Problem set 1
Deep Learning E1394

Gresa Smolica (224725)
Steven Kerr (211924)

Out on Sept 18
Due on Oct 2, 23:59

Submit your written answers as a pdf typed in LaTeXtogether with your code. Submit one answer per group (as assigned on Moodle) and include names of all group members in the document. Round answers to two decimal places as needed. Include references to any external sources you have consulted (points are deducted if those were used but not cited). See "Submission" at the bottom of the problem set for more details on how to submit using Github classroom.

1 Theoretical part

These are some recap and refresher problems to get you up to speed with the mathematics and statistical learning that is behind deep learning, as well as problems that help you work out how neural networks are trained.

1.1 Optimization

Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by $f(x, y) = x^2y + xy^2 - 6xy$. Identify and classify all critical points of f.

We have $\nabla f(x, y) = (2xy + y^2 - 6y,\ x^2 + 2xy - 6x)$, therefore any critical points of f must satisfy:

$2xy + y^2 - 6y = y(2x + y - 6) = 0$ and $x^2 + 2xy - 6x = x(x + 2y - 6) = 0$

If $x = 0$, $y = 0$, then $\nabla f(x, y) = 0$, so (0,0) is a critical point.

If $x \neq 0$, $y = 0$, then we need that $x + 2y - 6 = x - 6 = 0$ and so (6, 0) is a critical point.

If $x = 0$, $y \neq 0$, then we need that $2x + y - 6 = y - 6 = 0$ and so (0, 6) is a critical point.

Lastly, if $x \neq 0$, $y \neq 0$, then we need that $2x + y - 6 = 0$ and $x + 2y - 6 = 0$. This means that $x = 3 - \frac{1}{2}y = 6 - 2y)$, and hence $\frac{3}{2}y = 3 \leftrightarrow y = 2$ and $x = 2$, thus (2, 2) is our last critical point.

The Hessian of f is given by $H_f(x, y) = \begin{bmatrix} 2y & 2x + 2y - 6 \\ 2x + 2y - 6 & 2x \end{bmatrix} = \begin{bmatrix} 2y & 2(x + y - 3) \\ 2(x + y - 3) & 2x \end{bmatrix}$.

Thus $H_f(0, 0) = \begin{bmatrix} 0 & -6 \\ -6 & 0 \end{bmatrix}$. Since det $H_f(0, 0) = -36 < 0$, we have that (0, 0) is a saddle point.

And $H_f(6, 0) = \begin{bmatrix} 0 & 6 \\ 6 & 12 \end{bmatrix}$. Since det $H_f(6, 0) = -36 < 0$, (6,0) is a saddle point as well.

And $H_f(0, 6) = \begin{bmatrix} 12 & 6 \\ 6 & 0 \end{bmatrix}$. Since det $H_f(6, 0) = -36 < 0$, (0,6) is a saddle point as well.

Lastly, $H_f(2, 2) = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$. Since det $H_f(2, 2) = 12 > 0$ and $\frac{\partial^2 f(2,2)}{\partial x^2} = 4 > 0$, we have that (2, 2) is a minimum.

1.2 Activation functions

Compute the gradient of the function

$$f(b, w) = ReLU(b + xw) = \begin{cases} 0 & \text{if } b + xw < 0 \\ b + xw & \text{if } b + xw \geq 0 \end{cases} \tag{1}$$

with respect to the parameters b and w, with $x, b, w \in \mathbb{R}$.

We start by taking the partial derivative with respect to b:

If $b + xw < 0 \Rightarrow f(b, w) = 0 \Rightarrow \frac{\partial f}{\partial b} = 0$

If $b + xw \geq 0 \Rightarrow f(b, w) = b + xw \Rightarrow \frac{\partial f}{\partial b} = 1$

$$\frac{\partial f}{\partial b} = \frac{\partial}{\partial b}(ReLU(b + xw)) = \begin{cases} 0, & \text{if } b + xw < 0. \\ 1, & \text{if } b + xw \geq 0. \end{cases} \quad (2)$$

Next, we take the partial derivative with respect to w:

If $b + xw < 0 \Rightarrow f(b, w)) = 0 \Rightarrow \frac{\partial f}{\partial b} = 0$

If $b + xw \geq 0 \Rightarrow f(b, w) = b + xw \Rightarrow \frac{\partial f}{\partial b} = x$

$$\frac{\partial f}{\partial w} = \frac{\partial}{\partial w}(ReLU(b + xw)) = \begin{cases} 0, & \text{if } b + xw < 0. \\ x, & \text{if } b + xw \geq 0. \end{cases} \quad (3)$$

$$\nabla f(b, w) = (\frac{\partial f}{\partial b}, \frac{\partial f}{\partial w}) = \begin{cases} (0, 0), & \text{if } b + xw < 0. \\ (1, x), & \text{if } b + xw \geq 0. \end{cases} \quad (4)$$

1.3 Overfitting

This question will test your general understanding of overfitting as it relates to model complexity and training set size. Consider a continuous domain and a smooth joint distribution over inputs and outputs, so that no test or training case is ever duplicated exactly. *The graphs in your solutions should be drawn on paper and then included as a picture. In your answers describe how how your graphs should be read.*

1.3.1 Error rate versus dataset size

Sketch a graph of the typical behavior of training error rate (y-axis) as a function of training set size (x-axis). Add to this graph a curve showing the typical behavior of test error rate versus training set size, on the same axes. (Assume that we have an infinite test set drawn independently from the same joint distribution as the training set). Indicate on your y-axis where zero error is and draw your graphs with increasing error upwards and increasing training set size rightwards.

As shown in the graph, the **training error rate** initially starts relatively low because the model is overfitting to the training data. At this point, the model has not the caputred underlying patterns in the data, but rather has learned to perform well when applied exclusively to the data upon which it has been trained. As the dataset size increases, the training error gradually increases as well as the model is not able to rely on overfitting to boost performance as easily. Eventually, once the dataset is sufficiently large, the training error rate will

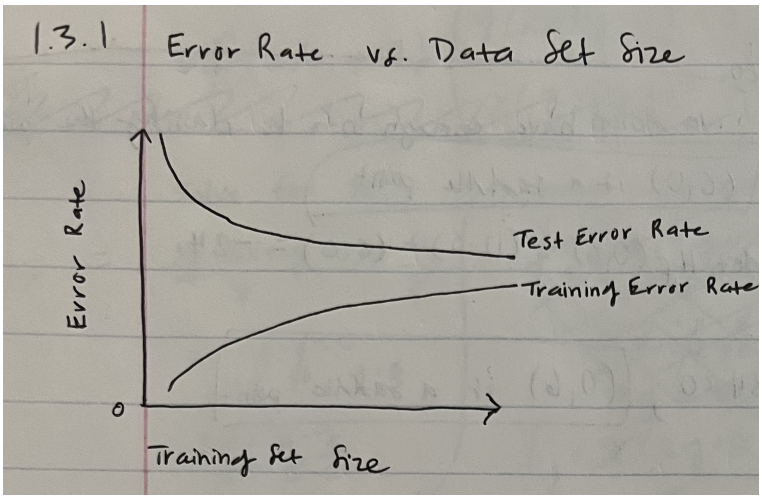

Figure 1: Error rate vs. Dataset size

plateau. At this point, the model has learned to rely on the patterns present in the data, so despite the fact that the training error rate increases with dataset size, the model is able to generalize better to previously unseen data.

The **test error rate**, on the other hand, initially starts relatively high because the model cannot capture the underlying patterns in the data with so few examples. As the dataset size increases, the test error gradually decreases because the model is able to learn more from the data thus improving performance. Eventually, once the dataset is sufficiently large, the test error rate will plateau at a relatively low value. At this point, the model has learned most of the patterns present in the data and additional data points result in diminishing returns when it comes to further reducing the test error.

1.3.2 Error rate versus model complexity

For a fixed training set size, sketch a graph of the typical behavior of training error rate (y-axis) versus model complexity (x-axis). Add to this graph a curve showing the typical behavior of the corresponding test error rate versus model complexity, on the same axes (again on an IID infinite test set). Show where on the x-axis you think is the most complex model that your data supports (mark this with a vertical line). Choose the x-range so that this line is neither on the extreme left nor on the extreme right. Indicate on your vertical axis where zero error is and draw your graphs with increasing error upwards and increasing complexity rightwards.

As shown in the graph, the level of model complexity can impact the training error rate and test error rate differently. With respect the **training error**

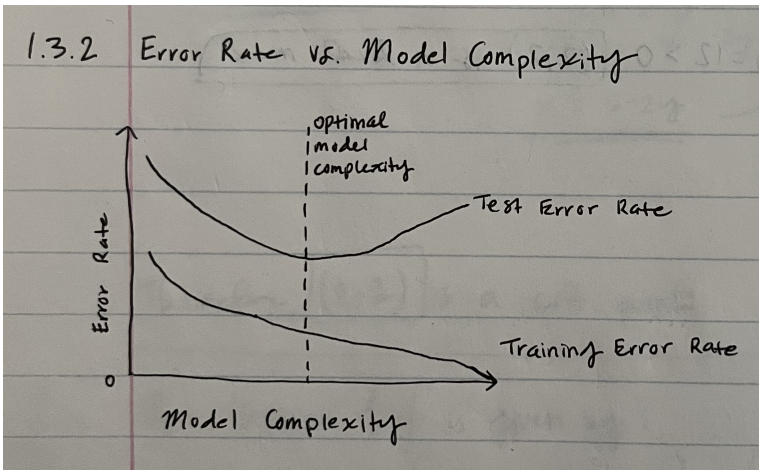

Figure 2: Error rate vs. Model complexity

rate, increasing model complexity can gradually reduce this error. At first, model performance is genuinely improved as we reach a level complexity that better suits our data. However, past a certain, further increases to the model complexity eventually result in overfitting. A model with a high enough level of complexity can even reduce the training error rate to zero as it perfectly learns the training data.

At first, the **test error rate** follows a similar trajectory, gradually decreasing as we increase model complexity insofar as the level of model complexity better suits our data. Howewever, after a certain point, the level of model complexity will no longer impact the generalizability of the model and will eventually lead to increases in the test error rate when applied to previously unseen data.

1.3.3 Training epochs

Use a similar graph to illustrate the error rate as a function of training epochs in neural networks. One of the commonly used regularization methods in neural networks is early stopping. Describe (also using the graph) how early stopping is applied, and argue qualitatively why (or why not) early stopping is a reasonable regularization metric.

Initially, both the **training error rate** and **test error rate** will gradually decrease as the number of training epochs increase. However, after a certain point their trajectories diverge as the model begins to overfit to the training data. While the training error rate continues to gradually decrease with the number of training epochs, there is a point where the test error rate plateaus before eventually increasing. In such instances, it may be advisable to apply early stopping when training the model. Early stopping functions by measur-

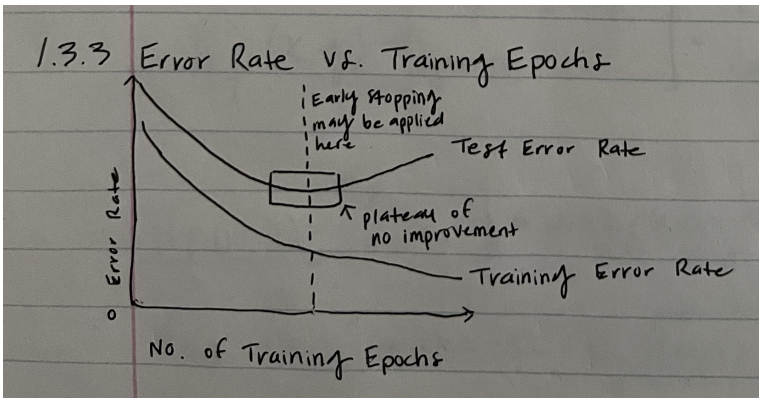

Figure 3: Error rate vs. Training epochs

ing the loss value of the model performance on the validation dataset for each additional epoch and stopping once training epochs are shown to have either no impact or a negative impact on the model's performance. The risk with implementing early stopping is that the model may encounter a local minimum as opposed to a global one, in which case early stopping may prevent the model from reaching its highest potential performance. As such, data scientists should exercise discretion when choosing to apply early stopping to their models.

1.4 Capacity of neural network

We have a single hidden-layer neural network with two hidden units, and data with 2 features $X = (x_1, x_2)$. Design an activation function $g(\cdot)$ applied in the hidden layer, and an output activation function $o(\cdot)$ applied in the output layer, as well as a set of values of the weights that will create a logistic regression model where the input to the logistic function is of the form: $\beta_0 + \beta_1 x_1 x_2$.

For the **activation function in the hidden layer** $g(\cdot)$, we can use the quadratic input activation function: $g(\cdot) = (a(x_1, x_2))^2$

For the **output activation function in the output layer** $o(\cdot)$, we can use the sigmoid activation function: $o(z) = \sigma(z) = \frac{1}{1+e^{-z}}$

We define the weights and biases for the **hidden layer** as follows:

$b_1^{[1]} = 0$
$b_2^{[1]} = 0$
$w_{11}^{[1]} = 1$

$w_{12}^{[1]} = 1$
$w_{21}^{[1]} = 1$
$w_{22}^{[1]} = -1$

We define the weights and biases for the **output layer** as follows:

$b^{[2]} = \beta_0$
$w_1^{[2]} = \frac{1}{4}\beta_1$
$w_2^{[2]} = -\frac{1}{4}\beta_1$

With these weights and activation functions, the **output of the hidden layer** will be:

$$h_1(x_1, x_2) = g(b_1^{[1]} + \sum_j^2 w_{1j}^{[1]} x_j) = g(b_1^{[1]} + w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2) = g(x_1 + x_2)$$
$$h_2(x_1, x_2) = g(b_2^{[1]} + \sum_j^2 w_{2j}^{[1]} x_j) = g(b_2^{[1]} + w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2) = g(x_1 - x_2)$$

And the **output of the output layer** will be:

$$\begin{aligned}
\mathrm{f}(\mathrm{x}_1, x_2) &= o(b^{[2]} + \sum_i^2 w_i^{[2]} h_i) \\
&= o(b^{[2]} + w_1^{[2]} \cdot h_1 + w_2^{[2]} \cdot h_2) \\
&= o(b^{[2]} + w_1^{[2]} \cdot g(x_1 + x_2) + w_2^{[2]} \cdot g(x_1 - x_2)) \\
&= o(\beta_0 + \frac{1}{4}\beta_1 \cdot (x_1 + x_2)^2 - \frac{1}{4}\beta_1 \cdot (x_1 - x_2)^2) \\
&= o(\beta_0 + \frac{1}{4}\beta_1 \cdot (x_1^2 + 2x_1x_2 + x_2^2) - \frac{1}{4}\beta_1 \cdot (x_1^2 - 2x_1x_2 + x_2^2)) \\
&= o(\beta_0 + \frac{1}{4}\beta_1 x_1^2 + \frac{1}{2}\beta_1 x_1 x_2 + \frac{1}{4}\beta_1 x_2^2 - \frac{1}{4}\beta_1 x_1^2 + \frac{1}{2}\beta_1 x_1 x_2 - \frac{1}{4}\beta_1 x_2^2) \\
&= o(\beta_0 + \beta x_1 x_2) \\
&= \sigma(\beta_0 + \beta x_1 x_2) \\
f(x_1, x_2) &= \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 x_2)}}
\end{aligned}$$

1.5 Neural network theory

Derive the weight updates in gradient descent for a neural network with 2 hidden layers (superscripts [1] and [2]) that each have $H^{[1]}$ and $H^{[2]}$ hidden units respectively. The output layer has superscript [3], and we want to classify the data into K classes.

The output of the network for classes $k = 1, 2, ..., K$ and one data point $x \in \mathbb{R}^d$ can be written as such:

$$f_k(\theta; X) = o(a_k^{[3]}) = o(b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} h_m^{[2]}) \tag{5}$$

$$h_m^{[2]} = \sigma\left(a_m^{[2]}\right) = \sigma\left(b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]}\right) \tag{6}$$

$$h_i^{[1]} = \sigma\left(a_i^{[1]}\right) = \sigma\left(b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]} x_j\right), \tag{7}$$

where θ indicates the vector of all weights and biases.

While this is typically not recommended, in this exercise we use a a quadratic loss function for classification. We use a softmax activation function at the output layer and a ReLU activation function at the hidden layers. Derive the the weight update for the weights of the first hidden layer $w_{ij}^{[1]}$. Start by stating the gradient descent weight update for the $(r+1)^{th}$ iteration as a function of the $(r)^{th}$ iteration, and then compute the partial derivative needed in the update.

Show all the steps in your derivation. You may use the derivatives for activation functions introduced in class. There is no need to show the derivations of those.

The weight update is given by:

$w_{ij}^{[1](r+1)} = w_{ij}^{[1](r)} - \eta \frac{\partial L}{\partial w_{ij}^{[1]}}$, where η represents the learning rate, $\frac{\partial L}{\partial w_{ij}^{[1]}}$ is the partial derivative of the loss with respect to $w_{ij}^{[1]}$, and $L = (y - f_k)^2$

$$\frac{\partial L}{\partial w_{ij}^{[1]}} = \frac{\partial L}{\partial f_k} \frac{\partial f_k}{\partial a_k^{[3]}} \frac{\partial a_k^{[3]}}{\partial h_m^{[2]}} \frac{\partial h_m^{[2]}}{\partial a_m^{[2]}} \frac{\partial a_m^{[2]}}{\partial h_i^{[1]}} \frac{\partial h_i^{[1]}}{\partial a_i^{[1]}} \frac{\partial a_i^{[1]}}{\partial w_{ij}^{[1]}}$$

Next we calculate the derivatives:

$$\frac{\partial L}{\partial f_k} = -2(y - f_k)$$

$$\frac{\partial f_k}{\partial a_k^{[3]}} = \frac{\partial o\left(a_k^{[3]}\right)}{\partial a_k[3]} \Rightarrow o\left(a_k^{[3]}\right)\left(\delta - o\left(a_k^{[3]}\right)\right)$$

$\frac{\partial a_k^{[3]}}{\partial h_m^{[2]}} = \frac{\partial}{\partial h_m^{[2]}}\left[b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} h_m^{[2]}\right] = \omega_{km}^{[3]}$

$\frac{\partial h^{[2]}}{\partial a_m^{[2]}} \Rightarrow \sigma\left(a_m^{[2]}\right) = \text{ReLU}\left(a_m^{[2]}\right) \Rightarrow \frac{\partial}{\partial a_m^{[2]}} \text{ReLU}\left(a_m^{[2]}\right) = \begin{cases} 1 \text{ if } a_m^{[2]} > 0 \\ 0 \text{ otherwise} \end{cases}$

$\frac{\partial a_m^{[2]}}{\partial h_i^{[i]}} \Rightarrow \frac{\partial}{\partial h_i^{[1]}}\left[b_m^{[1]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]}\right] = w_{mi}^{[2]}$

$\frac{\partial h_i^{[1]}}{\partial a_i^{[1]}} \Rightarrow \sigma\left(a_i^{[1]}\right) = \text{ReLU}\left(a_i^{[1]}\right) \Rightarrow \frac{\partial}{\partial a_i^{[i]}}\left[b_i^{[1]} + \sum_{i=1}^{d} w_{ij}^{[1]} x_j\right] = \begin{cases} 1 \text{ if } a_i^{[1]} > 0 \\ 0 \text{ otherwise} \end{cases}$

$\frac{\partial a_i^{[1]}}{\partial \omega_{ij}^{[1]}} \Rightarrow \frac{\partial}{\partial w_{ij}^{[1]}}\left[b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]} x_j\right] = x_j$

Putting all the terms partial derivatives into the chain rule formula, we get:

$$\frac{\partial L}{\partial w_{ij}^{[1]}} = -2(y-f)\cdot o\left(a_k^{[3]}\right)\left(\delta - o\left(a_k^{[3]}\right)\right)\cdot w_{km}^{[3]} \cdot \begin{cases} 1 \text{ if } a_m^{[2]} > 0 \\ 0, \text{otherwise} \end{cases} \cdot w_{mi}^{[2]} \cdot \begin{cases} 1 \text{ if } a_i^{[1]} > 0 \\ 0, \text{otherwise} \end{cases}$$

As such, our gradient update is as follows:

$$\mathrm{w}_{ij}^{(r+1)} = w_{ij}^{(r)} - \eta\left(\frac{\partial L(f_k, y)}{\partial w_{ij}}\right)^{(r)}$$

$$= w_{ij}^{(r)} - \eta\left(-2(y-f)\cdot o\left(a_k^{[3]}\right)\left(\delta - o\left(a_k^{[3]}\right)\right)\cdot w_{km}^{[3]} \cdot \begin{cases} 1 \text{ if } a_m^{[2]} > 0 \\ 0, \text{otherwise} \end{cases} \cdot w_{mi}^{[2]} \cdot \begin{cases} 1 \text{ if } a_i^{[1]} > 0 \\ 0, \text{otherwise} \end{cases}\right)^{(r)}$$

2 Neural network implementation

You will implement different classes representing a fully connected neural network for image classification problems. There are two classes of neural networks: one using only basic packages and one using PyTorch. In addition to that, a third class of neural network has been implemented using Tensorflow and requires some fixing in order to function correctly.
As you work through this problem, you will see how those machine learning libraries abstract away implementation details allowing for fast and simple construction of deep neural networks.

For each approach, a Python template is supplied that you will need to complete with the missing methods. Feel free to play around with the rest, but please do not change anything else for the submission. All approaches will

solve the same classification task, which will help you validate that your code is working and that the network is training properly.

For this problem, you will work with the MNIST dataset of handwritten digits, which has been widely used for training image classification models. You will build models for a multiclass classification task, where the goal is to predict what digit is written in an image (to be precise, this is a k-class classification task where in this case $k = 10$). The MNIST dataset consists of black and white images of digits from 0 to 9 with a pixel resolution of 28x28. Therefore, in a tensor representation the images have the shape 28x28x1. The goal is to classify what digit is drawn on a picture using a neural network with the following characteristics:

- an arbitrary amount of hidden layers, each with arbitrary amount of neurons
- sigmoid activation function for all hidden layers
- softmax activation function for the output layer
- cross entropy loss function. *

** For the implementation from scratch (Part (a)) we use a mean squared error (MSE) loss function. This is not recommended for a classification task, but we use it to simplify the implementation. In the future please consider using cross entropy / log loss instead.*

Figure 4: Example images from MNIST dataset with 28x28 pixel resolution.

The training set with n data points is of the form

$$(x^{(i)}, y^{(i)}),\ i = 1, ..., n, \tag{8}$$

with $y^{(i)} \in \{0, 1\}^k$, where $y_j^{(i)} = 1$ when j is the target class, and 0 otherwise (i.e., the output is one-hot encoded). The softmax output function $\hat{y} = h_\theta(x)$ outputs vectors in $\mathbb{R}^k$, where the relative size of the $\hat{y}_j$ corresponds roughly to how likely we believe that the output is really in class j. If we want the model to predict a single class label for the output, we simply predict class j for which $\hat{y}_j$ takes on the largest value.

Tasks

1. Complete the implementation of the neural network classes marked by TODO comments:
   (a) From scratch (`network_scratch.py`)
   (b) Using PyTorch (`network_pytorch.py`)

A repository with the template files will automatically be created for your team when registering for the GitHub classroom assignment at `https://classroom.github.com/a/XgELWE_C`.

2. In `network_tensorflow.py` a Neural Network class has been implemented using Tensorflow and Keras.
   (a) There are three mistakes in the code preventing the network from working correctly. Comment the mistakes out and repair the class.
   (b) Implement a time-based learning rate class `TimeBasedLearningRate`: it should be initialized with a positive integer as initial learning rate and have the learning rate reduced by 1 at each step, until a learning rate of 1 is reached.
3. In addition to the implementation, show how your networks perform on the MNIST classification task in `MNIST_classification.ipynb`.
   (a) Plot the accuracy of each neural network on the training and on the validation set as a function of the epochs
   (b) Please make sure that the uploaded Notebook shows printed training and validation progress.
   (c) Please add team number, team member names and matriculation numbers as a comment at the top of the notebook.

Please note that the classifier does not need to get anywhere close to 100% accuracy. Especially, with the small amount of training data this is difficult. Instead, the training output should indicate that the model learns something, i.e., the accuracy increases over the epochs on the training data and is significantly better than random guessing.

Sources

To complete the tasks of this problem set, among others, the following sources were consulted:

Platforms: Medium, PyTorch, Toward Data Science, Chat GPT-3

Youtube: 3blue1brown Channel

Specific articles consulted:

- Calleda, C. (n.d.). Finding Optimal Model: Model complexity and Data Size. www.linkedin.com. https://www.linkedin.com/pulse/finding-optimal-model-complexity-data-size-carlo-calledda
- Brownlee, J. (2020). Use Early Stopping to Halt the Training of Neural Networks At the Right Time. MachineLearningMastery.com.

  https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping
- GeeksforGeeks. (2023). Activation functions in Neural Networks. GeeksforGeeks. https://www.geeksforgeeks.org/activation-functions-neural-networks/
- May, M., Bart, A. (2023). *Business Calculus with Excel.* 6.3 Critical Points and Extrema. https://mathstat.slu.edu/may/ExcelCalculus/sec-6-3-CriticalPointsExtrema.html
- PyTorch. (n.d.). Build the Neural Network — PyTorch Tutorials 2.0.1+cu117 documentation. https://pytorch.org/tutorials/beginner/basics/buildmodel tutorial.html
- PyTorch. (n.d.). Defining a Neural Network in PyTorch — PyTorch Tutorials 2.0.1+cu117 documentation. https://pytorch.org/tutorials/recipes/recipes/defining_a_neural_network.html
- Tripathi, A. (2021, December 16). How to Build a Neural Network From Scratch - The Startup - Medium. Medium. https://medium.com/swlh/how-to-build-a-neural-network-from-scratch-b712d59ae641
- Valkov, V. (2021, December 9). Build your first Neural Network in TensorFlow 2 — TensorFlow for Hackers (Part I). Medium. https://towardsdatascience.com/building-your-first-neural-network-in-tensorflow-2-tensorflow-for-hackers-part-i-e1e2f1dfe7a0

Submission

The submission of the whole problem set is done via GitHub classroom. You have to register for the assignment with your GitHub account at `https://classroom.github.com/a/XgELWE_C`. Please make sure that your GitHub account profile includes your real name. When starting the assignment, please create or join a team according to the teams assigned in Moodle (use the exact team name from Moodle).

Please upload / push all solutions to the GitHub repository which was created for your team. Please upload your solutions for the theoretical part (1) as a PDF to GitHub. If you upload multiple PDF files, please indicate in the file name to which subtask they are corresponding (we much prefer one single pdf). For the practical part (2), just push your code changes to the existing files.

Anybody in your team can push as often as they want. Once the deadline has passed, you are not able to push to your repository anymore and all changes

on your **main** branch will be considered for grading. See the README.md in the repository for more details on how to push your changes to GitHub.