|  |
| --- |
| Deep Learning Handbook |
| *Tips & tricks gathered from various courses and books* |

|  |
| --- |
| keivan ipchi Hagh  10-9-2020 |

This is a fast-paced document I wrote for myself to remember the important steps and save some time when dealing with a new machine learning problem. The goal of the document is not to teach something new, but rather to have a step-by-step guideline on how the process is and how to approach the problem as well as some tricks and alternative options which might appear useful. Some contents of this documents might be faulty as I am not a professional in the field (Yet!). therefore, if you notice any bugs or have any suggestions or questions, feel free to contact me at [ipchi1380@gmail.com](mailto:ipchi1380@gmail.com). I’ll mention a few good links I found when learning about some of the concepts as well. Make sure to check them out.

Table of Contents

[Useful links 4](#_Toc53165323)

[Roadmaps 4](#_Toc53165324)

[Courses and books 4](#_Toc53165325)

[Data set Repositories 4](#_Toc53165326)

[Machine Learning projects 4](#_Toc53165327)

[Competitions 4](#_Toc53165328)

[Quick overview of the process 5](#_Toc53165329)

[Solving a Machine Learning problem 5](#_Toc53165330)

[1. Understanding the problem 5](#_Toc53165331)

[2. Importing the necessary libraries & packages 6](#_Toc53165332)

[ Keras 6](#_Toc53165333)

[ NumPy 6](#_Toc53165334)

[ Pandas 6](#_Toc53165335)

[ Matplotlib - pyplot 6](#_Toc53165336)

[3. Preparing (pre-processing) your data 6](#_Toc53165337)

[ One-hot encoding (NLP) 6](#_Toc53165338)

[ Normalizing the data 6](#_Toc53165339)

[ Vectorization 6](#_Toc53165340)

[ Handling missing values 7](#_Toc53165341)

[ General tips 7](#_Toc53165342)

[4. Building your model 7](#_Toc53165343)

[ Loss function (aka. Objective function) 7](#_Toc53165344)

[ Optimizer 7](#_Toc53165345)

[ Activation functions 7](#_Toc53165346)

[5. Training your model 8](#_Toc53165347)

[ Splitting data & Validation sets 8](#_Toc53165348)

[ Setting batches 8](#_Toc53165349)

[ K-fold validation 9](#_Toc53165350)

[ Overfitting & Underfitting 9](#_Toc53165351)

\_ Blank Page \_

**Note: I don’t own any of the contents in this document. Credits have been mentioned!**

# Useful links

## Roadmaps

* [Machine Learning roadmap 2020](https://whimsical.com/CA7f3ykvXpnJ9Az32vYXva) – An awesome roadmap made by [*Daniel Bourke*](https://www.mrdbourke.com/)

## Courses and books

* [Deep Learning with Python](https://www.google.com/search?q=deep+learning+with+python&rlz=1C1RLNS_enIR790IR790&oq=Deep+Learning+with+p&aqs=chrome.0.0j46l2j69i57j0l3j69i61.3949j0j1&sourceid=chrome&ie=UTF-8) – Most of this document is pieces of this book. This book gets you up and running.
* [Machine Learning Crash Course](https://developers.google.com/machine-learning/crash-course) – This course is made by Google and provides lots of information in the matter.
* [Introduction to Machine Learning](https://classroom.udacity.com/courses/ud187) – Free course provided by Udacity
* [Kaggle courses](https://www.kaggle.com/) – Free fast-paced courses

## Data set Repositories

* [UCI Machine Learning Repository](http://archive.ics.uci.edu/ml/index.php) – 557 data sets provided by UC Irvine
* [U.S. Government’s Open Data](https://www.data.gov/) – Over 200,000 datasets!
* [Kaggle Repositories](https://www.kaggle.com/) – Hundreds of datasets

## Machine Learning projects

* [47 Machine Learning projects 2020](https://data-flair.training/blogs/machine-learning-project-ideas/) – 47 projects with included datasets and source

## Competitions

* [Kaggle competitions](https://www.kaggle.com/) – Machine Learning competitions

# Quick overview of the process

The workflow is pretty simple:

* **Gather raw data**
* **Do** [***Feature Engineering***](https://en.wikipedia.org/wiki/Feature_engineering#:~:text=Feature%20engineering%20is%20the%20process,as%20applied%20machine%20learning%20itself.). This is where you transform the raw data into a more useful shape or extract useful information that might require some digging. Basically preparing your data.
* **Make a model**. This consists of making the model and adding the required layers and finally compiling it. You have to define a [*Loss Function*](https://en.wikipedia.org/wiki/Loss_function)and an [*Optimizer Function*](https://keras.io/api/optimizers/) in this step.
* **Train the model**. In this step, you train the network.
* **Evaluate the model**. Time to see how well the network is doing by evaluating the metrics and results captured from training & evaluating.

# Solving a Machine Learning problem

## Understanding the problem

First thing first, you must find out which type of problem are you dealing with. This is basically because they implementations are varied across different machine learning problems as well as some of the techniques. In the table below, you can see examples of common supervised and unsupervised ML problems:

|  |  |  |
| --- | --- | --- |
| **Type of ML Problem** | **Description** | **Example** |
| Classification | Pick one of N labels | Cat, dog, horse, or bear |
| Regression | Predict numerical values | Click-through rate |
| Clustering | Group similar examples | Most relevant documents (unsupervised) |
| Association rule learning | Infer likely association patterns in data | If you buy hamburger buns, you're likely to buy hamburgers (unsupervised) |
| Structured output | Create complex output | Natural language parse trees, image recognition bounding boxes |
| Ranking | Identify position on a scale or status | Search result ranking |

## Importing the necessary libraries & packages

These are commonly used libraries for solving all sorts of ML problems:

### Keras

TensorFlow’s most common high API open-source library

### [NumPy](https://numpy.org/)

NumPy is mainly used for its multi-dimensional arrays and store procedures

### [Pandas](https://pandas.pydata.org/)

Pandas is used for storing and manipulating large data sets

### [Matplotlib](https://matplotlib.org/) - pyplot

We use pyplot for drawing statistics and creating a visual representation of the data

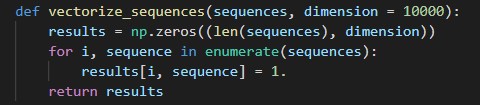
## Preparing (pre-processing) your data

This is the most important step as you are preparing the nutrition for your model. You will be filtering (Basically cleaning up) your data, converting it into a more useful shape for your model and removing the irrelevant rows or columns. There are a few techniques you could use to deal with raw data:

### One-hot encoding (NLP)

Sequences of words can be encoded as binary vectors. One-hot encode your lists to turn them into vectors of 0s and 1s. This would mean, for instance, turning the sequence [3, 5] into a 10,000-dimensional vector that would be all 0s except for indices 3 and 5, which would be 1s. Then you could use as the first layer in your network a Dense layer, capable of handling floating-point vector data.

Here is how you might want to implement this technique:

****

### Normalizing the data

The smaller the inputs, the better for the model. You should try reducing the noise in your data. One technique is as followed which reduces the values though keeping the scale:

* + Calculate the mean value
  + Subtract the mean from actual values
  + Calculate the standard deviation value
  + Divide the results by the STD

### Vectorization

All inputs and targets in a neural network must be tensors of floating-point data (or, in specific cases, tensors of integers). Whatever data you need to process—sound, images, text—you must first turn into tensors, a step called data vectorization.

### Handling missing values

You may sometimes have missing values in your data. In general, with neural networks, it’s safe to input missing values as 0, with the condition that 0 isn’t already a meaningful value. The network will learn from exposure to the data that the value 0 means missing data and will start ignoring the value.

### General tips

* + Cleaning data redundancy – Removing duplicate data
  + Shuffling the data – Note: Sequence of data might be of importance in some cases!

## Building your model

Once the network architecture is defined, you have to choose two more things for your model:

### Loss function (aka. Objective function)

The quantity that will be minimized during training. It represents a measure of success for the task at hand.

When it comes to common problems, there are simple guidelines you can follow to choose:

|  |  |  |
| --- | --- | --- |
| **Type of ML Problem** | **Common Loss function** | **Last layer’s Activation Function** |
| Binary Classification | Binary Crossentropy | Sigmoid |
| multi-Class Classification | Categorical Crossentropy | Softmax |
| Regression | Mean-Squared-Error (MSE) | Sigmoid |
| Sequence-Learning | Connectionist Temporal Classification (CTC) | ? |

### Optimizer

Determines how the network will be updated based on the loss function. It implements a specific variant of stochastic gradient descent (SGD).

Here are a few optimizers:

#### [RMSProp](https://keras.io/api/optimizers/rmsprop/)

The RMSProp optimizer is generally a good enough choice, whatever your problem. That’s one less thing for you to worry about.

### Activation functions

These functions determine what is fired into the next neuron which provides some sort of filtering to our layers. Without an activation function, the layers consist of two linear operations which forces the layer to learn only linear transformations of the data. In order to get access to a much richer hypothesis space that would benefit from

deep representations, you need a non-linearity, or activation function. Here are a few common functions:

#### [Sigmoid](https://en.wikipedia.org/wiki/Sigmoid_function)

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve which scales your input between 0 & 1.

#### [Relu](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

the rectifier is an activation function defined as the positive part of its argument: where x is the input to a neuron.

#### [Softmax](https://en.wikipedia.org/wiki/Softmax_function)

The Softmax function takes as input a vector z of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.

## Training your model

In this step, you start training (Also called *fitting*) your model using the training data. There are a few things to keep in mind while doing so:

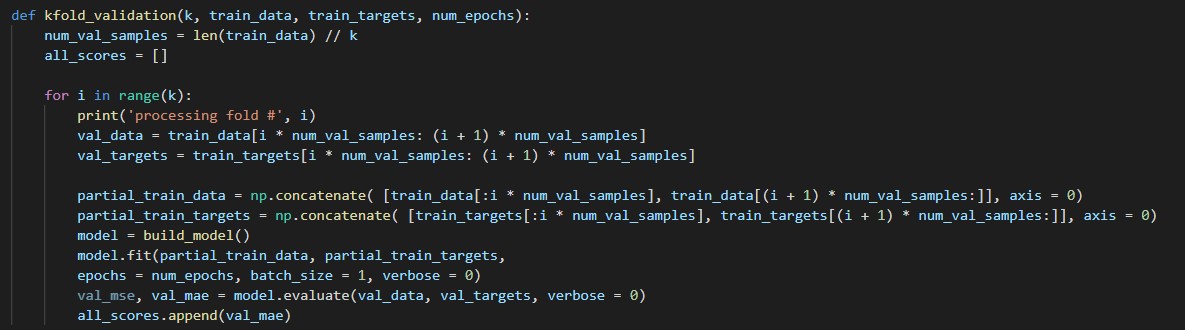
### Splitting data & Validation sets

You must split your data into two main categories: *Training* data (80%) and *Test* data (20%). You don’t want to test your model on the data it has been trained on! This is a fine way, however, [*Validation set*](https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets) could come to your aid. This is an important concept that you’ll need when training a model. This splits partial of the *training* data for evaluation purposes. This helps you keep an eye on how the model is doing in the process of training.

### Setting batches

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Note that smaller batch size leads to higher accuracy but in cost of longer training time. On the other hand, higher noise can help it jump out of a bad local minimum, rather than leaving it stuck in it.

### K-fold validation

This consists of splitting the available data into K partitions (typically K = 4 or 5), instantiating K identical models, and training each one on K – 1 partition while evaluating on the remaining partition. The validation score for the model used is then the average of the K validation scores obtained. In terms of code, this is straightforward.

### Overfitting & Underfitting

Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. Now how to prevent (Minimize) this:

#### Reducing network’s size

The simplest way to prevent overfitting is to reduce the size of the model: the number of learnable parameters in the model. Intuitively, a model with more parameters has more memorization capacity and therefore can easily learn a perfect dictionary-like mapping between training samples and their targets—a mapping without any generalization power.

On the other hand, if the network has limited memorization resources, it won’t be

able to learn this mapping as easily; thus, in order to minimize its loss, it will have to

resort to learning compressed representations that have predictive power regarding

the targets—precisely the type of representations we’re interested in. At the same

time, keep in mind that you should use models that have enough parameters that they don’t underfit: your model shouldn’t be starved for memorization resources.

#### Adding weight regularizations

a common way to mitigate overfitting is to put constraints on the complexity

of a network by forcing its weights to take only small values, which makes the distribution of weight values more regular. This is called *weight regularization*. This cost comes in two flavors:

* L1 regularization—The cost added is proportional to the absolute value of the weight coefficients (the L1 norm of the weights).



* L2 regularization—The cost added is proportional to the square of the value of the weight coefficients (the L2 norm of the weights). L2 regularization is also called weight decay in the context of neural networks. Don’t let the different name confuse you: weight decay is mathematically the same as L2 regularization.

#### Adding dropouts

Dropout is one of the most effective and most commonly used regularization techniques for neural networks. Let’s say a given layer would normally return a vector [0.2, 0.5, 1.3, 0.8, 1.1] for a given input sample during training. After applying dropout, this vector will have a few zero entries distributed at random: for example, [0, 0.5, 1.3, 0, 1.1]. The dropout rate is the fraction of the features that are zeroed out; it’s usually set between 0.2 and 0.5.