Introduction to Data Science 2024
Assignment 5

Your solution to Assignment 5 must be uploaded to Absalon no later than April 9th, 15.00. Responsible TAs: Antonia Karamolegkou(antka@di.ku.dk), Mustafa Hekmat (muhe@di.ku.dk).

Guidelines for the assignment:

- **The assignments in IDS must be completed and written individually.** This means that your code and report must be written completely by yourself.
- Upload your report as a single PDF file (no Word .docx file) named `firstname_lastname.pdf`. The report should include all the points of the deliverables stated at the end of each exercise. Adding code snippets to the report is not always required, but nice to have especially when you implement things from scratch.
- Upload your code as `firstname.lastname.zip`. Upload it in a zip archive, even if there is only one file. Expected is either one or several "`.py`" files or a "`.ipynb`" notebook.
- We will grade using Python 3.10 and the specific package versions specified in requirements.txt. Other versions may work, or may break; using these versions is the safest. You can create an appropriate environment with all the requirements using Anaconda or reuse the same environment you had in the previous assignment and install the requirements file.
- To reuse a previous environment, 1) open a terminal or command prompt and navigate to the directory containing the requirements.txt using `cd` (e.g. `cd Desktop/courses/ids/as5`). 2) activate your environment (make sure to replace `name_of_env` with the name you specified in the previous assignment).

  ```
  conda activate name_of_previous_env
  pip install -r requirements.txt
  ```

  Otherwise, if you want to create a new environment you first need to run:

  ```
  conda create --name my_env python=3.10
  ```

  Then use the above two command lines to install the corresponding requirements.

1 Academic Code of Conduct

You are welcome to discuss the assignment with other students, but sharing of code is not permitted. Copying code or text from the report directly from other students will be treated as plagiarism. Please refer to the University's plagiarism regulations if in doubt. For questions regarding the assignment, please ask on the Absalon discussion forum.

In short, plagiarism means copying text or ideas from others without acknowledging the underlying sources. Crucially, this does not mean that you are prohibited from building on others' ideas or use external sources, but rather that you have to properly acknowledge all sources used in your work. This holds for instance for building on code from lectures or lab sessions. If in doubt, we recommend erring on the side of over- rather than under-acknowledging sources.

While guidance on AI assistance usage is not covered in this course, you are welcome to take a look at the Absalon course on Learning resources for Digital Literacy.

You are also welcome to use AI assistance (e.g., ChatGPT, GitHub Copilot) for tutoring purposes, augmenting the TAs for help with questions and issues. However, keep in mind that their output is not guaranteed to be either comprehensive, true or aligned with the course scope and expectations. Always check with the TAs in case of doubt. Importantly, the use of AI assistance while writing the assignment is allowed only for the following purposes:[1]

- As coding tools (e.g., GitHub Copilot): no restrictions.
- As writing tools to improve the writing of original content, i.e. when the prompt you write contain all the ideas to be formulated: no restrictions.
- As search tools to identify related literature: no restrictions. Usual citation requirements apply (see plagiarism note above): you must cite the original work you identify, even if you used an LLM to find it. Just like you do not cite Google Search for papers you find using it, you should not cite ChatGPT for this either. In particular, always make sure that the citations it provides actually exist—LLMs are known to often generate plausible but nonexistent references.
- As generation tools for *new* ideas: generated content must be clearly highlighted even if post-edited by yourself. All prompts/transcripts from the tools used must be included as an appendix at the end of the submission in PDF, after the references.

For all uses of AI assistance, the purpose, tool, and version[2] must be stated in the submission—e.g, in a dedicated section. Here is such a statement for example:

```
ChatGPT August 3 Version was used as a writing assistance tool and as a serch tool
 to identify related literature.  GitHub Copilot July 14 Version was used while
                        developing the code for...
```

Neural Networks

Exercise 1 (Single-Layer Neural Network)**.** In this exercise, we will explore a single layer neural network with just one neuron, the simplest type of neural networks and the building block for more complex models. This NN consists of one neuron with input features, weights associated with these features, a bias, and an activation function to make predictions, see figure 1. Specifically, we will focus on a binary classification problem where the neuron uses the sigma function as its activation function, which is defined as

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

And we measure the log loss:

$$\mathcal{L}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

1. Implement a simple 1-layer 1-neuron Neural Network model in Python with the following:
   - Function to compute the output of the NN given inputs. The function should also have access to the weights and bias, to make the prediction.
   - Function which given the data trains the model. The training function updates weights (w) and bias (b) using gradient descent to minimize the loss function. The update rules for w and b are derived from the gradient of the loss.

[1] Note that evaluating LLMs as models on task data is not considered "AI assistance" and is not restricted or affected by the rules here.

[2] If multiple version have been used throughout the assignment, list all of them. In the ChatGPT web interface, the version is specified at the bottom of the screen, under the text input field.

Given the predicted output $\hat{y} = \sigma(w \cdot x + b)$ and true label y, the partial derivatives of the loss $\mathcal{L}$ with respect to w and b consider the derivative of the sigmoid function, $\sigma'(z) = \sigma(z)(1 - \sigma(z))$, leading to:

For weights: $\Delta w = \eta(y - \hat{y})\hat{y}(1 - \hat{y})x$
For bias: $\Delta b = \eta(y - \hat{y})\hat{y}(1 - \hat{y})$

The weights and bias are updated as:
$w = w + \Delta w$
$b = b + \Delta b$
Where η is the learning rate.

- Initialize the weights randomly.
- You should not use a ready made model from a library like sklearn.
- Implement it as an object like so:

```
class SingleLayerNN:
    def __init__(self, input_dim):
        self.weights = ...
        self.bias = ...

    def predict(self, x):
        ...

    def train(self, X, y, epochs):
        ...
```

2. Apply your model on a simple synthetic dataset:

   - Generate a synthetic dataset with 2D features that are linearly separable. You may use this code:

```
import numpy as np
# Generate synthetic data for two classes
n_samples = 100
features_class_0 = np.random.randn(n_samples, 2) + [2, -2]  # Class 0
features_class_1 = np.random.randn(n_samples, 2) + [-2, 2]  # Class 1

```

   - Train your NN (10 epochs) on this dataset, and plot the decision boundary before and after training. Hint: We can leverage the fact that the decision boundary occurs where the output of the network (prior to applying the activation function) is 0. That is, where $w \cdot x + b = 0$. For a 2D dataset, this translates to finding the line $x_2 = (-w_1/w_2)x_1 - (b/w_2)$. You may use this function for plotting:

```
def plot_decision_boundary(nn, X, y, title="Decision Boundary"):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x_values = np.linspace(x_min, x_max, 300)

    """
    TODO: Calculate the y_values for the decision boundary line based on the
     model's weights, the x_values and the model bias.
    """

    y_values =

    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor="k")
    plt.plot(x_values, y_values, label="Decision Boundary")
    plt.xlim(x_min, x_max)
    plt.ylim(X[:, 1].min() - 1, X[:, 1].max() + 1)
    plt.title(title)
    plt.legend()
    plt.show()

```

3. Discuss the following:

   - How did you choose the learning rate.
   - What happens if the dataset is not linearly separable? How does the NN behave, and why? What can we do to mitigate this issue?

Deliverables. 1) A brief discussion of your implementation/results. 2) A plot each of the decision boundary (and datapoints) before and after training. 3) Discussion on the impact of learning rate and the behavior of perceptron on non-linearly separable datasets. What can we do to mitigate this?

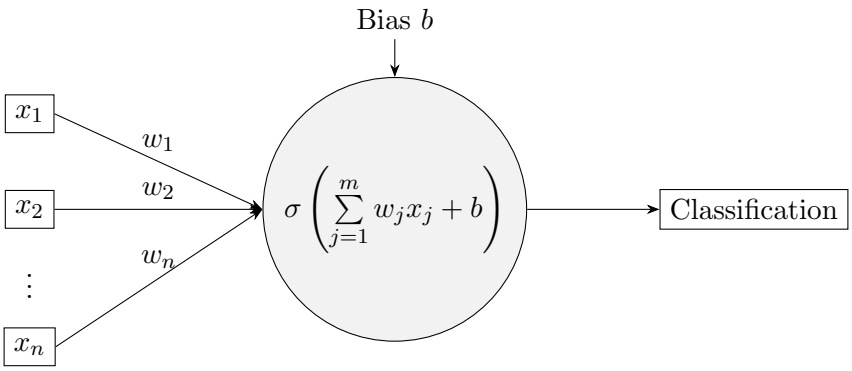

Figure 1: Single-Layer Neural Network with one neuron.

Backpropogation

Exercise 2 (Backpropagation)**.** In traditional machine learning algorithms (like linear regression, nearest neighbors, or k-means clustering) and in the previous exercise computing gradients manually is feasible. However, the training procedure of neural networks is often considerably more complex, making it impractical to derive gradients of the loss function with respect to all parameters. Recognizing this challenge, Geoffrey Hinton introduced the backpropagation algorithm, which updates weights by layers separately through backward propagation, significantly accelerating neural network training [2]. We basically decompose the derivative using the chain rule and compute the gradient step-by-step going backwards through a computational graph. To understand how this works, your task is to compute gradients for intermediate and leaf nodes in the computational graph of a logistic regression model. The graph is shown in Figure 2.

1. Calculate and report the local gradients, indicated with a red question mark, based on the inputs and outputs/activations of each operation (values in blue) and the final gradient (which is equal to 1.00). For each unique operation, also write down the function and its derivative (which you use to compute the local gradients). E.g., for a "$+c$" operation, i.e. a function $f(x) = x + c$ with constant c, the derivative is $\frac{\partial f}{\partial x} = 1$.

2. What patterns do you see for backpropagation through addition and multiplication operations, i.e., is there a specific way gradients are routed through all addition operations (and likewise for multiplication operations)?

3. Assume we are running gradient descent on this computational graph. Report how the weights $\mathbf{w} = (w_0, w_1, w_2) = (1.00, 0.50, 0.50)$ would change after one update step of gradient descent with learning rate $\eta = 0.1$. Do we need the local gradients of our input features x_1 and x_2?

Deliverables. 1) Local gradients of the computational graph and gate functions and function derivatives used to compute them, 2) Discussion of patterns in backpropagation, 3) updated weights w and single-sentence answer whether local gradients are needed.

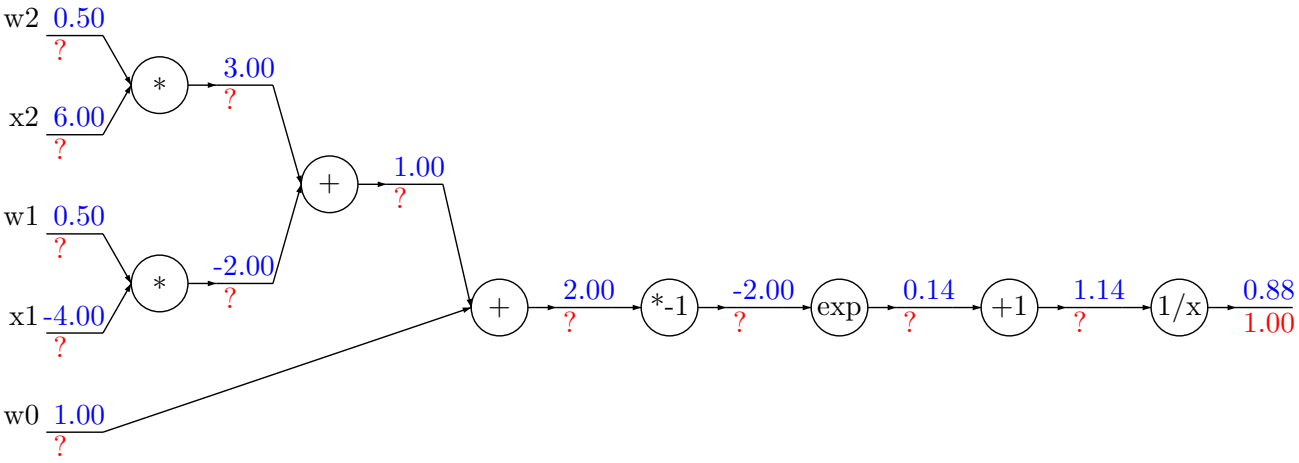

Figure 2: Computational graph for logistic regression model with $\mathbf{w} = (w_0, w_1, w_2) = (1.00, 0.50, 0.50)$ and input features $x_1 = -4.00$ and $x_2 = 6.00$. The sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ is decomposed into four operations.

Note: When reporting the local gradients, you can re-use our computational graph for your report by replacing the question marks in the `computational_graph.svg` file (the file has a lot of numbers, but you can search using `cmd/ctrl + f`) and using the `svg` package for LaTeX. You can also draw your own graph or simply report the values in text form as long as it is clear which reported value corresponds to which local gradient in the graph.

Natural Language Processing

Exercise 3 (Word Embeddings)**.** As you have probably suspected, the input features to a neural network have to be numerical. However, what if our data is text? Then we will need to find some numerical representation of the text. In this exercise, we will explore word embeddings, which are a type of word representation (or word vectors) that allows words with similar meanings to have a similar numerical representation. Word embeddings are foundational in natural language processing (NLP) and enable machines to understand the semantic and syntactic similarity between words. The exercise will involve working with pre-trained word embeddings, understanding their geometric properties, and uncovering biases within them.

1. **Exploring Word Embeddings**:

   - Load a pre-trained word embedding model (GloVe). You should use the Gensim library in Python to load a pre-trained model. Here is an example code snippet to load the Google News Word2Vec model:

     ```
     import gensim.downloader as api
     # Download pre-trained GloVe word vectors
     glove_vectors = api.load("glove-wiki-gigaword-100")
     ```

   - Find the most similar words for a given set of words: ”computer”, ”laptop”, ”queen”, ”king”. Briefly discuss the results.

     You may use:

     ```
     glove_vectors.most_similar(word, topn=5)
     ```

     which returns the 5 most similar words and their similarity values, as a list of pairs:

     ```
     [(similar_word1, similarity), (similar_word2, similarity)...]
     ```

   - Explore vector arithmetic with word embeddings by finding the most similar words to the result of the vector operations:

```
"king" - "man" + "woman"
"vehicle" - "computer" + "laptop"
```

2. **Investigating Societal Biases**:

   - Demonstrate societal biases encoded in word embeddings by comparing the similarities between 3 professions ("engineer", "nurse", "scientist") and names ("james", "emily", "mohammed", "ling", "juan", "fatima"). Use cosine similarity to quantify these relationships.
   - Use PCA to reduce the dimensions of the following word embeddings to 2D: ["king", "queen", "man", "woman", "nurse", "engineer"]. Plot the words in a scatter plot.

     You may retrieve the vectors by:

```
words = ["king", "queen", "man", "woman", "nurse", "engineer"]
word_vectors = [glove_vectors[word] for word in words]
```

3. **Discussion**: Reflect on the implications of these biases in applications that use word embeddings, such as search engines and resumé screening systems.

Deliverables. 1) Brief description of word similarity in the embedding for the given words. A brief comment on what you found doing vector arithmetics on the words. 2) Your plot and a brief description/analysis. 3) A brief analysis of societal biases encoded in word embeddings and their implications.

Note: For loading pre-trained models and performing calculations, you will need to install additional Python libraries like 'gensim'. This exercise aims to provide hands-on experience with word embeddings and encourage critical thinking about the ethical considerations in AI.

Exercise 4 (Text Prediction)**.** In this exercise, we will delve into the realm of natural language processing (NLP) by exploring various methods for predicting text continuations. You will work with a specific text corpus, "The Adventures of Sherlock Holmes" by Arthur Conan Doyle, and apply different techniques to predict the continuation of given sentence prefixes. The objective is to compare the effectiveness of simple statistics, n-gram models, and a pre-trained language model in generating text continuations.

1. Implement a method to predict the most likely next word for a given prefix by examining the last word in the prefix, and predicting the most likely next word, frequency wise (with regards to the given text_corpus).

   These are the prefixes you need to test on:

   ["It is not for me to", "Sherlock Holmes is", "The mystery of"

   You will need to tokenize the text. You can for example do so like this:

```
import re

# We split each word and punctuation into a separate token
tokenized_text = re.findall(r"[\w']+|[.,!?;]", text.lower())

# We remove punctuation etc.
words = [word for word in tokenized_text if word.isalpha()]
```

2. Develop a unigram, bigram and a trigram model.

3. Use a pre-trained language model (gpt-2), to predict the continuation of the same sentence prefixes. You may make a copy of this notebook and use Google Colab for convenience. To access the notebook click here: IDS-A5Ex4-Notebook.

4. Discussion:

- Compare the approaches based on the predictions made for each prefix, discussing the advantages and limitations of each method.
- Consider the implications of each method's predictions in terms of coherence, relevance, and creativity.

Deliverables:

- 1) The methodology used for each prediction technique.
- 2) A comparison of the text predictions made by the simply frequency approach, n-gram models, and the pre-trained language model.

Computer Vision

Exercise 5 (Image Classification)**.** In this exercise, you will go hands on with a neural network. You are tasked with training, validating and testing a ResNet (short for Residual Network) image classification model [1] on the custom FashionMNIST dataset described below. ResNet is based on the idea of residual learning, which allows for very deep neural networks to be trained successfully without encountering the vanishing gradient problem.

1. (Create a copy of this Google Colab notebook by clicking here: IDS-A5Ex5-Notebook.)
2. Use the Colab notebook to train a `FashionMnistResNet18` on our custom FashionMNIST dataset with the hyper-parameters batch size $B = 256$, number of epochs $E = 100$, learning rate $\eta = 0.1$. Evaluate your model every 100 training steps.
3. Create a plot containing the training and validation loss curves; use the running mean loss values `mean_train_loss` and `mean_eval_loss` for this. Also create a second plot that shows the development of the training and validation accuracies throughout the training process. Tip: Make use of the `metrics_dict` returned by the training function.
4. Describe the plots created in step 3. Based on your observations, would you recommend changing the number of training epochs? If so, why?
5. Report the test accuracy of your model at the best validation loss and plot the confusion matrix on the test dataset. What are your takeaways from the confusion matrix?

Deliverables. 1) a plot containing the training and validation loss curves 2) a plot showing the development of the training and validation accuracies throughout the training process 3) a description of the plots 4) report test accuracy at best validation loss 5) plot and discuss the confusion matrix on the test dataset.

Appendix: Data material

FashionMNIST

FashionMNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. The labels are:

- 0 T-shirt/top
- 1 Trouser

Figure 3: Fashion MNIST dataset

- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker
- 8 Bag
- 9 Ankle boot

For the assignment, we provide three custom splits: `fashion_mnist_training.pt` (5000 examples), `fashion_mnist_validation.pt` (1000 examples), and `fashion_mnist_test.pt` (1000 examples). They can be loaded via our custom PyTorch dataset found in the corresponding notebook.

Each row is a separate image. Column 1 is the class label, while the remaining columns are pixel numbers (784 total). Each value is the darkness of the pixel (1 to 255).

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.