

Problem set 1

Deep Learning E1394

Fernanda Ortega
Luke Smith

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

Submit your written answers as a pdf typed in \LaTeX together 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 $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ be defined by $f(x, y) = x^2y + xy^2 - 6xy$. Identify and classify all critical points of f .

$$\begin{aligned}\frac{\partial}{\partial x} (x^2y + xy^2 - 6xy) &= y(2x + y - 6) \\ \frac{\partial}{\partial y} (x^2y + xy^2 - 6xy) &= x(x + 2y - 6)\end{aligned}\tag{1}$$

$$\begin{aligned}y(2x + y - 6) &= 0 \\ x(x + 2y - 6) &= 0\end{aligned}\tag{2}$$

$$(x, y) = (0, 0), (x, y) = (0, 6), (x, y) = (2, 2), (x, y) = (6, 0).\tag{3}$$

$$\begin{aligned}
\frac{\partial^2}{\partial x^2} (x^2y + xy^2 - 6xy) &= 2y \\
\frac{\partial^2}{\partial y \partial x} (x^2y + xy^2 - 6xy) &= 2x + 2y - 6 \\
\frac{\partial^2}{\partial y^2} (x^2y + xy^2 - 6xy) &= 2x
\end{aligned} \tag{4}$$

$$D = \frac{\partial^2 f}{\partial x^2} \frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial y \partial x} \right)^2 = 4xy - 4(x + y - 3)^2 \tag{5}$$

Since $D(0, 0) = -36$ is less than 0, $(0, 0)$ is a saddle point.

Since $D(0, 6) = -36$ is less than 0, $(0, 6)$ is a saddle point.

Since $D(2, 2) = 12$ is greater than 0 and $\frac{\partial^2}{\partial x^2} (x^2y + xy^2 - 6xy) = 4$ (when $(x, y) = (2, 2)$) and is greater than 0, $(2, 2)$ is a local minimum.

Since $D(6, 0) = -36$ is less than 0, $(6, 0)$ is a saddle point.

1.2 Activation functions

Compute the gradient of the function

$$f(b, w) = \text{ReLU}(b + xw) = \begin{cases} 0 & \text{if } b + xw < 0 \\ b + xw & \text{if } b + xw \geq 0 \end{cases} \tag{6}$$

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

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

$$\frac{\partial f}{\partial b} = \begin{cases} 0, & \text{if } b + xw < 0 \\ 1, & \text{if } b + xw \geq 0 \end{cases}$$

$$\frac{\partial f}{\partial w} = \begin{cases} 0, & \text{if } b + xw < 0 \\ x, & \text{if } b + xw \geq 0 \end{cases}$$

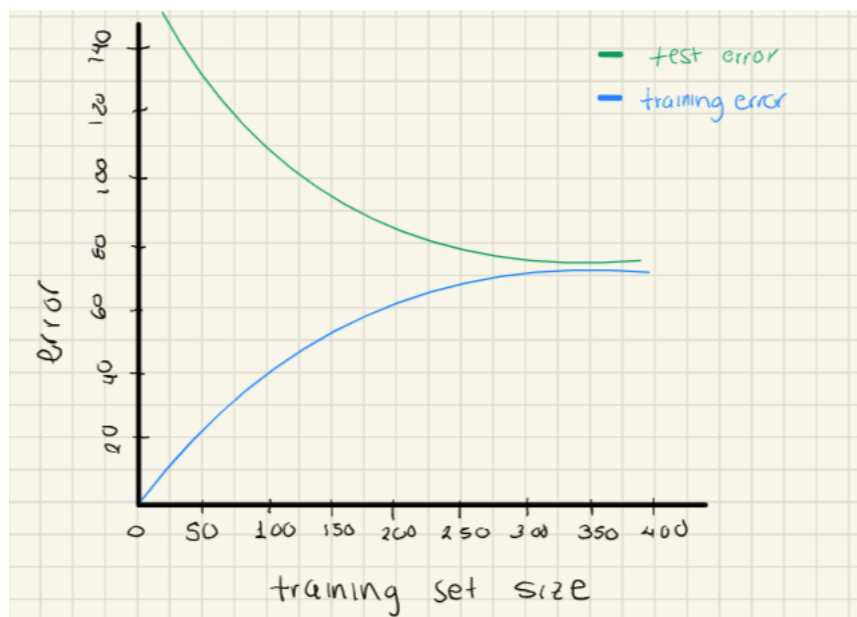
$$\nabla f(b, w) = \begin{cases} 0, & \text{if } b + xw < 0 \\ \begin{bmatrix} 1 \\ x \end{bmatrix}, & \text{if } b + xw \geq 0 \end{cases}$$

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.

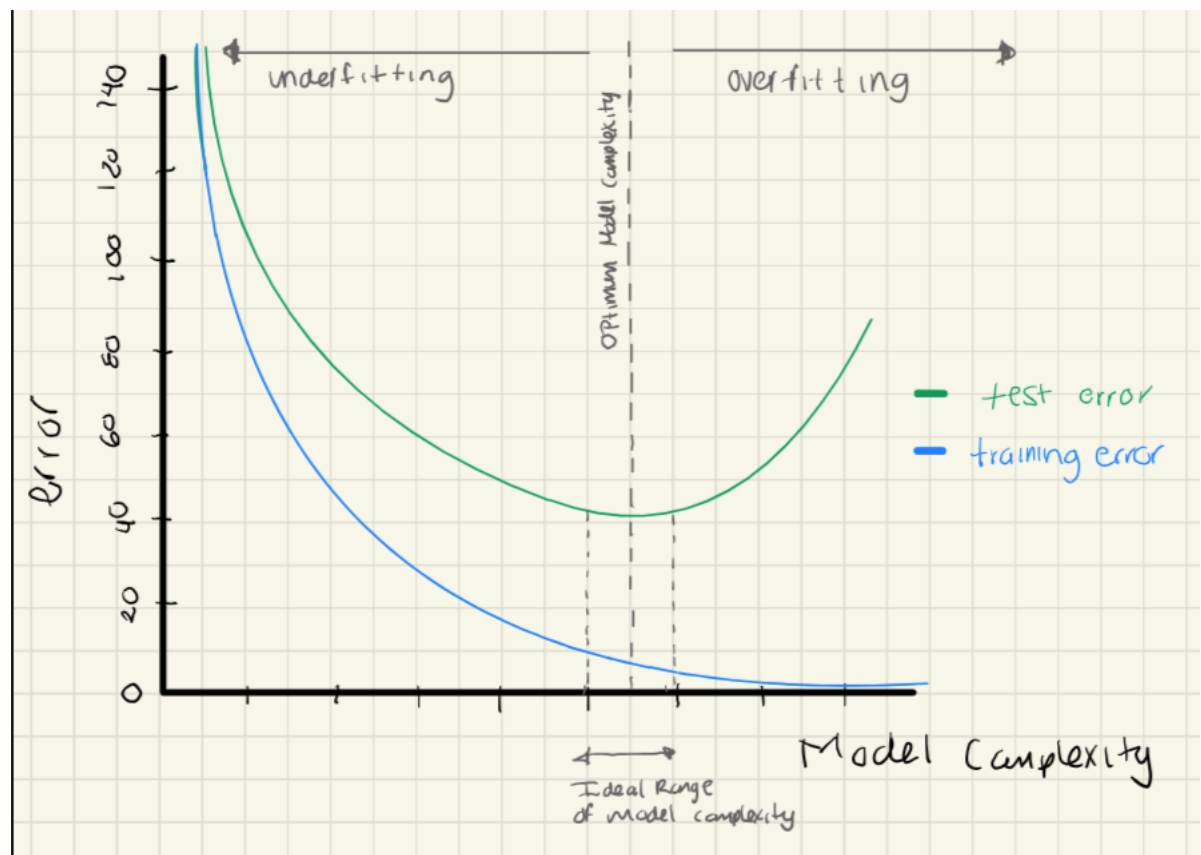


Training error increases with training size from zero to a stable value. Test error decreases with training size from a large value to the stable value.

Source: When you plot the test and training error vs the data size, is it expected the test error is much more dispersed? (n.d.). Quora. Retrieved October 2, 2023, from <https://www.quora.com/When-you-plot-the-test-and-training-error-vs-the-data-size-is-it-expected-the-test-error-is-much-more-dispersed>

1.3.2 Error rate versus model complexity

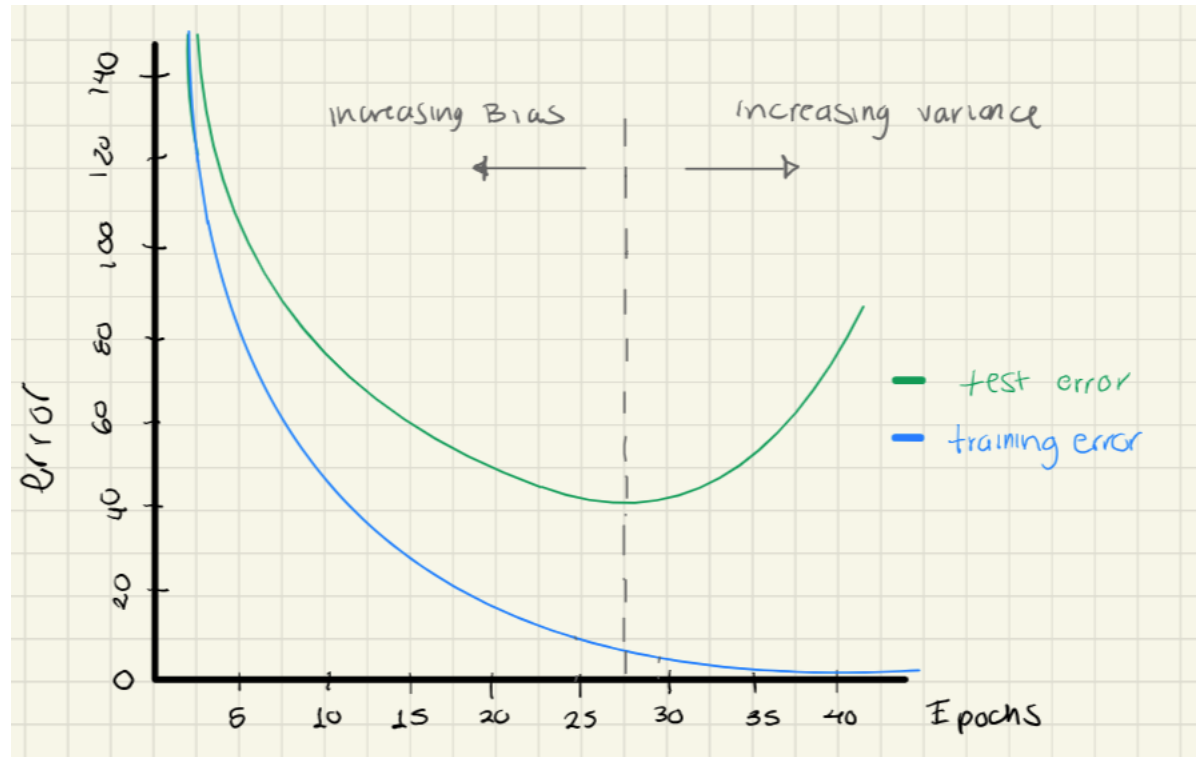
For a fixed training set size, sketch a graph of the typical behavior of training error rate (y-axis) versus model complexity (x-axis). Add to this graph a curve showing the typical behavior of the corresponding test error rate versus model complexity, on the same axes (again on an IID infinite test 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.



Source: Sajee, A. (2020, May 30). Model Complexity, Accuracy and Interpretability. Medium. <https://towardsdatascience.com/model-complexity-accuracy-and-interpretability-59888e69ab3d>

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.



The training process is stopped, when the performance of the model on the validation dataset starts to degrade (e.g. loss begins to increase, or accuracy begins to decrease). Thus, it stops training as soon as the validation error reaches a minimum.

Early stopping helps prevent the model from overfitting the training data by halting training when it starts to generalize poorly to the validation set. This can lead to a model with better generalization performance on unseen data. In other words, this regularization technique increases the bias and reduces the variance of the model.

Source: Chen, B. (2020, August 3). A Practical Introduction to Early Stopping in Machine Learning. Medium. <https://towardsdatascience.com/a-practical-introduction-to-early-stopping-in-machine-learning-550ac88bc8fd>

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: $\beta_0 + \beta_1 x_1 x_2$.

Activation Function Choice

The identity function, $g(z) = z$, is an acceptable choice in activation function for the hidden layers. It allows the weighted sum of inputs to pass through the hidden layer without any transformation. This aligns with the goal of modeling the linear and interaction terms in the logistic regression equation provided initially.

By using the identity function, we simplify the hidden layer's role to capture the linear and interaction terms in the logistic regression model. The identity function is linear and introduces no additional non-linearity.

Activation Function for Output Layer

For the output layer, denoted as $o(\cdot)$, we need an activation function that aligns with the requirements of logistic regression. The sigmoid activation function, denoted as $o(z) = \frac{1}{1+e^{-z}}$, is a solid choice for binary classification tasks. It is able to map the network's output to the range $[0, 1]$, representing probabilities, which is consistent with logistic regression.

Input Features: x_1 and x_2

Activation Function for Hidden Layer ($g(\cdot)$):

$$g(z) = z$$

Activation Function for Output Layer ($o(\cdot)$):

$$o(z) = \frac{1}{1 + e^{-z}}$$

Weight from x_1 to Neuron 1:	$w_{11}^{(1)} = 0.7$
Weight from x_2 to Neuron 1:	$w_{12}^{(1)} = -0.4$
Weight from x_1 to Neuron 2:	$w_{21}^{(1)} = -0.3$
Weight from x_2 to Neuron 2:	$w_{22}^{(1)} = 0.6$
Bias for Neuron 1:	$b_1^{(1)} = -0.2$
Bias for Neuron 2:	$b_2^{(1)} = 0.1$

Function for Hidden Layer 1 with Two Hidden Units:

Neuron 1: Activation of Neuron 1 = $(0.7 \cdot x_1) + (-0.4 \cdot x_2) + (-0.2)$

Neuron 2: Activation of Neuron 2 = $(-0.3 \cdot x_1) + (0.6 \cdot x_2) + (0.1)$

Weight from Neuron 1 to Output Neuron:	$w_1^{(2)} = 1.2$
Weight from Neuron 2 to Output Neuron:	$w_2^{(2)} = -1.0$
Bias for the Output Layer:	$b^{(2)} = -0.1$

Function for the Output Layer:

Output Neuron: $(1.2 \cdot ((0.7 \cdot x_1) + (-0.4 \cdot x_2) + (-0.2))) + (-1.0 \cdot ((-0.3 \cdot x_1) + (0.6 \cdot x_2) + (0.1))) + (-0.1)$

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 $k = 1, 2, \dots, K$ and one data point $x \in \mathbb{R}^d$ can be written as such:

$$f_k(\theta; X) = o(a_k^{[3]}) = o(b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} h_m^{[2]}) \quad (7)$$

$$h_m^{[2]} = \sigma(a_m^{[2]}) = \sigma\left(b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]}\right) \quad (8)$$

$$h_i^{[1]} = \sigma(a_i^{[1]}) = \sigma\left(b_i^{[1]} + \sum_{j=1}^d w_{ij}^{[1]} x_j\right), \quad (9)$$

where θ indicates the vector of all weights and biases.

While this is typically not recommended, in this exercise we use 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 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.

First, we want to find the gradient of the loss L_k with respect to the output of the neural network for class k , which is denoted as $f_k(\theta; X)$. The loss function is:

$$L_k = (f_k(\theta; X) - y_k)^2$$

$$\frac{\partial L(f_k(\theta; X), y)}{\partial w_{ij}} = \frac{\partial L}{\partial f_k} \cdot \frac{\partial f_k}{\partial a_k^{[3]}} \cdot \frac{\partial a_k^{[3]}}{\partial h_m^{[2]}} \cdot \frac{\partial h_m^{[2]}}{\partial a_m^{[2]}} \cdot \frac{\partial a_m^{[2]}}{\partial h_i^{[1]}} \cdot \frac{\partial h_i^{[1]}}{\partial a_i^{[1]}} \cdot \frac{\partial a_i^{[1]}}{\partial w_{ij}^{[1]}}$$

1. Next, we calculate the gradient of the loss with respect to $f_k(\theta; X)$, we take the derivative of the loss function with respect to $f_k(\theta; X)$:

$$\frac{\partial L}{\partial f} = \frac{\partial}{\partial f}(y - f(x))^2 = -2(y - f)$$

2. Derivative of the softmax

$$\frac{\partial f_k}{\partial a_k^{[3]}} = o(a_k^{[3]})_i \left(\delta_{ij} - o(a_k^{[3]})_j \right)$$

3. Derivative of output activation

$$\frac{\partial a_k^{[3]}}{\partial h_m^{[2]}} = \frac{\partial}{\partial h_m^{[2]}} \left[b_k^{[3]} + \sum_i^H w_{k,m}^{[3]} h_m^{[2]} \right] = w_{k,m}^{[3]}$$

4. Derivative of activation function (Relu)

$$\frac{\partial h_m^{[2]}}{\partial a_m^{[2]}} = \frac{\partial \sigma(a_m^{[2]})}{\partial a_m^{[2]}} = \begin{cases} 1, & \text{if } a_m^{[2]} > 0 \\ 0, & \text{if } a_m^{[2]} < 0 \end{cases}$$

5. Derivative of hidden layer

$$\frac{\partial a_m^{[2]}}{\partial h_i^{[1]}} = \frac{\partial}{\partial h_i^{[1]}} \left[b_m^{[2]} + \sum_i^H w_{m,i}^{[2]} h_i^{[1]} \right] = w_{m,i}^{[2]}$$

6. Derivative of activation function

$$\frac{\partial h_i^{[1]}}{\partial a_i^{[1]}} = \frac{\partial \sigma(a_i^{[1]})}{\partial a_i^{[1]}} = \begin{cases} 1, & \text{if } a_i^{[1]} > 0 \\ 0, & \text{if } a_i^{[1]} < 0 \end{cases}$$

7. Derivative for hidden layer activation

$$\frac{\partial a_i^{[1]}}{\partial w_{ij}^{[1]}} = \frac{\partial}{\partial w_{ij}^{[1]}} \left[b_i^{[1]} + \sum_m^d w_{im}^{[1]} x_m \right] = x_j$$

Now, let's put it all together to calculate $\frac{\partial L_k}{\partial w_{ij}^{[1]}}$, the gradient of the loss with respect to $w_{ij}^{[1]}$

Substituting the values we calculated:

$$\frac{\partial L_k}{\partial w_{i,j}^{[1]}} = -2(y-f)(a_k^{[3]})_i \left(\delta_{ij} - o(a_k^{[3]})_j \right) w_{k,m}^{[3]} \begin{cases} 1, & \text{if } a_m^{[2]} > 0 \\ 0, & \text{if } a_m^{[2]} < 0 \end{cases} w_{m,i}^{[2]} \begin{cases} 1, & \text{if } a_i^{[1]} > 0 \\ 0, & \text{if } a_i^{[1]} < 0 \end{cases} x_j$$

This equation above is the expression for the gradient of the loss with respect to $w_{ij}^{[1]}$.

And finally, we'll use the gradient $\frac{\partial L_k}{\partial w_{ij}^{[1]}}$ calculated above to determine how to update the weight.

$$w_{ij}^{(r+1)} = w_{ij}^{(r)} + \eta \Delta w_{ij}^{(r)} = w_{ij}^{(r)} - \eta \left(\frac{\partial L(f_k(\theta; X), y)}{\partial w_{ij}} \right)^{(r)}$$

Substituting the value of $\frac{\partial L_k}{\partial w_{ij}^{[1]}}$ from above:

$$w_{ij}^{(r+1)} = w_{ij}^{(r)} + \eta \Delta w_{ij}^{(r)} = w_{ij}^{(r)} - \eta \left(-2(y-f)(a_k^{[3]})_i \left(\delta_{ij} - o(a_k^{[3]})_j \right) w_{k,m}^{[3]} \begin{cases} 1, & \text{if } a_m^{[2]} > 0 \\ 0, & \text{if } a_m^{[2]} < 0 \end{cases} w_{m,i}^{[2]} \begin{cases} 1, & \text{if } a_i^{[1]} > 0 \\ 0, & \text{if } a_i^{[1]} < 0 \end{cases} x_j \right)^{(r)}$$

This equation represents the updated value of the weight $w_{ij}^{[1]}$ after one step of gradient descent. It accounts for the gradient of the loss, the learning rate η , and ReLU activation function.

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.


../img/MNIST-few-examples.png

Figure 1: Example images from MNIST dataset with 28x28 pixel resolution.

The training set with n data points is of the form

$$(x^{(i)}, y^{(i)}), \quad i = 1, \dots, n, \quad (10)$$

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.

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.