

# Problem set 1

## Deep Learning E1394

Alvaro Guijarro (226883)  
Jiayu Yang (213444)

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

Submit your written answers as a pdf typed in  $\text{\LaTeX}$  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  $f$ .

**Solution:**

**Step 1: Find all critical points of  $f(x, y)$**

The partial derivatives of  $f$  is:

$$\nabla f(x, y) = (2xy + y^2 - 6y, x^2 + 2xy - 6x)$$

Thus any critical points of  $f$  must satisfy:

$$2xy + y^2 - 6y = 0$$

and

$$x^2 + 2xy - 6x = 0$$

From the first equation, factor out  $y$ :

$$y(2x + y - 6) = 0$$

$$y = 0 \text{ or } y = 6 - 2x$$

Substitute  $y = 0$  into the second equation:

$$x^2 - 6x = 0$$

$$x(x - 6) = 0$$

$$x = 0 \text{ or } x = 6$$

So, when  $y = 0$ , the critical points are  $(0, 0)$  and  $(6, 0)$ .

Substitute  $y = 6 - 2x$  into the second equation:

$$x^2 + 2x(6 - 2x) - 6x = 0$$

$$2x(2x - 3) = 0$$

$$x = 0 \text{ or } x = \frac{3}{2}$$

So, when  $y = 6 - 2x$ , the critical points are  $(0, 6)$  and  $(\frac{3}{2}, 3)$ .

Hence, there are four critical points in total:

$$(0, 0), (6, 0), (0, 6), \left(\frac{3}{2}, 3\right)$$

***Step 2: Classify the above critical points***

The Hessian of  $f$  is given by

$$H_f(x, y) = \begin{bmatrix} 2y, & 2x + 2y - 6 \\ 2x + 2y - 6, & 2x \end{bmatrix}$$

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

For  $(6, 0)$ , we have:  $H_f(6, 0) = \begin{bmatrix} 0, & 6 \\ 6, & 12 \end{bmatrix} = -36$ . Since  $\det H_f(6, 0) = -36 < 0$ , indicating that  $(6, 0)$  is also a saddle point.

For  $(0, 6)$ , we have:  $H_f(0, 6) = \begin{bmatrix} 12, & 6 \\ 6, & 0 \end{bmatrix} = -36$ . Since  $\det H_f(0, 6) = -36 < 0$ , indicating that  $(0, 6)$  is also a saddle point.

For  $(\frac{3}{2}, 3)$ , we have:  $H_f(\frac{3}{2}, 3) = \begin{bmatrix} 6, & 3 \\ 3, & 3 \end{bmatrix} = 9$ . Since  $\det H_f(\frac{3}{2}, 3) = 9 > 0$ , and  $\frac{\partial^2 f(\frac{3}{2}, 3)}{\partial x^2} = 6 > 0$ , so  $(\frac{3}{2}, 3)$  is a local minimum of  $f$ .

## 1.2 Activation functions

Compute the gradient of the function

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

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

**Solution:**

For the derivative with respect to  $b$ :

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

For the derivative with respect to  $w$ :

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

## 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.

**Solution:**

The x-axis contains the Training Set Size while the y-axis has the training error rate. The way this graph has to be interpreted is that the training error rate (bottom line) decreases as the training set size increases. For the test error rate (top line), the more data you have for training, the more the error decreases, but until a certain point. Afterwards, it can lead to overfitting.

### 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

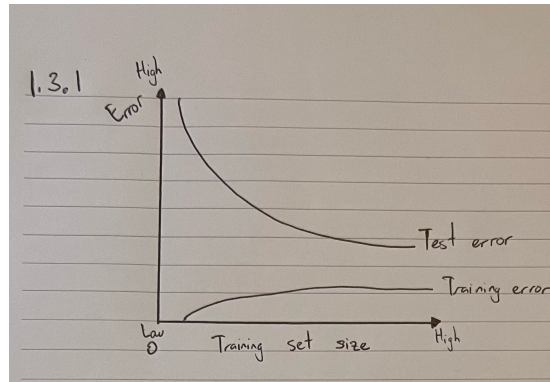


Figure 1: Drawing of Error rate versus dataset size

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.

**Solution:**

The x-axis contains the model complexity while the y-axis contains the error rate. The bottom line represents the training error rate, showing how it typically decreases as model complexity increases. The top line represents the test error rate. It initially decreases with model complexity but may start to rise after a certain point. The vertical line marks the point where the data supports the most complex model without overfitting.

The vertical line indicates the chosen level of model complexity where overfitting is not a significant concern.

### 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.

**Solution:**

The dotted line shows the validation error rate and how it changes as the training epochs increase. The full line is the training error rate, showing how it typically decreases with more training epochs, as the model learns and fits the data better, it reduces the error. Early stopping is a reasonable regularization metric because it prevents the model from continuing to train when it starts to

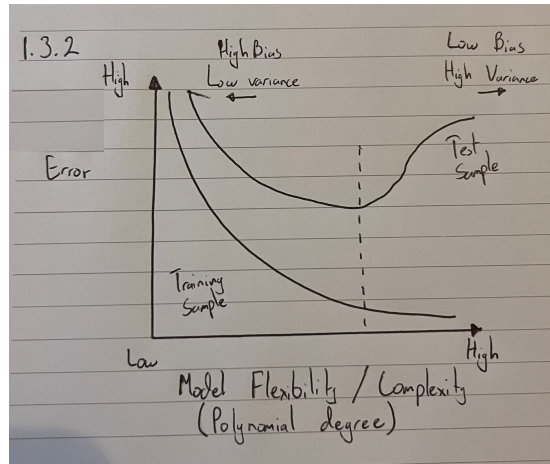


Figure 2: Drawing of Error rate versus model complexity

fit the noise in the data, ensuring better generalization to unseen data.

In this graph, the place where the dotted line starts to rise represents the optimal point for early stopping, in this case it would be epoch 10.

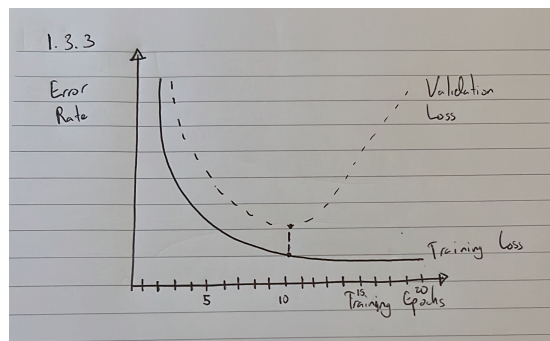


Figure 3: Drawing of Training epochs vs error rate

## 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:  $\beta_0 + \beta_1 x_1 x_2$ .

**Solution:**

In order to design this neural network we can follow these steps:

1. Hidden Layer Activation Function  $g(\cdot)$ :  $g(z)=z$  , with this linear function we have that  $z$  is the weighted sum of inputs.
2. Output Layer Activation Function  $o(\cdot)$ : Since we want the final output to be in the form of logistic regression, which involves the logistic (sigmoid) function, we can set the output activation function to be the logistic sigmoid function (this function maps the weighted sum of inputs to values between 0 and 1, which is suitable for logistic regression:  $o(z) = \frac{1}{1+e^{-z}}$
3. Weight Values: To be able to obtain the desired output of  $\beta_0 + \beta_1 x_1 x_2$  we need to set the right weight values. For the first hidden unit, we will set weights in such way that it computes:  $z_1 = \beta_0 + \beta_1 x_1 x_2$ . The weights could be  $\beta_0 = 1$  for the bias term and  $\beta_1 = 2$  for the weight connecting  $x_1$  and  $x_2$  to the first hidden unit. For the second hidden unit we can set the weights to 0.

With this Hidden Layer Activation Function, Output Layer Activation Function, and Weighted values, let's check how the neural network computes the output:

1. Hidden Layer output:  $z_1 = \beta_0 + \beta_1 x_1 x_2 = 1 + 2x_1 x_2$
2. Output Layer output:  $o(z_1) = \frac{1}{1+e^{-z_1}} = \frac{1}{1+e^{-(1+2x_1 x_2)}}$
3. Now let's consider some input values for  $x_1$  and  $x_2$ :  $x_1 = 1$   $x_2 = 2$ 
  - $z_1 = 1 + 2 * 1 * 2 = 5$
  - $o(z_1) = \frac{1}{1+e^{-5}} = 0.9933$

As you can see, the neural network with the specified weights and activation functions indeed produces outputs that match the logistic regression model's form  $\beta_0 + \beta_1 x_1 x_2$

## 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 has  $H^{[1]}$  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:

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

$$h_m^{[2]} = \sigma(a_m^{[2]}) = \sigma\left(b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]}\right) \quad (3)$$

$$h_i^{[1]} = \sigma(a_i^{[1]}) = \sigma\left(b_i^{[1]} + \sum_{j=1}^d w_{ij}^{[1]} x_j\right), \quad (4)$$

where  $\theta$  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.

*Shows 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.*

**Solution:**

1.  $f_k(\theta; X) = o(a_k^{[3]}) = o(b_k^{[3]} + \sum_{m=1}^{H^{[2]}} (w_{km}^{[3]} h_m^{[2]}))$
2. Gradient Descent Weight Update:  $w^{[r+1]} = w^{[r]} - \alpha \frac{\partial J}{\partial w}$
3. Cost function:  $J(\theta) = \frac{1}{2} \sum_{k=1}^K (y_k - \hat{y}_k)^2$
4. Softmax Activation Function:  $\hat{y}_k = \frac{e^{a_k^{[3]}}}{\sum_{l=1}^K e^{a_l^{[3]}}}$
5. ReLU Activation Function:  $\sigma(z) = \max(0, z)$ ,  $\sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$

We will be working with the following notation to make the calculations easier to follow:

- $f_k(\theta; X) = o(a_k^{[3]}) = o(b_k^{[3]} + \sum_{m=1}^{H^{[2]}} (w_{km}^{[3]} h_m^{[2]})) = o(a^{[3]})$
- $h_m^{[2]} = o(a_m^{[2]}) = \sigma(b_m^{[2]} + \sum_{i=1}^{H^{[1]}} (w_{mi}^{[2]} h_i^{[1]})) = g(a^{[2]})$

Now we are going to apply the weights update for the bias, where:

- $\eta = \text{learning rate}$
- $b_1^{(r+1)} = b_1^{(r)} + \eta \Delta b_1^{(r)} = b_1^{(r)} - \eta \left( \frac{\partial L}{\partial b_1} \right)$

We will now take the following assumptions as well:

- $f(\theta; x_i) = f$
- $h_m^{[2]} = h^{[2]}$

Now we are tasked with computing the gradient  $\frac{\partial L}{\partial b_1}$  with respect to  $b_1$  while expressing L as a function of  $b_1$ . In this context, we define L as the square of the difference between  $y$  and  $f$ .

Considering a model with two hidden layers, we must follow this sequence: Starting from the input  $b$ , it undergoes transformation through the first hidden

layer to create an intermediate representation, denoted as  $a^{[1]}$ . This representation then passes through an activation function,  $h^{[1]}$ , yielding the output  $a^{[2]}$  for the first hidden layer. Subsequently,  $a^{[2]}$  acts as the input for the second hidden layer, generating  $h^{[2]}$ . After undergoing another activation function, we arrive at  $a^{[3]}$ . Finally, the function  $f$  operates on  $a^{[3]}$  to produce the desired output  $L$ .

Then:  $L \leftarrow f \leftarrow a^{[3]} \leftarrow h^{[2]} \leftarrow a^{[2]} \leftarrow h^{[1]} \leftarrow a^{[1]} \leftarrow b_1$

Apply chain rule to get  $\frac{\partial L}{\partial b_1}$  :

- $\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial a^{[3]}} \sum_{m=1}^{H^{[2]}} \frac{\partial a^{[3]}}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial h^{[1]}} \frac{\partial h^{[1]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial b_1}$

Know we need to calculate the terms of the gradient:

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

Activation function:  $\frac{\partial f}{\partial a^{[3]}} = o'(z) = 1$

- $\frac{\partial a^{[3]}}{\partial h^{[2]}} = \frac{\partial}{\partial h^{[2]}} \left[ b^{[3]} + \sum_{m=1}^{H^{[2]}} w_m^{[3]} h_m^{[2]} \right] = w_m^{[3]}$
- $\frac{\partial h^{[2]}}{\partial a^{[2]}} = g'(z) = 1$  for all  $z > 0$ .  $g(z) = \text{ReLU Function}$
- $\frac{\partial a^{[2]}}{\partial h^{[1]}} = \frac{\partial}{\partial h^{[1]}} \left[ b^{[2]} + \sum_{n=1}^{H^{[1]}} w_{ni}^{[2]} h_i^{[1]} \right] = w_m^{[2]}$
- $\frac{\partial h^{[1]}}{\partial a^{[1]}} = g'(z) = 1$  for all  $z > 0$ .  $g(z) = \text{ReLU Function}$
- $\frac{\partial a^{[1]}}{\partial b_1} = \frac{\partial}{\partial b_1} \left[ b_i^{[1]} + \sum_{j=1}^d w_{ij}^{[1]} x_j \right] = 1$

Now we plug all of the partial derivatives in the gradient chain:

- $\frac{\partial L}{\partial b_1} = -2(y - f) \sum_{m=0}^{H^{[2]}} w_m^{[3]} w_m^{[2]}$

Now we update the weights for the bias in the first layer:

- $b_1^{(r+1)} = b_1^{(r)} - \eta \left( 2(y - f) \sum_{m=1}^{H^{[2]}} w_m^{[3]} w_m^{[2]} \right)$

To finish we now compute the weights  $w_{ij}^{[1]}$  at the  $(r+1)^{th}$  iteration:

- $w_{ij}^{(r+1)} = w_{ij}^{(r)} - \eta \left( -2(y - f) \sum_{m=1}^{H^{[2]}} w_m^{[3]} w_m^{[2]} x_{ij} \right)$

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

The training set with  $n$  data points is of the form

$$(x^{(i)}, y^{(i)}), \quad i = 1, \dots, n, \quad (5)$$

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](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](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.