Deep Learning & AI

Setting up the environment

Jannis Seemann

Installing the required tools

- We will install Python and Visual Studio Code on your system
- After that, we will configure the course materials
- And install PyTorch through a Python Virtual Environment

Deep Learning & AI

Setting up the environment (Linux)

Jannis Seemann

Installing the required tools (Linux)

- We will check the version of Python... and install Visual Studio Code on your system
- After that, we will configure the course materials
- And install PyTorch through a Python Virtual Environment

Deep Learning & AI

Setting up the environment (macOS)

Jannis Seemann

Installing the required tools (macOS)

- We will install Python and Visual Studio Code on your system
- After that, we will configure the course materials
- And install PyTorch through a Python Virtual Environment

Deep Learning & AI

Executing a file

Deep Learning & AI

What is a model?

Jannis Seemann

What is a model?

- **Training data:**
  - Examples from which the model learns
- **Features:**
  - Important attributes of the data

- **Parameters:**
  - Values the model adjusts to fit (describe) the data
- **Training:**
  - Process in which the model learns
  - It tries to reduce the difference between prediction and actual result

- **Inference:**
  - Applying the model to new data
- **Prediction:**
  - Predicted result
  - The output of the model

Deep Learning & AI

A first neuron

Jannis Seemann

Figure based on:
Wei-Chung Allen Lee, Hayden Huang, Guoping Feng, Joshua R. Sanes, Emery N. Brown, Peter T. So, Elly Nedivi- Dynamic Remodeling of Dendritic Arbors in GABAergic Interneurons of Adult Visual Cortex. Lee WCA, Huang H, Feng G, Sanes JR, Brown EN, et al. PLoS Biology Vol. 4, No. 2, e29. doi:10.1371/journal.pbio.0040029, Figure 6f. Slightly altered (plus scalebar, minus letter "f".).
License: CC-BY 2.5

A

Prediction formula:

- $\hat{y} = 1 \cdot b + X_1 \cdot w_1 + X_2 \cdot w_2 + X_3 \cdot w_3$

Example: °C to °F

- **Let's say we wanted to create a neuron to calculate °F from °C:**
  - **Input:**
    - The input only consists out of a single feature
    - Temperature in °C
  - **Output:**
    - Temperature in °F

Prediction formula:

- $\hat{y} = 1 \cdot b + X_1 \cdot w_1$
- $\hat{y} = 1 \cdot 32 + X_1 \cdot 1.8$
- **Example:**
  - $X_1 = 10°C$
  - $1 \cdot 32 + 10 \cdot 1.8 = 50°\text{F}$

Deep Learning & AI

A first neuron (in Python)

Jannis Seemann

Deep Learning & AI

What is a tensor?

Jannis Seemann

What is a tensor?

- **Tensors:**
  - A core concept in PyTorch
  - It's pretty much just a box, in which values are being stored
  - Enables efficient data handling
  - This box can be stored on optimized hardware, such as GPUs
  - Significantly speed up deep learning training and inference

0-tensor	1-tensor	2-tensor	
Scalar	**Vector**	**Matrix**	**...**
5	$\begin{pmatrix} 5 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 5 & 7 \\ 3 & 9 \end{pmatrix}$	

Deep Learning & AI

Unpacking tensors, accessing vectors

Jannis Seemann

Deep Learning & AI

Storing a matrix in a tensor

Jannis Seemann

2-tensor

Matrix

$$\begin{pmatrix} 5 & 7 \\ 3 & 9 \end{pmatrix}$$

Deep Learning & AI

From tensor to model

Jannis Seemann

Deep Learning & AI

The datatype (`dtype`) of a tensor

Jannis Seemann

Deep Learning & AI

Overview: Training a neuron

Jannis Seemann

Overview: Training a neuron

- **In this chapter:**
  - We now want to train the Neuron
  - Instead of us providing the parameters..
  - ... the neuron should learn these automatically
- **This will prepare us for the next chapter:**
  - In which we'll predict car prices based on their age / mileage, and find out which is more detrimental to the value of a used car:
    - More mileage
    - Higher age

Deep Learning & AI

How does a Neuron learn? (1st try)

Jannis Seemann

Heads-up

- The math is a bit technical, with derivatives, gradients, ...
- **My recommendation:**
  - Grab a coffee / tea, and then continue with this chapter
- **But also, don't worry too much about it:**
  - PyTorch is doing all the calculations for us, in a highly optimized way
  - **The maths is optional:**
    - You could even skip it
    - But it is the core part of how a neural network trains...
    - In a course about this topic, I can't really skip it

How does a neuron learn?

- **Example:**
  - Let's say we wanted the neuron to learn the formula for calculating Celsius → Fahrenheit

Reminder: Prediction formula

$$\hat{y} = X_1 \cdot w_1 + 1 \cdot b$$

Prediction $\hat{y}$: [°F]

Input X_1: [°C]

$$[°\mathrm{F}] = [°\mathrm{C}] \cdot w_1 + 1 \cdot b$$

$$[°\mathrm{F}] = [°\mathrm{C}] \cdot 1.8 + 1 \cdot 32$$

$$50°\mathrm{F} = 10°C \cdot 1.8 + 32$$

- **It should learn the following parameters:**
  - $b = 32$
  - $w_1 = 1.8$
- But how can we "*teach*" the correct parameters?

Terminology

- $\boldsymbol{X_1}$**:** Input (temperature in °C)
- $\boldsymbol{\hat{y}}$**:** Prediction from the model (°F)
- $\boldsymbol{y}$**:** Actual value (°F)
- $\boldsymbol{b}$**:** Bias

How does a neuron learn?

- **1st approach:**
  - We solve it as a mathematical equation
  - Since there are two unknown parameters, b and w_1, we need two equations to solve for both
  - We can then solve this system of equations
- **Learning:**
  - Learning can be seen as finding b and w_1 from examples
  - Solving the equations directly is manageable here
- **However:**
  - It is impractical with many variables (w_1, w_2, w_3, w_4, ...)
  - This approach does not scale!

Maths: °C → °F

- $[°\mathrm{F}] = [°\mathrm{C}] \cdot w_1 + 1 \cdot b$
- **Given the following equations:**
  - $50°\mathrm{F} = 10°\mathrm{C} \cdot w_1 + 1 \cdot b$
  - $100°\mathrm{F} = 37.78°\mathrm{C} \cdot w_1 + 1 \cdot b$
- **We get:**
  - $b \approx 32$ and $w_1 \approx 1.8$

Deep Learning & AI

How does a neuron learn? (2nd try)

Jannis Seemann

2nd approach: Learning

- **DJ analogy:**
  - **Mixing multiple instruments:**
    - If an instrument's volume is too high → reduce it
    - If an instrument's volume is too low → increase it
  - **Key idea:**
    - **Adjusting parameters:**
      - Each volume dial represents a parameter / variable
      - No equation system must be solved
      - Parameters are adjusted iteratively
      - This is more efficient if the model consists out of many parameters, or we don't have an ideal solution

How does a neuron learn?

- **2nd approach, in technical terms:**
  - We still need sufficient example data, like:
    - y: 100°F
    - $X_1 = 37.78°\mathrm{C}$
  - **The first step:**
    - We initialize the volume sliders / parameters randomly:
    - Example: $b = 0,\ w_1 = 0.5$
  - **Then, we calculate the result**
  - But what do we do with it?

Reminder: °C → °F

- $[°\mathrm{F}] = [°\mathrm{C}] \cdot w_1 + 1 \cdot b$
- **Equation:**
  - $\hat{y} = 37.78 \cdot w_1 + 1 \cdot b$
- **Let's fill in random values for w_1 and b:**
  - $\hat{y} = 37.78 \cdot 0.5 + 1 \cdot 0$
  - $\hat{y} = 18.89$
- **This is far from the optimal result:**
  - $y = 100$

How does a neuron learn?

- **2nd approach, in technical terms:**
  - **Idea:**
    - We need a metric to quantify how far our predictions are from the desired values, so we can adjust the parameters accordingly
  - **Let's create a loss function:**
    - The higher the loss, the worse the solution
    - Let's take a standard loss function for numerical problems:
      - Mean-squared error (MSE)
      - The exact value doesn't matter - all that's important is that we want to minimize this value

Maths: °C → °F

- $[°\mathrm{F}] = [°\mathrm{C}] \cdot w_1 + 1 \cdot b$
- **Equation:**
  - $100 = 37.78 \cdot w_1 + 1 \cdot b$
- **Let's fill in random values for w_1 and b:**
  - $100 \stackrel{?}{=} 37.78 \cdot 0.5 + 1 \cdot 0$
  - $100 \stackrel{?}{=} 18.89$
- **Mean-squared error:**
  - $L = (\hat{y} - \mathrm{y})^2$
  - $L = (18.89 - 100)^2 \approx 6580$

How to adjust the weights

- **Learning:**
  - We can nudge the parameters w_1 and b into the right direction to minimize the loss
  - **Learning rate:**
    - Usually referred to as η ("eta")
    - Defines how much we nudge the parameters / weights
    - Typically, in a range between 0.001 and 0.1
- **After:**
  - We continue with the next example...
  - ... and then start the whole process again!

Deep Learning & AI

How does a neuron learn? (gradient descent)

Jannis Seemann

How does it learn?

- **But how should we adjust the weights to minimize the loss?**
- **Idea:**
  - Let's adjust parameters based on their influence
  - The greater the impact of a parameter, the greater we need to adjust it
- **The gradient:**
  - Tells us how much a parameter affects the error
  - Shows the direction and rate of change of the error with respect to a parameter
  - We calculate the gradient for each parameter that we want to optimize

For each parameter, L

How does it learn?

- **But how should we adjust the weights to minimize the loss?**
- **Idea:**
  - Let's adjust parameters based on their influence
  - The greater the impact of a parameter, the greater we need to adjust it
- **The gradient:**
  - Tells us how much a parameter affects the error
  - Shows the <u>direction and rate of change of the error</u> with respect to a parameter
  - We calculate the gradient for each parameter that we want to optimize

[optional]: Theory

- **Loss function (MSE):**
  - $L = (\hat{y} - y)^2$
  - $L = \left((X_1 \cdot w_1 + 1 \cdot b) - y\right)^2$
  - $L = \left((37.78 \cdot w_1 + 1 \cdot b) - 100\right)^2$
- **Derivative for w_1:**
  - $\frac{\partial L}{\partial w_1} = 2 \cdot X_1 \cdot \left((X_1 \cdot w_1 + 1 \cdot b) - y\right)$
  - $2 \cdot 37.78 \cdot \left((37.78 \cdot 0.5 + 1 \cdot 0) - 100\right) = -6124.15$
- **Derivative for b:**
  - $\frac{\partial L}{\partial b} = 2 \cdot \left((X_1 \cdot w_1 + 1 \cdot b) - y\right)$
  - $2 \cdot \left((37.78 \cdot 0.5 + 1 \cdot 0) - 100\right) = -162.22$

How does a neuron learn?

- **But how do we adjust the weights based on the gradients?**
  - **We introduce a learning rate**
    - Called η ("eta")
    - Typically, between 0.001 and 0.1
    - Think of it as the step size that controls how much each weight is updated
    - This process is called **gradient descent**
  - **After this:**
    - We repeat the process for each data point, adjusting the weights as we go
    - Then, we start again from the beginning with the updated weights

Adjusting the weights, $\eta = 0.001$

- $w_1^{new} = w_1^{old} - \eta \cdot \frac{\partial L}{\partial w_1}$
- $w_1^{new} = 0.5 - 0.001 \cdot (-6124.15) = 6.62$
- $b^{new} = b^{old} - \eta \cdot \frac{\partial L}{\partial b}$
- $b^{new} = 0 - 0.001 \cdot (-162.22) = 1.62$

Deep Learning & AI

Training a neuron (part 2, in Python)

Jannis Seemann

Deep Learning & AI

Bonus: Why MSE?

Bonus: Why MSE?

- **Why did we use MSE as a loss function?**
  - $L = (\hat{y} - \mathrm{y})^2$
- **Why would this not work:**
  - $L = |\hat{y} - \mathrm{y}|$
  - This is just the difference between the predicted value ($\hat{y}$), and the actual value (y)
- **Let's explore this interactively**

Deep Learning & AI

Batch learning

Jannis Seemann

Batch learning

- **Our existing approach worked:**
  - We learned the correct parameters
  - But... **it took forever**
- **DJ analogy:**
  - Imagine constantly tuning knobs up and down for each song
  - Repeated adjustments are time-consuming
- **Batch learning concept:**
  - **Instead of adjusting for each song separately:**
    - Let's adjust for multiple songs at once
    - Back-and-forth adjustments cancel out
    - => Fluctuations are smoothed out
    - Learning becomes more stable
    - We need fewer iterations!

pandas

Deep Learning & AI

Overview: Project car prices

Jannis Seemann

Overview: Project car prices

- **Our goal:**
  - Predict vehicle price based on age and miles driven
  - **Key question:**
    - Which factor impacts resale value more:
      - An additional year of age
      - Or an extra 10,000 miles driven?
- **Additional, technical topics:**
  - In addition, we'll explore the interactive Jupyter environment
  - We will explore how to load tabular data (such as .csv files), and explore the data
  - We will explore how we can normalize data
  - We will explore how we can store / load the model

Deep Learning & AI

The data in this chapter: Used car prices

The data in this course: CSV

- In this chapter, we'll be working with a CSV file containing prices of used cars
- **The data:**
  - **License:** CC BY 4.0 ATTRIBUTION 4.0 INTERNATIONAL, https://creativecommons.org/licenses/by/4.0/
  - **Source:** TAEEF NAJIB, https://www.kaggle.com/datasets/taeefnajib/used-car-price-prediction-dataset/data

Deep Learning & AI

Data-science stack: Jupyter

Jannis Seemann

Python environment: Jupyter

- An interactive environment, in which we can run code
- Enables us to interactively write Python programs
- **Let's just explore it!**

Deep Learning & AI

Data-science stack: Pandas

Jannis Seemann

Data-science stack: Pandas

- **Pandas overview:**
  - Python library used for data analysis and manipulation
- **Key data structures:**
  - **Series:**
    - Like a single column in a spreadsheet
  - **DataFrame:**
    - Like a table with rows and columns
- **Common uses:**
  - Loading data
  - Cleaning data
  - Manipulating data

Deep Learning & AI

Data-science stack: Pandas (part 2)

Jannis Seemann

Deep Learning & AI

Data-science stack: Pandas (preparing the columns)

Jannis Seemann

Deep Learning & AI

Training a neuron (part 1)

Jannis Seemann

Deep Learning & AI

Training a neuron (part 2)

Jannis Seemann

Deep Learning & AI

Normalizing the output

The problem with our model

- **Issue summary:**
  - Data was difficult for the neuron to learn
  - Large changes in output made learning unstable
- **Result: Gradient explosion:**
  - High difference between prediction ($\hat{y}$) and the actual value (y)
  - Large gradients caused drastic weight updates
- **Impact: Unstable learning:**
  - Weights became too large to compute
  - Gradients became too large
  - This led to invalid numbers (`NaN`)

Normalizing data

- **Why normalize the output?**
  - **Stabilizes learning:**
    - Puts predictions into a smaller range
    - Results in smaller, controlled gradients
    - Leads to smoother weight updates and more stable learning
  - **How to normalize (Z-score):**
    - **Step 1 :**
      - Center data around 0
    - **Step 2:**
      - Divide by the standard deviation (σ)
    - This ensures most data is (mostly) between -2 and 2

Z-score normalization

- $y_{norm} = \frac{y-\mu}{\sigma}$
  - y_{norm} = Normalized output data
  - y = Original value of the data point (output)
  - μ = Mean of the output dataset
  - σ = Standard deviation of the output dataset

Example: Normalizing data

Original input data

Normalized input data

Deep Learning & AI

Normalizing the output: Python (PyTorch)

Jannis Seemann

Deep Learning & AI

Normalizing the input

Normalizing the input

Formula

- $\hat{y} = age \cdot w_{age} + miles \cdot w_{miles}$

- **Problem: Different scales**
  - **Age of car:**

  - **Number of miles driven:**

- **Impact:**
  - Weights for "*age*" have reduced effect on the output
  - Weights for "*miles*" have an amplified effect
- **Solution:**
  - Normalize the input data to bring everything to the same scale

Deep Learning & AI

Normalizing the input (PyTorch)

Jannis Seemann

Deep Learning & AI

Visualizing the Loss

Jannis Seemann

Learning rate

- A hyperparameter that controls how much the model's weight change during each step
- **Why is it so important?**
  - **Too high:**
    - Training becomes unstable
  - **Too low:**
    - Training becomes slow
- **Goal:**
  - Find the right balance between fast & accurate learning

Deep Learning & AI

Saving and loading a model

Jannis Seemann

Deep Learning & AI

Exercise: Adding an additional column

Exercise: Adding an additional column

- **The key question now:**
  - **Should we incorporate another column into our model?**
  - Upon examining the dataset, we notice a potentially useful column titled "`accident`"
  - **This raises a possibility:**
    - Would including the accident column benefit our model?
    - Would it decrease the mean squared error?
- Let's now investigate this column further...
- ... and then it will be your task to add it to the model

Deep Learning & AI

Solution: Adding an additional column

Deep Learning & AI

Overview:

Binary classification problems

Jannis Seemann

Binary classification problems

- **In this chapter:**
  - **Goal:**
    - Develop a neuron for binary classification
      (in our case: spam classifier, spam / not spam)
- **Steps:**
  - **Preprocessing:**
    - Convert text data into numerical data for the model
    - How could we represent text messages?
  - **Technical aspect:**
    - **Activation functions:**
      - Predicting binary outcomes (yes / no)
      - Preparation for stacking multiple neurons to a network

Deep Learning & AI

Overview:

The dataset for this chapter

The data: SMS spam collection

- In this chapter, we'll work with the SMS spam collection dataset
- You can find the `.tsv` file in the course materials
- **Original source:**
  - https://archive.ics.uci.edu/dataset/228/sms+spam+collection
  - **License:**
    - This dataset is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license
    - https://creativecommons.org/licenses/by/4.0/legalcode
    - This allows for the sharing and adaptation of the datasets for any purpose, provided that the appropriate credit is given
    - In case you find this corpus useful, please make a reference to previous paper and the web page in your papers, research, etc.:
    - http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/
    - **Creators:**
      - Tiago Almeida, Jos Hidalgo

Deep Learning & AI

Counting words:

`CountVectorizer`

Jannis Seemann

The tool: Scikit-learn

- **Scikit-learn:**
  - A popular Python library for machine learning and data analysis
  - Provides simple and efficient tools for various machine learning algorithms... but also for data preprocessing
  - **In our case:**
    - We'll use neurons / neural networks from PyTorch for our machine learning models throughout this course
    - But for preprocessing text, we'll use the tools from Scikit-learn

1st idea: Counting words

- **1st idea:**
  - We could calculate the top 1000 words...
  - ... and then keep track how often each word occurs in each message
- **`CountVectorizer`:**
  - A tool for converting a collection of text documents into a matrix of token counts
  - Commonly used for text processing and NLP (natural language processing)
  - This will allow us to turn the words into parameters for the neural network
  - **Let's explore this!**

Deep Learning & AI

[Bonus]: Counting words, TF-IDF

Jannis Seemann

TF-IDF

- **TF-IDF:**
  - **TF: Term frequency**
    - Counts how frequently a term appears in a document
  - **IDF: Inverse document frequency**
    - Reduces the weight of common terms
- **The result:**
  - TF-IDF prioritizes important, rare words over frequently occurring, less meaningful words

Mathematical formula

- **Term frequency (TF):**

$$\mathrm{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- **Inverse document frequency (IDF):**
  - **Heads-up:**
    - To avoid division by zero, we can add a smoothing factor

$$\mathrm{IDF}(t,D) = \log\left(\frac{1 + \text{Total number of documents}}{1 + \text{Number of documents containing the term t}}\right)$$

- **TF-IDF:**
  - $\mathrm{TFIDF}(t,d,D) = \mathrm{TF}(t,d) \cdot \mathrm{IDF}(t,D)$

Deep Learning & AI

Training the neuron

Jannis Seemann

Deep Learning & AI

Activation function: Sigmoid

Activation function: Sigmoid

Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Purpose:**
  - Converts the neuron's output to a value between 0 and 1
- **Idea:**
  - Apply the sigmoid function to map the output to the range $(0, 1)$
  - **Implemented in PyTorch:**
    - `torch.nn.functional.sigmoid`

Deep Learning & AI

Using sigmoid for a neuron

Jannis Seemann

Using sigmoid

- **Goal:** Predict binary classes
- **Challenge with MSE Loss:**
  - Error is too small
  - Gradients become small as well
  - Neuron struggles to learn effectively
- **Result:**
  - We need a different loss function for better learning

The problem with MSE Loss

- **Let's say:**
  - $\hat{y} = 0.9$
  - $y = 1$
- **Using MSE (mean square error):**
  - $L = (y - \hat{y})^2 = 0.01$
  - The error seems deceptively small, leading to minimal updates!

Using sigmoid

- **New loss function:**
  - `torch.nn.BCEWithLogitsLoss:`
    - **`BCE`:** Binary Cross Entropy
    - **`WithLogits`:** Also applies the sigmoid function before
- **Heads-up:**
  - The output of the neuron is still linear
  - **The sigmoid activation function became part of the loss function:**
    - Simplifies mathematical calculations
    - Improves numerical stability
  - **During prediction:**
    - Apply sigmoid manually

BCE loss

- $L = y \cdot \log(\sigma(\hat{y})) + (1 - y) \cdot \log(1 - \sigma(\hat{y}))$
- $\boldsymbol{\sigma}$**:** Sigmoid function
- $\boldsymbol{y}$**:** True y value
- $\boldsymbol{\hat{y}}$**:** Predicted y value

Deep Learning & AI

PyTorch:
Using sigmoid for a neuron

Jannis Seemann

Deep Learning & AI

Evaluating the model

Jannis Seemann

Evaluating the model

- We can use several metrics to evaluate the model
- **Among others, the most important ones:**
  - **Accuracy:**
    - Proportion of correctly classified instances (spam or not spam)
    - *For what percentage is this true:* $y = \hat{y}$
  - **Sensitivity (true positive rate):**
    - Proportion of actual spam messages correctly identified
    - *Given* $y = spam$*, what percentage did the model predict correctly*
  - **Specificity (true negative rate):**
    - Proportion of actual non-spam messages correctly identified
    - *Given* $y = not\ spam$*, what percentage did the model predict correctly?*
  - **Precision:**
    - Proportion of predicted spam messages, that are spam
    - *Given* $\hat{y} = spam$*, what percentage had been* $spam$ *originally?*

Deep Learning & AI

Evaluating the model: Train / Validation / Test

Jannis Seemann

Train / Validation / Test

- **Now, we want to evaluate:**
  - How good is the model? Should we change something in the structure (hyperparameters)?

- **Training data (60-70%):**
  - Already used for training the model
  - We can't use this data for validating the performance

- **Validation data (10-20%):**
  - Used for selecting the model architecture and deciding when to stop training
  - Used to tune the "*hyperparameters*"
  - This data has already been "*seen*" by our algorithm, once we chose a model

- **Test data (10-20%):**
  - We need completely new data to answer this question
  - We could've held back data from the start to use for test data
  - Or we can collect new data now

Deep Learning & AI

Evaluating the model: Train / Validation (in Python)

Jannis Seemann

Deep Learning & AI

Testing the model

Testing the model

- **Important:**
  - We could've done it by holding back 10-20% of the initial data
- **However:**
  - This data is from 2011, and is already quite old
  - We can also just test it on new data instead
  - How would it perform here?
  - **Heads-up:**
    - We will just apply the model to new data
    - We won't calculate accuracy or other measurements for it
    - But of course, before deploying this model, we should do so!

Deep Learning & AI

Bonus: Improving performance with an LLM

Jannis Seemann

Bonus: Using an LLM

- **LLM:**
  - Large Language Model
  - Example: ChatGPT
- How does an LLM work?

- **Tokenizing the text**
  - Splits words into shorter tokens

- **Training of LLM:**
  - LLM tries to predict the next token
  - For this, it must develop an internal representation of the input text

- **Generating text:**
  - Based on the internal representation, the LLM predicts the next token

Structure of an LLM (overly simplified)

- An LLM consist out of many layers, that are attached to each other

Idea: Embeddings

- **Idea:**
  - Let's take an existing, pre-trained LLM
  - How about we take the internal representation of the input inside the LLM
  - ... and just add our Neuron after it?
- **This internal representation:**
  - Captures the meaning of the text
  - Can we use it to enhance our spam filter?

Deep Learning & AI

Bonus: Generating Embeddings (part 2)

Deep Learning & AI

Bonus: Using Embeddings for the Spam filter

Jannis Seemann

Deep Learning & AI

Overview:

From Neuron to Neural Network

Jannis Seemann

Overview: Neural Network

- **So far:**
  - We have learned how a single neuron learns
- **Next:**
  - Let's stack multiple neurons to create a neural network
  - This enables learning of more complex relationships

From Neuron to Neural Network

From Neuron to Neural Network

Deep Learning & AI

[Optional]:
The linearity problem

Jannis Seemann

The linearity problem

- **Linearity = Limited learning capability:**
  - Cannot capture complex, non-linear patterns
- A single, linear neuron (even with a sigmoid function) won't be able to capture these non-linear relationships

Linear relationships can be learned by the network

Non-linear relationships can't be learned by the network (yet)

Linearity: Limited Learning Power

- **The problem:**
  - If hidden layers are linear, the final output layer is also linear
  - Behaves like a single linear model, offering no extra learning power
- **Conclusion:** Non-linear activations are required to unlock the network's full potential

Solution: Add non-linear activation functions

- **Why non-linear activation functions?**
  - Helps the network learn non-linear patterns
  - Enhance the model's ability to capture complex relationships
- Any activation function works, as long as it breaks linearity

Activation function

Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **First idea:** Apply the sigmoid function
  - Breaks linearity
  - **Sufficient for learning:** A large enough network with sigmoid activations can approximate many other functions

Deep Learning & AI

[Maths, optional]: The linearity problem

Jannis Seemann

- $\boldsymbol{w}^{[1]}$: Weights for the 1st layer
- $\boldsymbol{w}^{[2]}$: Weights for the 2nd layer

How not to do it

- **We can just simplify the network:**

The solution

- We need to break this linearity
- **The idea:**
  - Let's put a sigmoid activation function around the output
  - Suddenly, we can't simplify the formula anymore
  - **Which is great:** Now our network can grasp non-linear concepts

Deep Learning & AI

Backpropagation

Jannis Seemann

- **1st step: Forward propagation:**
  - The model is applied to the data to make a prediction
  - The error (difference between predicted and actual value) is calculated
- **2nd step: Backpropagation:**
  - Compute gradients to propagate the error backward to determine how much each neuron influences the error
- **3rd step:** Adjust weights of the previous layer to reduce the error

Deep Learning & AI

[Optional]:
The math behind backpropagation

Jannis Seemann

[optional]: The math behind it

- **Sigmoid:**
  - $\sigma(x) = \frac{1}{1+e^{-x}}$
  - $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$
- **Loss function (MSE):**
  - $L = (\hat{y} - y)^2$
- **Weights:**
  - $w_{jk}^{[i]}$: Weight from neuron j to neuron k in layer i
- **Error:**
  - $e_k^{[i]}$: Error for neuron k in layer i

Heads-up:
The formula for e_{output} varies with the choice of loss function (in this case: MSE)

$$e_1^{[2]} = \frac{\partial L}{\partial \hat{y}} = 2\hat{y} - 2y$$

$$e_1^{[1]} = e_1^{[2]} \cdot w_{1,1}^{[2]} \cdot \sigma'(x)$$

$$e_3^{[1]} = e_1^{[2]} \cdot w_{3,1}^{[2]} \cdot \sigma'(x)$$

Deep Learning & AI

The data for this chapter

Deep Learning & AI

Optional:
Developing a neuron

Jannis Seemann

Deep Learning & AI

Developing a first network

Jannis Seemann

The structure of the network

- **The structure of our network:**
  - 2 inputs
  - 10 neurons in the hidden layer
  - 1 output neuron

Deep Learning & AI

Applying the network: Calculating accuracy

Jannis Seemann

Deep Learning & AI

PyTorch: Simplifying the code with `nn.Sequential`

Jannis Seemann

Deep Learning & AI

Optimizing training: Activation function ReLU

Jannis Seemann

Activation function

- **So far:**
  - We have used the sigmoid function as an activation function
  - The main goal was to break linearity - the weights of the network can then adjust to the rest

...

Activation function: ReLU

- **So far:**
  - We have used the sigmoid function as an activation function
  - The main goal was to break linearity - the weights of the network can then adjust to the rest
- **How about we change the activation function?**

...

Activation function: ReLU

- **ReLU:**
  - Rectified Linear Unit
  - Enables us to train large neural networks
  - **In general:** One of the most popular activation functions for neural networks

ReLU function

$$f(x) = \max(0, x) = \frac{x + |x|}{2}$$

ReLU

- **Advantages:**
  - **Simplified gradient propagation:**
    - The sigmoid function outputs values between 0 and 1, which can result in very small gradients during backpropagation
    - ReLU outputs either 0 or the input itself (if positive)
    - This preserving larger gradients, making backpropagation easier
  - **Sparse activation:**
    - Only about 50% of neurons are activated => reduced computational complexity and improved generalization
  - **Efficient computation:**
    - It only involves a `max()` operation
    - This is easier to compute compared to exponentials

Deep Learning & AI

Optimizing training: The optimizer Adam

Jannis Seemann

How to optimize parameters

- **Previously:**
  - To train the network, we had to minimize the loss function
  - **This means:** Our goal was to find the best parameters (w_1, w_2, b) to minimize the loss function
  - **Let's focus on a single parameter, such as w_1:**
    - If the parameter is too low, the loss is high
    - If the parameter is too high, the loss is high as well
  - Our goal is to minimize the loss
- **Our current approach:**
  - We picked a random value to initialize the parameter
  - And then walked our way down from there...
  - ... step by step
  - The steeper the slope, the faster we descended
  - Eventually, we reach the minimum

Loss function

- $L = (\hat{y} - y)^2$
- $L = ((X_1 \cdot w_1 + X_2 \cdot w_2 + 1 \cdot b) - y)^2$

Loss, depending on w_1

Adaptive Moment Estimation

- **Isn't there a faster approach?**
  - How about instead of walking...
  - ... we take a ball and let it pick up speed?
- **Our new approach:**
  - We still pick a random value to start with (w_1)
  - But we're now a ball, picking up speed (*momentum*)
  - This gets us to the bottom even faster
  - **In addition:**
    - Other things can be optimized as well
    - **Example:** In steep slope areas, we'll take smaller "steps" (but still pick up momentum)
    - But we don't dive into that here
- **This approach is called:**
  - **Adam:** Adaptive Moment Estimation

Loss function

- $L = (\hat{y} - \mathrm{y})^2$
- $L = ((X_1 \cdot w_1 + X_2 \cdot w_2 + 1 \cdot b) - \mathrm{y})^2$

Loss, depending on w_1

Deep Learning & AI

Mini-batch learning

Jannis Seemann

Mini-batch learning

- **So far:**
  - We calculated the loss based on the whole training set
  - So far, this had been easily possible
  - But what if this data no longer fits into memory?
  - Do we always have to run gradient descent on all data?
- **The idea: Mini-batch learning**
  - We split the data into smaller chunks, for example 32 or 64 entries per chunk
  - And train with these chunks, one chunk at a time
  - For the same amount of compute, we can do significantly more steps
  - This allows us to approach the minimum faster

Deep Learning & AI

Mini-batch learning: Keeping track of the loss

Jannis Seemann

Deep Learning & AI

Exercise:

Predicting loan approvals

Jannis Seemann

Exercise: Predicting loan approvals

- Exercise: Build a model to predict if a loan gets approved or not (`loan_status`)
- **You can especially use the following features:**
  - `person_income`, `loan_intent`, `loan_percent_income`
  - **Heads-up:**
    - `loan_intent` is a categorical variable
    - You can use the following code to turn it into multiple binary columns:
      - `pd.get_dummies(df, columns=["loan_intent"]).astype("float32")`
    - You may need to normalize the X data before applying the model
  - **Also, examine, if it would make sense to include an additional column:**
    - `credit_score`
  - **Also, try to find answers to the following questions:**
    - Is a single neuron enough, or could a neural network improve the accuracy?
    - What accuracy are you able to achieve?

Deep Learning & AI

Exploring the dataset: Loan approvals

Jannis Seemann

Deep Learning & AI

Sample solution: Predicting loan approvals (part 2)

Jannis Seemann

Deep Learning & AI

Overview:

Classifying handwritten digits

Jannis Seemann

Classifying handwritten digits

- **In this chapter:**
  - We aim to predict the digit in an image
- **For this:**
  - We need to explore how we can work with 2-dimensional data: *matrices*
  - We need to explore how we can classify an image into 10 different categories at once
  - (Softmax, cross entropy loss)

Deep Learning & AI

Overview: MNIST data

Jannis Seemann

Overview: MNIST data

- **What is MNIST?:**
  - A popular dataset in machine learning, specifically used for image classification
  - Consists of handwritten digits from 0 to 9
- **Dataset Details:**
  - 60,000 training images
  - 10,000 test images
  - Each image is 28x28 pixels in size, grayscale
- **Why use MNIST?**
  - Standard benchmark
  - Great for getting started with image detection

Deep Learning & AI

From DataSet to DataLoader

Jannis Seemann

Deep Learning & AI

Predicting: 0 vs. not a 0

Predicting: 0 vs. not a 0

- **For now, let's stick to what we know:**
  - We can create a neural network to predict whether an image contains a 0 or not
  - How would we architect this network?

- ***Note:***
  - *Arrows and biases omitted for clarity reasons*

Deep Learning & AI

Evaluating the model

Jannis Seemann

Deep Learning & AI

Theory:

Predicting 10 different classes

Jannis Seemann

Predicting 10 different classes

- **From the perspective of a neural network:**
  - Neural networks recognize digits by analyzing pixels, not by understanding numerical values or their order
  - They don't understand the numerical sequence of 1 coming before 2. They just see different patterns of pixels
  - **Thus:**
    - Each digit is treated as a separate category
    - ... just as a cat is different from a dog
    - ... there's no implied sequence or relationship

Predicting 10 different classes

- **Now imagine:**
  - You had a slider from 0 to 9 to tell me what kind of animal is in a picture...
  - ... you could move the slider to 2, indicating that it's a dog
  - ... or put it to 1.95 to indicate you're quite confident that it's a dog, but it might (although extremely unlikely) also be a cat

Predicting 10 different classes

- **But what if you got slightly confused by the costume?**
  - Suppose you were 80% sure it's a dog...
  - ... but by 20% thought it could be a bear?
  - If you combine these guesses, you might end up with a mixed prediction, landing somewhere closer to the number 3

- **TLDR:** Averaging probabilities from multiple guesses does not work for categorization

Predicting 10 different classes

- Imagine I gave you 10 different sliders
- Now, this problem becomes trivial!

- Let's give our network 10 output options!

- $\hat{y}_0$:
  - The predicted probability that this image shows a 0
  - **The goal:** 0 => 0% likely, 1 => 100% likely

Deep Learning & AI

Idea: One-hot encoding

One-hot encoding: Training the data

- But how do we generate the desired output data for the network?
- We can't train the network with this vector for the y data: `[1, 0, 2, 3, ...]`
- Each of the 10 outputs needs to be trained separately!

Desired output when the image shows a 2 ($y = 2$):

`[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]`

One-hot encoding

- How do we build this format?
- One-hot encoding!
- **In numerical terms:**
  - `[[0, 1, 0, 0, 0, 0, 0, 0, 0, 0],` ← Desired output for the 1st image of the training data, $y = 1$
    `[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],` ← Desired output for the 2nd image of the training data, $y = 0$
    `[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],` ← ...
    `[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],`
    `[0, 0, 0, 0, 0, 1, 0, 0, 0, 0], ... ]`
- **Luckily, one-hot encoding is already implemented for us:**
  - ```
    import torch.nn.functional as F
    F.one_hot(tensor)
    ```
    ```

Deep Learning & AI

Training a network with 10 output neurons (part 2)

Jannis Seemann

Deep Learning & AI

Predicting 10 different classes: Softmax

Jannis Seemann

The problem so far

- Let's say we train the model to predict 10 different classes / probabilities
- But somehow, it doesn't add up:

Example: Predicted output when the image shows a 1 ($y = 1$):

`[0.01, 0.9, 0.15, 0.1, 0.01, 0.0, 0.02, 0.2, 0.01, 0.01]`

- **The probabilities sum to more than 100%!**

The solution: Softmax

- **Softmax idea:**
  - **First, we exponentiate each value:**
    - This makes larger values "stand out" even more compared to smaller ones
    - In addition, it makes sure that the value is always positive
  - **After:**
    - We normalize this by value by dividing it by the sum of all exponentiated values (of all neurons)

Softmax formula

- **Terms:**
  - $\boldsymbol{x_i}$: Activation of an individual neuron
- **Calculating the exponent:**
  - e^{x_i}
  - **Example:**
    - $e^1 \approx 2.72$
    - $e^2 \approx 7.39$
- **Softmax formula:**

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^j}$$

How to apply Softmax

- **How to apply softmax:**
  - **Training:**
    - It's best to use a loss function with built-in softmax:
      - `torch.nn.CrossEntropyLoss()`
  - **Prediction:**
    - We must apply softmax manually:
      - `torch.nn.functional.softmax`

Deep Learning & AI

PyTorch: Using Softmax

Jannis Seemann

Deep Learning & AI

Expanding our model: Let's add an additional layer

Jannis Seemann

Let's include an additional layer

- **Current model:**
  - Streamlined and simple
- **New approach:**
  - Let's add an extra layer for more flexibility
  - Potential for improved results?
- **Caution:**
  - More layers / neurons increase complexity:
    - Increased complexity can lead to *overfitting*

      (we'll discuss this more later in this chapter)
  - The model might learn irrelevant patterns

The previous setup

The new setup

The new setup

Why not even more layers?

Deep Learning & AI

Evaluating during training

Jannis Seemann

Deep Learning & AI

Overfitting:

Exploring the problem

Jannis Seemann

Overfitting

- **Open questions:**
  - Is a larger model always better?
  - What is the correct size of a model?
  - How should we structure a model?
  - How do we know if we need additional data?

Deep Learning & AI

The concept: Overfitting

Jannis Seemann

Deep Learning & AI

Solving overfitting

Jannis Seemann

Concept: Overfitting

- **Definition:**
  - Overfitting occurs when a model learns not only the general patterns, but also the noise or random fluctuations in the training data
- **Characteristics:**
  - Performs well on training data, but fails to generalize to unseen data
  - **High variance:**
    - Adapts too closely to specific data points instead of general patterns

Deep Learning & AI

Solving overfitting

Jannis Seemann

Solving overfitting

- **Causes:**
  - The model is too complex:
  - Too many layers or neurons
  - Insufficient training data
- **Possible solutions:**
  - Limit model complexity
  - Provide more training data
  - Halt training before overfitting starts (early stopping)
- **Heads-up:**
  - If both training and validation loss are increasing...
  - ... the model doesn't learn anything (not even unimportant details)
  - => Learning rate is too high

Deep Learning & AI

Bonus: Using the model on our own image

Bonus: Applying the model

- **To apply the model to our own data:**
  - We should use the same pre-processing pipeline as the researchers
  - Otherwise, our results may vary
- **The problem:**
  - I don't know the pre-processing pipeline of the researchers
  - I'll just try to see if the network generalizes properly to images that may be different from the training data
- **The steps now:**
  - **First:** Saving the model
  - **Second:** Loading the model in a separate file, and then using it for making a prediction

Deep Learning & AI

Bonus: Using the model on our own image (part 2)

Jannis Seemann

Deep Learning & AI

Overview:
Convolutional Neural Networks (CNNs)

Jannis Seemann

Overview: Convolutional Neural Networks (CNNs)

- **So far:**
  - Our neural network has been able to learn how to detect the contents of an image
  - The way we trained it made learning difficult
  - The model had to learn everything from just 784 numerical values from each input
- **However:**
  - The input data is an image:
    - An image has a width and a height (here: $28\ x\ 28$ pixels)
    - The surrounding pixels play an important role
    - **Example:**
      - We could run an edge detection
      - Being able to find these edges, and then making a prediction based on that - this could be useful!

Deep Learning & AI

The data in this chapter: Fashion MNIST

The dataset: Fashion MNIST

- **Fashion MNIST:**
  - Created as a drop-in replacement for the original MNIST dataset
  - Developed by Zalando Research for benchmarking ML algorithms
  - Significantly more challenging than the original MNIST data
- **Classes:**
  - T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot
  - Each class is associated with a label from 0 to 9
- **Let's explore the dataset!**

Deep Learning & AI

Establishing a baseline: Fashion MNIST

Jannis Seemann

Deep Learning & AI

Computer vision: Edge detection

Jannis Seemann

Edge detection

Sobel operator

- Here, $*$ stands for the 2-dimensional convolution operation
- We could also use different filters to detect different kinds of edges (horizontally, diagonally,...)

The image

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$*$

Filter

1	0	-1
1	0	-1
1	0	-1

$=$

Result

0	30	30	0
0	30	30	0
0	30	30	0

$$
\begin{aligned}
&G_{1,1} = \\
&10 \cdot 1 + 0 \cdot 10 + (-1) \cdot 10 + \\
&10 \cdot 1 + 0 \cdot 10 + (-1) \cdot 10 + \\
&10 \cdot 1 + 0 \cdot 10 + (-1) \cdot 10 \\
&= 0
\end{aligned}
$$

Deep Learning & AI

CNN: Idea

CNN: Idea, edge detection

- **Goal:**
  - Let the model learn to detect edges automatically
- **Approach:**
  - Provide the model with an image (e.g. $1x28x28$ pixels)
  - Use a CNN layer to apply a filter
  - The filter of the CNN layer is initialized randomly
  - And learned automatically (as parameters to this layer)

The new setup

Deep Learning & AI

CNN: Implementing a CNN in PyTorch

Jannis Seemann

Padding

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0

Deep Learning & AI

CNN: Implementing a CNN in PyTorch (part 2)

Jannis Seemann

The previous setup

Deep Learning & AI

CNN: Reducing model complexity

Jannis Seemann

The previous setup

CNN: Reducing the complexity of the network

- **Problem:**
  - Intermediate output (3 channels, 28x28 pixels) contained many values
  - The next hidden layer had to learn many weights
  - This added complexity, and made learning take more time
- **Solution:**
  - Use a `MaxPool2d` to scale down the data
  - This reduces the size of the input for the next layer, making the network simpler to train

Example: `MaxPool2d`

Why maximum pooling?

- Why don't we average the values over $2x2 = 4$ pixels?
- Yes, we could use Average Pooling (AvgPool2d) instead
- **However:**
  - However, calculating this is more complex
  - With maximum pooling, 3 out of 4 paths are terminated, meaning fewer paths need to be backpropagated

The new setup

Deep Learning & AI

Overview: PyTorch on a GPU

Jannis Seemann

Running PyTorch on a GPU

- **Why do we want to run PyTorch on a GPU?**
  - GPUs are optimized for parallel operations, such as vector and matrix calculations
  - These are exactly the operations that help us train deep learning models
- PyTorch supports different backends to perform calculations
- **Example:**
  - **Cuda (Compute Unified Device Architecture, NVIDIA):**
    - **`torch.cuda.is_available()`:**
      - Returns true if the CUDA backend is currently available
  - **Metal (macOS):**
    - **`torch.mps.is_available()`:**
      - Returns true if the mps backend is currently available
      - mps = Metal Performance Shader Graph framework
- **The setup of these backends is completely optional for this course!**

Deep Learning & AI

[Optional, Setup]: Running PyTorch with CUDA

Jannis Seemann

Deep Learning & AI

[Optional, Setup]: Running PyTorch on a free GPU!

Jannis Seemann

Deep Learning & AI

Running calculations on a GPU / MPS

Jannis Seemann

Running calculations on different devices

- To run calculations on a GPU, we need to send the calculation graph / model / tensors to the device:
  - `tensor = tensor.to(device)`
- **We need to do this manually:**
  - This gives us more control over the GPUs memory
  - And we can ensure we don't run into memory limits
  - **Heads-up:**
    - When allocating too much memory, my MacBook crashed repeatedly
    - This may also happen on your system
- **To release GPU memory:**
  - You can just remove the python variable
  - GPU memory will be released eventually: `del tensor`
  - **When using `CUDA`:**
    - `torch.cuda.empty_cache()`: Releases GPU memory back to the operating system

Deep Learning & AI

Running a model on a GPU / MPS

Jannis Seemann

Deep Learning & AI

Enhancing performance

Jannis Seemann

Deep Learning & AI

CNN: Stacking multiple CNN layers

Jannis Seemann

Inputs:
Shape: (1, 28, 28)
CNN Layer, 3 filters:
Shape: (3, 28, 28)
MaxPool2d
Shape: (3, 14, 14)
CNN Layer, 6 filters:
Shape: (6, 14, 14)
MaxPool2d
Shape: (6, 7, 7)
Flatten:
Shape: (294)
Hidden layer:
Shape: (100)
Output layer:
Shape: (10)
Softmax:
10 predictions

Deep Learning & AI

CNN: Training a larger model

Jannis Seemann

Deep Learning & AI

Concept: Dropout

Concept: Dropout

- Imagine you have to make important decisions
- And one of your advisors is always correct
- **You could then learn the following strategy:**
  - Ignore all other advisors
  - And only listen to this one advisor
- **But how will this approach work in the future?**
  - You will fully rely on a single advisor
  - If this advisor is ever wrong, you would still follow the advice
  - Wouldn't it be better to avoid relying on a single opinion alone?

Concept: Dropout

- **For the neuron:**
  - Overly relying on a single advisor is easy to learn
  - Neurons tend to prefer simple strategies
  - We need to *force* neurons to learn more complex strategies
- **Idea:**
  - **During training:**
    - How about we just randomly drop connections between individual neurons?
    - Which connections are dropped is decided randomly for each training pass
    - This ensures a neuron never solely relies on a subset of neurons
  - **During inference:** We keep all connections - they all contain valuable information!

Dropout

- **During training:**
  - If we dropped $1/3$ of the incoming connections
  - The values of the other connections must be increased by $1/3$ to make up for it

Deep Learning & AI

Implementing Dropout

Jannis Seemann

Deep Learning & AI

Optimizing the model: Batch normalization

Jannis Seemann

Batch normalization

- **Previously:**
  - We had explored that standardizing the input data can be beneficial for training
- **The question now:**
  - Could it also be beneficial within the network?

Batch normalization

- **Previously:**
  - We had explored that standardizing the input data can be beneficial for training
- **The question now:**
  - Could it also be beneficial within the network?

Batch normalization

- **Idea:**
  - We turn this normalization into a separate layer
  - **The data will be:**
    - Standardized
    - And adjusted to a new mean and standard deviation, based on what's best for the network (learned automatically)
- **This is called:**
  - Batch normalization
- **Let's try it out!**

Deep Learning & AI

[Optional math]: Batch normalization

Jannis Seemann

For each output

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$$

- **Where:**
  - $\boldsymbol{y}$**:** Normalized value (after batch normalization)
  - $\boldsymbol{x}$**:** Original value
  - $\boldsymbol{\mu}$ **(mu):** Average / mean value, calculated across the batch or previous data
  - $\boldsymbol{\sigma}$ **(sigma):** Standard deviation, calculated across the batch or previous data
  - $\boldsymbol{\epsilon}$ **(epsilon):** Small constant, used for numerical stability
  - **Learned parameters:**
    - $\boldsymbol{\gamma}$ **(gamma):** New standard deviation
    - $\boldsymbol{\beta}$ **(beta):** New offset applied to shift the normalized data

Batch normalization

Deep Learning & AI

Training the final model

Jannis Seemann

Deep Learning & AI

Overview: Transfer learning & tire prediction

Jannis Seemann

205/55 R16 V
Continental

Overview: Transfer learning & tire prediction

- **In this chapter:**
  - We aim to develop a model to help predict the quality of car tires
  - Over time, rubber deteriorates - can we detect this by a simple image?
- **As you can imagine:**
  - This detection can be rather difficult to learn
  - **In addition:**
    - We only have 1,854 images available
    - Learning everything from scratch would be extremely difficult for our model
  - **Idea:**
    - Can we use an existing model... cut its end off, and replace it with our own?
    - By doing so, we would benefit from a model that has already learned to "see"
    - For this, we might have enough data!
    - But what model could we use?

Deep Learning & AI

The data in this chapter

Deep Learning & AI

Transfer learning: The ResNet model

Jannis Seemann

Transfer learning

- **Transfer learning:**
  - A machine learning technique where a model trained on one task is reused or adapted for a new, related task
  - Reduces training time and computational resources
  - Achieves better performance, especially when data is limited in the new task
  - **How it works:**
    - We start with a pre-trained model (typically trained on a large dataset)
    - We then adjust the model to optimize it for the new task

The model: ResNet

- In this chapter, we will utilize transfer learning based on the ResNet model
- **ResNet:**
  - Residual Network is a deep neural network architecture
  - Introduced by Microsoft Research in 2015
  - Trained on the ImageNet dataset:
    - Achieves state-of-the-art accuracy in image classification tasks
  - **Key concept: Residual blocks**
    - Allow layers to "skip" connections
    - Helps combat the vanishing gradient problem, enabling deeper networks

ImageNet database

- **ImageNet:**
  - Published in 2009 at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
  - Used to train Convolutional Neural Networks
  - Contains over 1 million hand-annotated images, including their bounding boxes
  - Over 20.000 classes: ("banana", "balloon",...)
  - ImageNet does not own the images, images are publicly available
- **Yearly challenge (ILSVRC):**
  - ImageNet Large Scale Visual Recognition Challenge
  - Neural Networks compete to correctly classify and detect objects and scenes
  - "Only" 1000 distinct classes need to be predicted

Image used for demonstration purposes only. Image may or may not be part of the ImageNet dataset.

Vanishing gradients

- When a neural network consists of many layers, the gradients can diminish
- **Imagine:**
  - You told your colleague something, your colleague told this to another colleague, and so on
  - At the end, important details of the message will have gotten lost

- **How can we avoid this, and still train a large model?**

Vanishing gradients

- During backpropagation, the initial error is multiplied with the weight in each path
- If multiple weights between $(0,1)$ are encountered on a path... the adjustment becomes smaller
- **In addition:**
  - Additional degradation (of training accuracy) may occur when adding more and more layers
  - You can imagine this as problems in the flow of information

Residual Neural Networks

- **ResNet50:**
  - Residual Network with 50 layers
  - Resnet introduces the concept of "*residual blocks*"
  - These blocks enable the network to focus solely on the difference
  - The original data is fed in after every few layers

Inputs:
CNN Layer
CNN Layer
+
CNN Layer
CNN Layer
+
CNN Layer
CNN Layer
+

Deep Learning & AI

Using ResNet

Jannis Seemann

Deep Learning & AI

Bonus:

Let's explore the paper

Bonus: Let's explore the paper

- **[Optional]:**
  - Let's explore it in the scientific paper
  - http://arxiv.org/pdf/1512.03385

Deep Learning & AI

Part 1: Loading the data

Jannis Seemann

Deep Learning & AI

Part 3: Creating the model (Transfer learning)

Jannis Seemann

Deep Learning & AI

Part 4: Training the model

Jannis Seemann

Deep Learning & AI

Part 5:

Finding the best parameters

Jannis Seemann

Idea:

Generating additional data

Generating additional data

- **The problem:**
  - We want to train the model with additional data...
  - ... without having additional data
  - **The idea:**
    - Let's just reuse existing data
    - Then, instead of having to manually create additional photos, existing photos could be reused in different ways
  - **In this case:**
    - Flip the image horizontally / vertically
    - Slightly crop the image
    - Rotate the image slightly by a few degrees
    - Move the image slightly by a few pixels
    - Adjust brightness or contrast

Deep Learning & AI

Training the model with additional data

Jannis Seemann

Problem:

Weights on `mps` device

Deep Learning & AI

Making a prediction

Deep Learning & AI

Testing the model

Jannis Seemann

Testing the model

- In our case, we want to use the model in a car workshop
- I tested the model with my own images
- **But before deploying:**
  - We'd have to test the model properly
  - How does it perform on real tires?
- **In addition, when preparing this course:**
  - The model indicated my tires were no longer good
  - But this can't be, they're just 1 year old
  - **Idea:**
    - Maybe it's the dirt on the tire?
    - Once I cleaned the tire, it managed to detect it as good
- **Maybe there are some additional constraints that we need to be aware of when taking the pictures?**

Testing the model

- **In practice:**
  - We need to use labeled data from a car workshop
  - **For example:**
    - 1,000 pictures of tires, of which 500 are alright, and 500 are faulty
    - If we wanted to collect data for training as well:
      - We might need even more data
  - These pictures must be labeled accurately (usually manually)
  - **How can we label the data?**
    - We can do it ourselves
    - We can hire people to do so
    - We can outsource it to specialized companies
    - Or use specialized services such as Amazon Mechanical Turk
  - **However, labeling must be done by hand!**

Deep Learning & AI

Overview:

Deploying the model

Jannis Seemann

Deploying the model

- We now want to deploy the model
- For this, we will use gradio
- **Gradio:**
  - A tool that allows us to easily deploy machine learning and AI apps
- **At the end of the chapter:**
  - I can go to my garage, open a website on my phone, take a picture there, and get a prediction from the model!
- **Important:**
  - We have not tested the model – never trust the model!
  - If you suspect your tires might be old – go to a specialist to have it checked
  - We don't even know how well the model performs for this use case!

Deep Learning & AI

Exploring gradio

Jannis Seemann

Deep Learning & AI

Gradio: Accepting image uploads

Jannis Seemann

Deep Learning & AI

Gradio: Combining with PyTorch

Jannis Seemann

Deep Learning & AI

Gradio: Accessing it from a phone

Jannis Seemann