Assignment 1: *Fundamentals of Machine Learning*

Date: *Jan 16, 2025* Due : *Jan 30, 2025*

Preface

This is the first series of assignments for the course *ECE1508: Applied Deep Learning*. The exercises are aimed to review the preliminaries studied in Chapter 1. Please adhere to the **Code of Honor** outlined on the Quercus page for this course. Below, you can find the information about contents of these exercises, as well as instructions on how to submit them. Please read them carefully.

General Information The assignments are given in two sections. In the first section, you have written questions that you can answer in words or by derivation. The questions are consistent with the material of Chapter 1 and you do not need any further resources to answer them. The second section includes the programming assignments. In the case that a question is unclear or there is any flaws, please contact over Piazza. Also, in case that any particular assumption is required to solve the problem, feel free to consider the assumption and state it in your solution. The total mark of the assignments is **100 points** with total mark of written questions adds to **40 points** plus **4 extra points** and the total mark of the programming assignments adds to **60 points**. Please note that the extra points are given to the optional items and can compensate other deductions in this assignment only.

How to Submit A notebook has been provided with basic code that you can complete. The submission is carried out through the Crowdmark. Please submit the answer of each question **separately**, following the **steps below**. Please note that *failure to meet the formatting can lead to mark deduction*.

1. For Written Questions, you can submit handwritten or typed answers as a `.pdf`, `.jpg` or `.png` file.
2. For Programming Questions, the answer should **complete the Python Notebook** shared with this assignment. For *each question*, please print the corresponding part of the notebook as a `.pdf` file and upload it in the corresponding field on Crowdmark. Your uploaded `.pdf` should contain **all figures, diagrams and codes requested** in the question.
3. The completed Notebook, i.e., the `.ipynb` file, including all the codes and the outputs should also be submitted as the attachment to the last item on Crowdmark.

When submitting your notebook, please pay attention to the following points:

1. The file should be named `Lastname_Firstname_Assgn1.ipynb`
2. Please make sure to name the files with the name that is displayed on the Quercus account

The deadline for your submission is on **January 30, 2025 at 11:59 PM**.

- You can delay up to two days, i.e., until **February 1, 2025 at 11:59 PM**. After this extended deadline no submission is accepted.
- For each day of delay, 10% of the mark after grading is deducted.

Please submit your assignment **only through Crowdmark, and not by email.**

1 Written Exercises

Question 1 [15 + 4 (Optional) Points] **(Linear Regression)** Recall the dummy example of the projectile:

> *a projectile located at height h is initiated by horizontal velocity v. We aim to find the distance at which the projectile hits the ground*

Using Newton's laws, we found the distance in Chapter 1 analytically as $d = 0.45v\sqrt{h}$. We now consider the machine learning approach and use a linear model to approximate the distance. The model is

$$y = w_0 v + w_1 h \tag{1.1}$$

as an estimate of the true distance d. For sake of compactness, let us define

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}. \tag{1.2}$$

Assume that we make a dataset with I samples as

$$\mathbb{D} = \left\{ \left(\mathbf{x}_i = \begin{bmatrix} v_i \\ h_i \end{bmatrix}, d_i \right) : i = 1, \dots, I \right\}, \tag{1.3}$$

and use squared error loss function to compute the loss between true label d and its approximation y as

$$\mathcal{L}(y, d) = (y - d)^2 \tag{1.4}$$

to train this model. Let matrix $\mathbf{X} \in \mathbb{R}^{I \times 2}$ be the collection of all data-points, i.e.,

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_I^\top \end{bmatrix} = \begin{bmatrix} v_1 & h_1 \\ v_2 & h_2 \\ \vdots & \vdots \\ v_I & h_I \end{bmatrix}, \tag{1.5}$$

and $\mathbf{d} \in \mathbb{R}^I$ be the vector of corresponding labels, i.e.,

$$\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_I \end{bmatrix}. \tag{1.6}$$

This is an example of *linear regression*; one of the few settings, where we can solve the training problem analytically. In this exercise, we try to find the optimal weights by answering the following items:

1. Starting from the definition, use basic linear algebra to show that *empirical risk* is given by

$$\hat{R}(\mathbf{w}) = \frac{1}{I} \|\mathbf{X}\mathbf{w} - \mathbf{d}\|^2. \tag{1.7}$$

2. Determine the gradient of the empirical risk in a *compact form*. Specify the update rule of gradient descent algorithm with learning rate η.

3. Find the minimizer of the *empirical risk* by setting its gradient to zero.
   Hint: *You can write it as a solution to a linear system of equation.*

4. Assume we have only $I = 2$ data-points, and that $\mathbf{X} \in \mathbb{R}^{2\times 2}$ in this case is invertable. Show that in this case the minimum empirical risk at is zero. Explain how the minimum empirical risk changes when $I \geq 3$.

5*. (Optional) Explain why we cannot necessarily find a learning rate, such that the gradient descent algorithm converges in *only one step*.

6*. (Optional) Replace the update rule of gradient decent with

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \begin{bmatrix} \eta_1 & 0 \\ 0 & \eta_2 \end{bmatrix} \nabla \hat{R}(\mathbf{w}^{(t)}) \tag{1.8}$$

for some positive reals η_1 and η_2. Show that in this case, we can find η_1 and η_2, such that this modified version of gradient descent converges in *one step*.

Question 2 [10 Points] **(Learning XOR via Artificial Neurons)** Recall the problem of learning XOR function: *our dataset consists of 4 points*

$$\mathbb{D} = \left\{ (\begin{bmatrix} 0 \\ 0 \end{bmatrix}, 0), (\begin{bmatrix} 0 \\ 1 \end{bmatrix}, 1), (\begin{bmatrix} 1 \\ 0 \end{bmatrix}, 1), (\begin{bmatrix} 1 \\ 1 \end{bmatrix}, 0) \right\} \tag{1.9}$$

and we learned this function by a network of three perceptrons of the following form

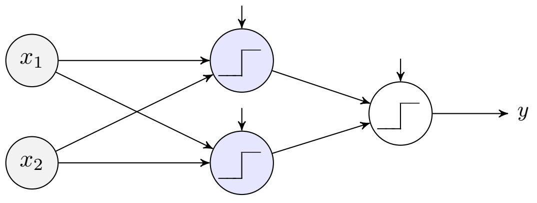


Figure 1.1: Neural network that realizes XOR of x_1 and x_2.

where x_1 and x_2 are the entries of the data-point and y is the corresponding label, i.e., $y = x_1 \oplus x_2$.

1. Replace the perceptrons in the hidden layer of **Figure 1.1**, i.e., those colored, with artificial neurons whose activation function is linear, i.e., $f(z) = z$. Show that this neural network is equivalent to a single perceptron. That means the hidden layer does nothing in this case.
2. Replace the perceptrons in the hidden layer of **Figure 1.1** once again; this time with artificial neurons whose activation function is ReLU, i.e., $f(z) = \max\{z, 0\}$. Find the weights and biases of these neurons, such this network also learn the XOR function.
Hint: *You can find the weights by simple geometrical investigations and do not need to define loss and solve optimization.*

Good to Know
What you have seen in Part 1 of Question 2 explains a general concept: *to learn a general nonlinear function, we need a hidden layers to have **nonlinear** activation.*

Question 3 [15 Points] **(Why Deep)** Consider the following N-dimensional binary function

$$f(x_1, x_2, \ldots, x_N) = x_1 \oplus x_2 \oplus \ldots \oplus x_N. \tag{1.10}$$

This function can be represented by a network of perceptrons with *only one hidden layer*: if we use only one hidden layer; then, we would need to have

$$\text{\# of perceptrons in two-layer network} = 2^{N-1} \tag{1.11}$$

perceptrons in its hidden layer. To check the validity of this argument, recall the case with $N = 2$ in **Figure 1.1**. There we have $2^{N-1} = 2^1 = 2$ perceptrons in the hidden layer.

We now want to learn the same function with a deeper network. Let us for sake of simplicity assume that $N = 2^K$ for some integer K, i.e., $\log_2 N = K$. To this end, answer the following items.

1. First assume that $N = 4$, i.e.,

   $$f(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4. \tag{1.12}$$

   Use the fact that

   $$x_1 \oplus x_2 \oplus x_3 \oplus x_4 = (x_1 \oplus x_2) \oplus (x_3 \oplus x_4) \tag{1.13}$$

   and design a neural network (whose neurons are all perceptrons) with 3 hidden layers that realizes the function $f(x_1, x_2, x_3, x_4)$. You do **not** need to specify the weights.
   Hint: *You can do this by first realizing $z_1 = x_1 \oplus x_2$ and $z_2 = x_3 \oplus x_4$, and then $z_1 \oplus z_2$.*

2. Repeat the hierarchical computation of Part 1; this time for $N = 8$. How many hidden layers does your final neural network have?

3. By induction, count the number of hidden layers for a general $N = 2^K$.

4. Count the total number of neurons in the neural networks of Parts 1 and 2, and use induction to count this number for general $N = 2^K$.
   Hint: *The following identity might be helpful:*

   $$\sum_{k=0}^{K-1} 2^k = 2^K - 1 \tag{1.14}$$

5. Compare the number of neurons in this *deep* neural network with 2^{N-1} we need in the shallow network with one hidden layer. What do you conclude?

GOOD TO KNOW
While we need *exponentially* large number of neurons in a shallow network, a deep neural network whose depth is proportional to the logarithm of the input dimension, i.e., $\log N$, can do the same job with *linearly* large number of neurons. This explains why *deep* neural network are so powerful and popular.

2 Programming Exercises

QUESTION 1 [30 Points] **(Projectile Example)** We implement the linear model that estimates the landing point of the projectile in our dummy example. To this end, we need to make a dataset, write a model and train it.

1. Start your code by writing a class called `LinearMachine()`. This is given in the attached notebook. You may complete all functions of this class in the notebook.

We know the model and training loop from the written question. We just need to learn how to generate a dataset. Read the following text to learn how to do it.

Generating Synthetic Dataset As we are not really going to do the experiment with a projectile and measure distances, we generate *synthetic data*. This means that we generate random data-points and assign them labels that are theoretically correct. This is in contrast to *real-world data* that would have been collected by really doing the projectile experiment and measure distances. To generate our synthetic dataset, we use `random` module in `NumPy` to generate some random heights and velocities. We then determine the label for each height and velocity using the analytical expression $d = 0.45v\sqrt{d}$.

2. Write a function that takes dataset size I, mean velocity v_0, velocity variance σ_v^2, mean height h_0, height variance σ_h^2 as input and returns a synthetic dataset.
   - Heights and velocities are generated i.i.d. as $v_i = |\tilde{v}_i|$ and $h_i = |\tilde{h}_i|$
     - $\rightarrow$ $\tilde{v}_i$ is a normal random variable with mean v_0 and variance σ_v^2
     - $\rightarrow$ $\tilde{h}_i$ is normal random variables with mean h_0, and variance σ_h^2

   Call this function `data_synthesizer`.

3. Write a function that takes the dataset and learning rate η, computes the empirical risk, and trains the linear model by minimizing the empirical risk using the *gradient descent* algorithm. Call this function `train_GD`.
   Hint: *Note that you have already computed the gradient in Part 2 of Written Question 1.*

4. Write a function that takes a dataset and returns the minimizer of the empirical risk. Call this function `train`.
   Hint: *Note that you have already computed the gradient in Part 3 of Written Question 1.*

5. Use the implemented class to generate a synthetic dataset with
   $\rightarrow$ $I = 100$, $v_0 = 1$, $\sigma_v^2 = 5$, $h_0 = 3$, and $\sigma_h^2 = 3$.

   Write a function (you may implement it outside the class) that gets an array of learning rates and returns the Euclidean distance (norm of the difference) between the outputs of `train` and `train_GD` for each learning rate in the array. Plot the difference against the vector of learning rates.

Now that we have trained the model, we should test it. Read the following text to learn how to do it.

Testing Trained Model For testing, we must generate a *separate* synthetic dataset of size J and inspect the performance of our model by comparing its outputs with the labels of the dataset. This means that we should determine the empirical risk with our new test dataset and evaluate its value for the trained model parameters.

6. Write a function that takes a size for the test dataset and returns the empirical risk of the trained model for the *test dataset*. Call this function `test`.
   Hint: *Note that based on description test and training datasets are independent.*

7. Write a function (you may implement it outside the class) that gets the values
   $\rightarrow$ T, I J, as well as v_0, σ_v^2, h_0, and σ_h^2,

   and returns the average test risk by looping over T independent realizations of training and test datasets.

8. Set the function inputs to
   $\rightarrow$ $T = 100$, $J = 10$, $v_0 = 1$, $\sigma_v^2 = 5$, $h_0 = 3$, and $\sigma_h^2 = 3$.

   Evaluate the average test risk for an array of training dataset sizes, i.e., an array of I, and plot the test risk against I.

Question 2 [30 Points] **(Pattern Recognition)** In this exercise, we train a single perceptron to recognize a pattern in an image. You may take a look at the perceptron algorithm in Chapter 1. We first make a minor modification to the algorithm. For this, read the following text.

Introducing Learning Rate We can introduce a learning rate to the algorithm by modifying its update rule as $\mathbf{w} \leftarrow \mathbf{w} - \eta \text{ sign}(z_i)\, \boldsymbol{x}_i$, where η is the learning rate.

With this modification in mind, answer the following items:

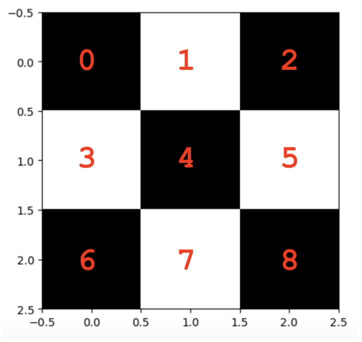
1. Define a `class` called `PerceptronMachine()` for a perceptron with 9 inputs. You may use the code given in the notebook.
   - $\rightarrow$ As an attribute of this class, we should define the weights and bias. Yo may initiate them with some random numbers.
2. Write a function that passes data-point $\boldsymbol{x} \in \mathbb{R}^9$ through the perceptron. Call this function `forward`.
3. Write a function that gets a dataset and trains the perceptron using the perceptron algorithm. Call this function `train`.

We next need to generate a dataset. Read the following text to learn how to do it.

Image as List As mentioned in the lecture, we can represent any image by an array of pixels. For this exercise, we consider a very simple example:

> *a 3×3 image consists of 9 pixels. We consider a simple case, where each pixel has either value 0 (black) or value 256 (white). We can visualize this image using* `Matplotlib`.

A sample code is provided in the notebook to plot the following 9-pixel pattern of X.



We can show this image in a list of size 9, where each entry of this list is the value of the pixel. The index of each entry has been specified in the above image. For instance the above X pattern is specified by the list `x = 256 * [0, 1, 0, 1, 0, 1, 0, 1, 0]`.

4. Write a function that generates the following dataset: the set of all 9-pixel images (each pixel either black or white), where the label of X pattern is 1 and the label of others is 0. You may note the following items
   - $\rightarrow$ We have $2^9 = 512$ binary vectors of dimension 9; thus, we have 512 images in the dataset.
   - $\rightarrow$ Label `x = 256 * [0, 1, 0, 1, 0, 1, 0, 1, 0]` with `y=1` and other lists with `y=0`.
5. Train the implemented perceptron with this new dataset.
6. Validate your training by printing the output of the perceptron to X pattern and 10 other randomly selected images from the dataset.
   Hint: *This problem is linearly separable; thus, the perceptron should exactly learn this classifier.*