Hertie School

Hertie School of Governance

Master of Data Science for Public Policy

Deep Learning

Assignment 1

Instructors:

Prof. Lynn Kaack

T.A. Chiara Fusar Bassini

Authors:

Augusto Fonseca - 225984 - a.fonseca@students.hertie-school.org

Jorge Roa - 226454 - j.roa@students.hertie-school.org

Santiago Ruz - 214661 - santiago.sordo@students.hertie-school.org

For coding questions, please check the repo problem-set-1-ps1_a on GitHub (https://github.com/Hertie-School-Deep-Learning-Fall-2023/problem-set-1-ps1_a.git).

Submit your written answers as a pdf typed in LaTeXtogether with your code. Submit one answer per group (as assigned on Moodle) and include the 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.

To identify the critical points of f, we first need to compute the gradient of f:

$$\nabla f(x,y) = (\tfrac{\partial f}{\partial x}, \tfrac{\partial f}{\partial y}) = \begin{pmatrix} 2xy + y^2 - 6y \\ x^2 + 2xy - 6x \end{pmatrix} = \begin{pmatrix} y(2x + y - 6) \\ x(x + 2y - 6) \end{pmatrix}$$

Then we need to solve $\nabla f(x,y) = 0$ to find x and y. Both equations need to be equal to 0, so:

Equation I-) $y(2x + y - 6) = 0$;

Equation II-) $x(x + 2y - 6) = 0$.

There are four solutions for these equations:

-1st solution: if x = 0 and y = 0, the $\nabla f(0,0) = \begin{pmatrix} 0 * (2 * 0 + 0 - 6) \\ 0 * (0 + 2 * 0 - 6) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ (trivial solution).

-2nd solution: if x = 0 and y ≠ 0. This means that $y(0 + y - 6) = 0$ and thus y = 6. So (0,6) is also a critical point.

-3rd solution: if x ≠ 0 and y = 0. This means that $x(x + 2 * 0 - 6) = 0$ and thus x = 6. So (6,0) is also a critical point.

- 4th solution: if x ≠ 0 and y ≠ 0. This means that we need to find a solution for equations I and II.

Equation I-) $(2x + y - 6) = 0$;

Equation II-) $(x + 2y - 6) = 0$.

If we multiply the equation II by "-2" and sum the equations, we will find that y = 2 and then x = 2. So (2,2) is also a critical point.

Thus, the critical points of f are (0,0), (0,6), (6,0), (2,2).

To classify the critical points, we need to compute the Hessian of f (H (f)) on each critical point.

$$Hf(x,y) = \begin{bmatrix} \frac{\partial^2 f(x,y)}{\partial x^2} & \frac{\partial^2 f(x,y)}{\partial x \partial y} \\ \frac{\partial^2 f(x,y)}{\partial y \partial x} & \frac{\partial^2 f(x,y)}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2y & 2x+2y-6 \\ 2x+2y-6 & 2x \end{bmatrix}$$

Now we need to compute $\frac{\partial^2 f(x,y)}{\partial x^2}$ and the determinant of the Hessian matrix ($D(x,y)$) in the critical point.

- if $D(x,y) > 0$ and $\frac{\partial^2 f(x,y)}{\partial x^2} > 0$ then the critical point is a local minimum of f.
- if $D(x,y) > 0$ and $\frac{\partial^2 f(x,y)}{\partial x^2} < 0$ then the critical point is a local maximum of f.
- if $D(x,y) < 0$ then the critical point is a saddle point of f.
- if $D(x,y) = 0$ then the critical point could be a minimum, a maximum or a saddle point of f.

- For the 1st critical point, $Hf(0,0) = \begin{bmatrix} 0 & -6 \\ -6 & 0 \end{bmatrix}$

The det $Hf(0,0) = \begin{bmatrix} 0 & -6 \\ -6 & 0 \end{bmatrix} = 0 - 36 = -36$. Thus (0, 0) is a saddle point of f.

- For the 2nd critical point, $Hf(0,6) = \begin{bmatrix} 12 & 6 \\ 6 & 0 \end{bmatrix}$

The det $Hf(0,6) = \begin{bmatrix} 12 & 6 \\ 6 & 0 \end{bmatrix} = 0 - 36 = -36 - >$. Thus, (0, 6) is a saddle point of f.

- For the 3rd critical point, $Hf(6,0) = \begin{bmatrix} 0 & 6 \\ 6 & 12 \end{bmatrix}$

The det $Hf(6,0) = \begin{bmatrix} 0 & 6 \\ 6 & 12 \end{bmatrix} = 0 - 36 = -36 - >$. Thus, (6, 0) is a saddle point of f.

- For the 4th critical point, $Hf(2,2) = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$

The det $Hf(2,2) = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix} = 16 - 4 = 12$

The $\frac{\partial^2 f(x,y)}{\partial x^2} = 4$. Thus, since both solutions are positive, (2, 2) is a local minimum of f.

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 $x, b, w \in \mathbb{R}$.

Answer:

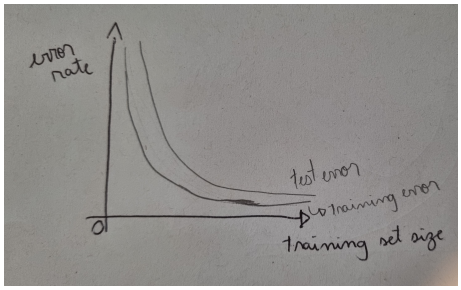
$\nabla f(b,w) = (\frac{\partial f}{\partial b}, \frac{\partial f}{\partial w}) = \begin{pmatrix} \frac{\partial f(b,w)}{\partial b} \\ \frac{\partial f(b,w)}{\partial w} \end{pmatrix} = \begin{pmatrix} 1 \\ x \end{pmatrix}$ if $b+xw \geq 0$ and 0, if $b+xw < 0$

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.*

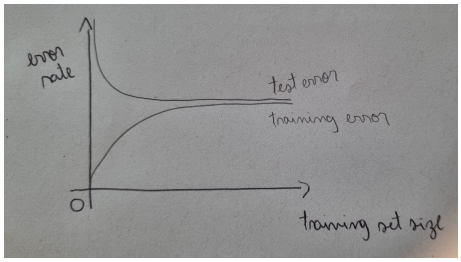
1.3.1 Error rate versus dataset size

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.



Answer: In general, considering an initial training set with multiple rows, since our model is not trained yet, the smaller the training set, the higher the training error and also the test error. As soon as our training set becomes bigger, the training error is sharply reduced and then stabilizes almost flatly. At this moment, the test error is also reduced because the model has learned the pattern and now can predict with a valida-

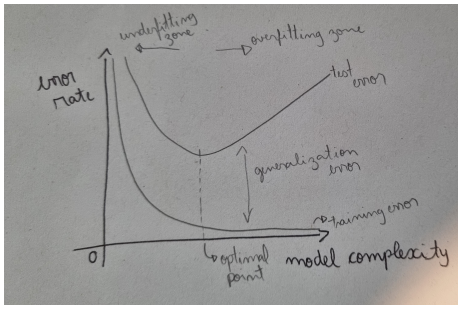
tion set more precisely, or else, with lower test error. Even in this case, the test error will be higher than the training error. Nevertheless, it could be the case where the initial training set size is one. In this case, the



model would not have any difficulty fitting on it, making the training error equal to zero. However, in this scenario, this model would not be able to predict in a validation set with multiple instances, and the test error would be enormous. Since we increase the training set size, the training error would increase sharply and stabilize, while the test error would reduce rapidly and stabilize close to the training 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.

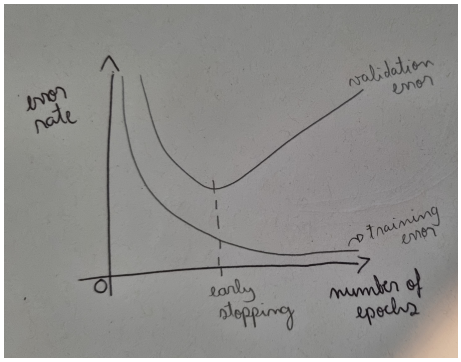


Answer: The generalization error can be understood as the capacity of your model to predict outputs correctly for unseen data, and it can be measured as the difference between the test and the training error. If we choose a too-simple model, the training error will be higher, and it will indicate that our model is not able to fit our training data (underfitting). Also, if we opt for a super complex model, maybe the training error will be lower, but the test error and the generalization will be high (overfitting zone in the graph). The

optimal point would be the point where the test error is lower.
Source: Zhang, Lipton, Li, and Smola, Dive into Deep Learning. Chapter 3.6.

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.



Answer: Early stopping is a form of regularization that stops the training process of a neural network, constraining the number of epochs of training. The idea is to stop the process when the test error, which was decreasing, starts to increase to prevent your system from learning too much from your training set and, consequently, losing the capacity to generalize to new data. The graph above represents this moment. If we didn't use the early stopping technique, the training error would go towards zero, but the test error would increase significantly. In this context, the early stopping is a reasonable regularization metric.
Source: Zhang, Lipton, Li, and Smola, Dive into Deep Learning. Chapter 5.5.

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: $\beta_0 + \beta_1 x_1 x_2$.

We can express neural-network with two hidden units and two features:

$$f(X) = \alpha(b^{[2]} + \sum_{i}^{H=2} \cdot w_i^{[2]} \cdot g(b_i^{[1]} + \sum_{j}^{d=2} \cdot w_{ij}^{[1]} x_j))$$

We know that:

Parameters	Value
$g(z)$	z^2
$b_1^{[1]}$	0
$w_{11}^{[1]}$	1
$w_{12}^{[1]}$	1
$b_2^{[1]}$	0
$w_{21}^{[1]}$	1
$w_{22}^{[1]}$	-1
$b^{[2]}$	β_0
$w_1^{[2]}$	$\frac{1}{4}\beta_1$
$w_2^{[2]}$	$-\frac{1}{4}\beta_1$
$\alpha(z)$	$\frac{1}{1+e^{-z}}$

Table 1: Given values and functions

$$f(x) = \alpha(\beta_0 + \frac{1}{4}\beta_1(x_1 + x_2)^2 - \frac{1}{4}\beta_1(x_1 - x_2)^2)$$

$$f(x) = \alpha(\beta_0 + \frac{1}{4}\beta_1(x_1^2 + 2x_1x_2 + x_2^2) - \frac{1}{4}\beta_1(x_1^2 - 2x_1x_2 + x_2^2))$$

$$\alpha(\beta_0 + \beta_1 x_1 x_2)$$

$$\frac{1}{1 + exp - (\beta_0 + \beta_1 x_1 x_2)}$$

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 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:

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

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

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

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.

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

1.- What we know about the Gradient Descent Weight Update:

$$w^{[r+1]} = w^{[r]} - \alpha \frac{\partial J}{\partial w}$$

2.- Cost Function

$$J(\theta) = \frac{1}{2} \sum_{k=1}^{K} (y_k - \hat{y_k})^2$$

3.- Softmax Activation Function

$$\hat{y_k} = \frac{e^{a_k^{[3]}}}{\sum_{l=1}^{K} e^{a_l^{[3]}}}$$

4.- ReLU activation function

$$\sigma(z) = max(0, z)$$

$$\sigma'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

For practice purposes, we will have this notation:

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

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

1.5.1 Method

Applying the weights update for the bias:

η is the learning rate

$$b_1^{(r+1)} = b_1^{(r)} + \eta \Delta b_1^{(r)} = b_1^{(r)} - \eta(\frac{\partial L}{\partial b_1})$$

Where:

$$f(\theta; x_i) = f$$

$$h_m^{[2]} = h^{[2]}$$

We need to calculate the gradient $\frac{\partial L}{\partial b_1}$ writing L in function of b_1

Here, $L = (y - f)^2$

Considering two hidden layers, this is what we should consider in the path:

Starting from the input b_1, it is transformed through the first hidden layer to generate an intermediate representation $a^{[1]}$. This representation is then passed through an activation function $h^{[1]}$, which results in the output $a^{[2]}$ for the first hidden layer. Subsequently, $a^{[2]}$ serves as the input to the second hidden layer, producing $h^{[2]}$. After passing through another activation function, we obtain $a^{[3]}$. Finally, the function f processes $a^{[3]}$ to produce the desired output L.

Hence:

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

Now, we need to apply the chain rule knowing our different paths to get $\frac{\partial L}{\partial b_1}$

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

Terms of the gradient:

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

$\frac{\partial f}{\partial a^{[3]}} = o'(z) = 1$ is our linear activation function

$$\frac{\partial a^{[3]}}{\partial h^{[2]}} = \frac{\partial}{\partial h^{[2]}}[b^{[3]} + \sum_{m=1}^{H^{[2]}} w_m^{[3]} h_m^{[2]}] = w_m^{[3]}$$

$\frac{\partial h^{[2]}}{\partial a^{[2]}} = g'(z) = 1$ for z > 0 as $g(z) = ReLu$

$$\frac{\partial a^{[2]}}{\partial h^{[1]}} = \frac{\partial}{\partial h^{[1]}}[b^{[2]} + \sum_{m=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]}] = w_{mi}^{[2]}$$

$\frac{\partial h^{[1]}}{\partial a^{[1]}} = g'(z) = 1$ for z > 0 as $g(z) = ReLu$

$$\frac{\partial a^{[1]}}{\partial b_1} = \frac{\partial}{\partial b_1}[b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]} x_j] = 1$$

Plugging the partial derivatives in the gradient chain:

$$\frac{\partial L}{\partial b_1} = -2(y - f) \sum_{m=1}^{H^{[2]}} w_m^{[3]} w_{mi}^{[2]}$$

Now, the weight update for the bias in layer 1:

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

Considering now the computation of the weights $w_{ij}^{[1]}$ update at the $(r+1)^{th}$ iteration:

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

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.*

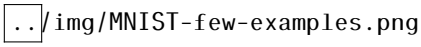


Figure 1: Example images from MNIST dataset with 28x28 pixel resolution.

The training set with n data points is of the form

$$(x^{(i)}, y^{(i)}),\ i = 1, ..., n, \tag{5}$$

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`.

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.