌĨÂHĮĢHĹã

fi GHĻ ĬĞĠHĬ ĮĞİH ÌĹIJİ ĬĞĠHĬ ĨĮÂHŃ
ĴĹHÂĨĢĻĨIJĮ ł ÌĞĹÎĨIJĮçİĮĨĹĻçĬĞĠHĬĹäĴĹHÂĨĢĻĨIJĮĹäflāā

ĴĹĨĮĻâĴĹHÂĨĢĻĨIJĮã
```