INTRODUCTION MACHINE LEARNING

EXERCISE 4

Bauhaus-Universität Weimar

TEACHER:
Johannes Kiesel

GROUP:
Group 16

SUBMITTED BY:

Aaron Perez Herrera
Cesar Fernando Gamba Tiusaba
Chun Ting Lin
Olubunmi Emmanuel Ogunleye

Exercise 1: Gradient Descent (1+1+1=3 Points)

(a) Name one difference between the perceptron training rule and the gradient descent method.

The Perceptron training rule is not based on residuals in the (p+1) dimensional input- output-space) but refers to the input space only, where it simply evaluates the side of the hyperplane as a binary feature (correct side or not). Gradient descent is a regression approach that exploits the residuals provided by a loss function of choice, whose differential is evaluated to guide hyperplane search

(b) What are the main requirements for the hypothesis space and error function for gradient descent to be successfully applied?

In batch gradient descent, the weight vector is determined by calculating the differences in weights for each example in the training data and subsequently computing the weight vector for the entire batch in a single step. Incremental gradient descent involves computing the weight vector immediately after calculating the difference in weights for each example in the training dataset, leading to a more iterative and continuous update process. Compared to batch gradient descent, the example-based weight adaptation of incremental gradient descent can better avoid getting stuck in a local minimum of the loss function.

(c) Name the key difference between the algorithm of batch gradient descent (BGD) and the algorithm of incremental gradient descent (IGD).

The perceptron y1(), with a non-zero bias term wo. xo, is more general than yo(), which has a bias term of zero. The bias term acts as an additional flexibility factor, allowing y1() to represent all the functions yo()'s weights can capture, plus additional functions that arise from the non-zero bias. In essence, y1() can model a broader range of relationships in the data, making it more versatile and general than yo(). The bias term allows models to fit the data better by providing additional flexibility.

Exercise 2 : Perceptron Learning (2 Points)

Consider two perceptrons, yo () and y1 (), both defined by the function heaviside(j_owjj). Both perceptrons have identical weights except for the bias term: wo = 0 for yo() and wo = 1 for y1(). Determine if one of the perceptrons, yo() or y1 (), is more general than the other (as defined in the lecture units on concept learning). If one is more general, specify which one and explain your answer.

We have two perceptrons, $y_0()$ and $y_1()$, which are defined using the Heaviside function (a step function) and both have identical weights except for the bias term. Specifically.

- $W_0 = 0$ for $y_0()$
- $W_0 = 1$ for $y_1()$

And each perceptron is represented as:

$$y_0 = heaviside\left(\sum_{j=0}^{p} w_j\, x_j\right) with\ w_o = 0$$

$$y_1 = heaviside\left(\sum_{j=0}^{p} w_j\, x_j\right) with\ w_o = 1$$

The bias between $y_0()$ and $y_1()$ is 0 and 1, respectively. This bias shifts the decision boundary of the perceptron. The general decision rule for a perceptron is:

$$Output\ = heaviside\left(\sum_{j=0}^{p} w_j\, x_j + w_0\right)$$

For perceptron $y_0()$, the output is determined by the sum of the weighted inputs, and since the bias is 0, the decision boundary is where:

$$\sum_{j=0}^{p} w_j x_j = 0$$

For perceptron $y_1()$, the decision boundary is where:

$$\sum_{j=0}^{p} w_j x_j + 1 = 0\ \ or\ \sum_{j=0}^{p} w_j x_j = -1$$

- The perceptron $y_0()$ has the decision boundary at $\sum_{j=0}^{p} w_j x_j = 0$
- The perceptron $y_1()$ has the decision boundary shifted by -1, i.e., at $\sum_{j=0}^{p} w_j x_j = -1$

Since the bias in $y_0()$ is 0, it has a decision boundary at the origin. On the other hand, the bias in $y_1()$ shifts the decision boundary. This shift in the decision boundary means that $y_1()$ is more general than $y_0()$ because it allows for a broader range of inputs to be classified as positive, due to the shift in the threshold.

In conclusion $y_1()$ is more general than $y_0()$ because the bias in $y_1()$ shifts the decision boundary, allowing for a greater variety of inputs to be classified as 1. Therefore, $y_1()$ can classify a broader range of inputs, making it more general.

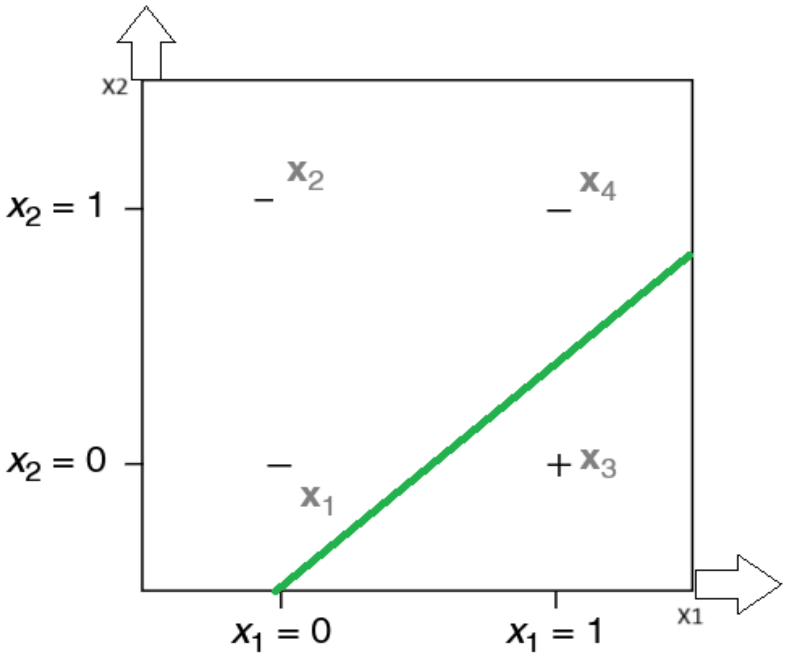
Exercise 3 : Perceptron Learning (1+1+2+2+1=6 Points)

In this exercise, you design a single perceptron with two inputs x1 and x2. This perceptron shall implement the Boolean formula A A -B with a suitable function y(x1, x2). Use the values 0 for false and 1 for true.

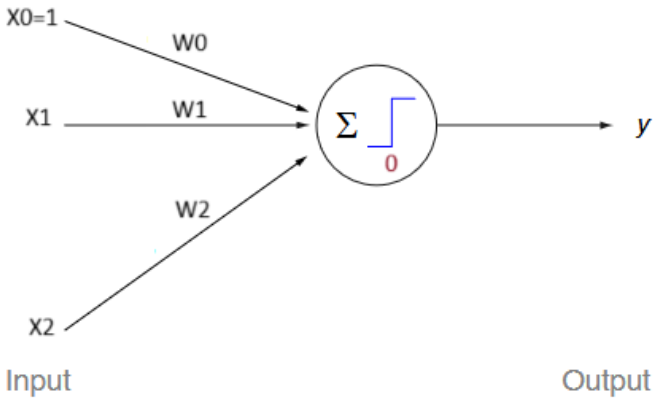
(a) Draw all possible examples and a suitable decision boundary in a coordinate system.

First, we build the table

Example	A	B	¬ B	A ^¬ B	C
X_1	0	0	1	0	-
X_2	0	1	0	0	-
X_3	1	0	1	1	+
X_4	1	1	0	0	-



(b) Draw the graph of the perceptron. The schematic must include a1, x2, and all model weights.



(c) Manually determine the weights w = (wo, w1, w2) for the decision boundary you drew in (a).

A suitable weight values would be by introducing a negative bias term W0

$w_0 = -0.5; \qquad w_1 = 1; \qquad w_2 = -1$

$y = w_0 + x_1 w_1 + x_2 w_2$

Considering we are using Heaviside function.

Example	X1	X2	y	C
X_1	0	0	-0.5 < 0	-
X_2	0	1	-1.5 < 0	-
X_3	1	0	0.5 > 0	+
X_4	1	1	-0.5 < 0	-

The results reflect the behavior of the perceptron.

(d) Now determine w using the perceptron training algorithm (PT). Use a learning rate n of 0.3 and initialize the weights with wo = -0.5 and w1 = w2 = 0.5. Instead of selecting examples randomly, use the following examples in the given order (stop after those four examples):

Start with iteration of following examples and check the updated value after each example

X1	X2	C
0	0	0
0	1	0
1	0	1
1	1	0

$w = [-0.5, 0.5, 0.5] \qquad \rightarrow n = 0.3$

Example	x1	x2	c	y	Δ W0	ΔW1	ΔW2		W0	W1	W2
X1	0	0	0	0	0	0	0		-0.5	0.5	0.5
X2	0	1	0	1	-0.3	0	-0.3		-0.8	0.5	0.2
X3	1	0	1	0	0.3	0.3	0		-0.5	0.8	0.2
X4	1	1	0	1	-0.3	-0.3	-0.3		-0.8	0.5	-0.1

$y = heaviside(w0 + w1 * x1 + w2 * x2)$

$\Delta w = n(c - y) * x \qquad \rightarrow x = [x0, x1, x2]$

$w = w + \Delta w$

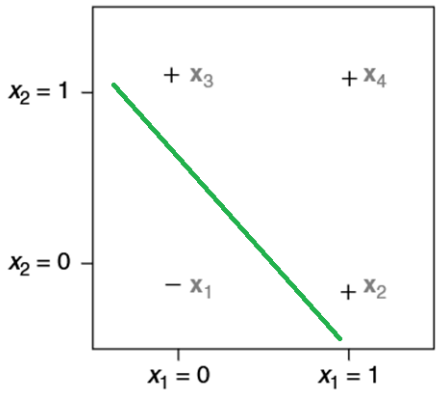
Final weights

W0 = -0.8 W1 = 0.5 W2 = -0.1

Draw the decision boundary after every weight update into the coordinate system of (a).

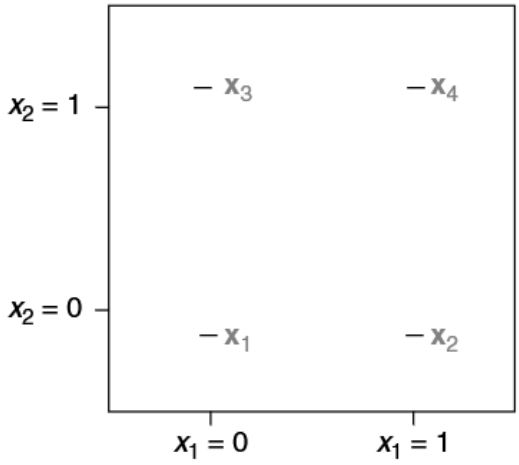
First calculation

Example	x1	x2	c	y
X1	0	0	0	0
X2	0	1	0	1
X3	1	0	1	1
X4	1	1	0	1



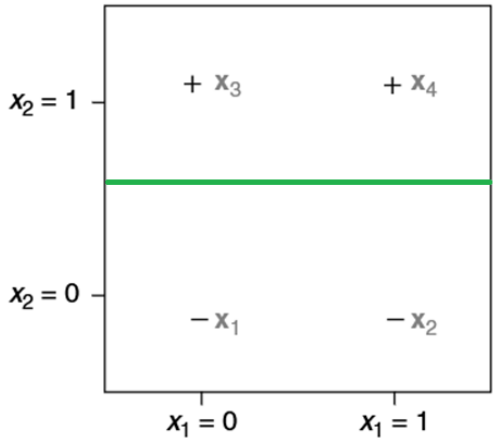
First update

Example	x1	x2	c	y
X1	0	0	0	0
X2	0	1	0	0
X3	1	0	1	0
X4	1	1	0	0



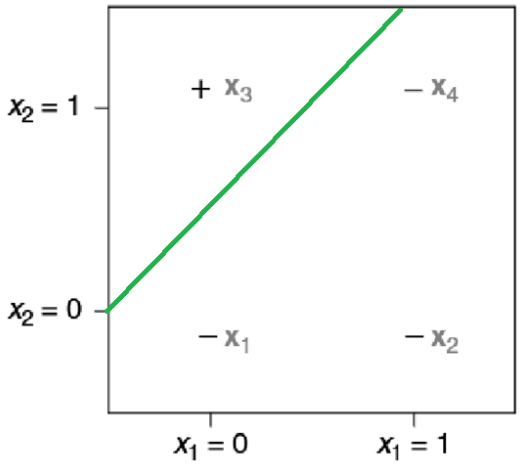
Second update

Example	x1	x2	c	y
X1	0	0	0	0
X2	0	1	0	0
X3	1	0	1	1
X4	1	1	0	1



Third / last update

Example	x1	x2	c	y
X1	0	0	0	0
X2	0	1	0	0
X3	1	0	1	1
X4	1	1	0	0



(e) Briefly describe one effect of changing the learning rate n on the learning progress.

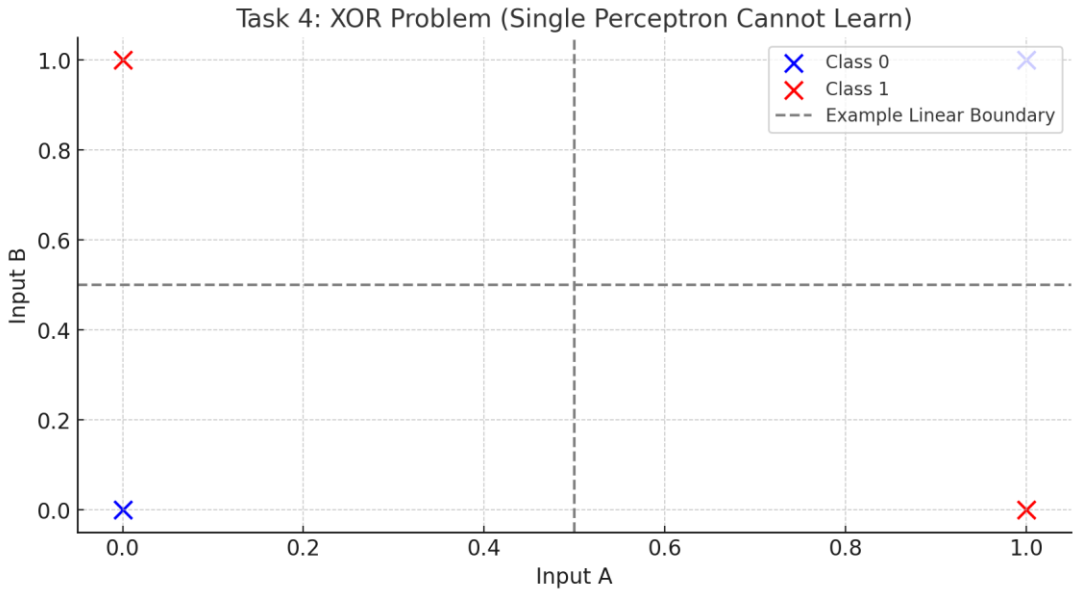
A large learning rate means we are making larger changes to the weights every time we do a learning update. However, if the changes are too large, we might oscillate back and forth rather than focusing on a solution.

- Higher η: Faster convergence but risks overshooting optimal weights.
- Lower η: Slower convergence but smoother and more stable learning.

Exercise 4: Perceptron Learning why a single perceptron cannot learn the Boolean XOR function A XOR B?

XOR function outputs 1 if exactly one of the inputs is 1, otherwise 0.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



Points (0, 0) and (1, 1) represent one class (e.g., blue, 0), while (0, 1) and (1, 0) represent the other class (e.g., red, 1).
Notice that the data is not linearly separable. A single straight line cannot divide the output classes into distinct groups.

No single straight line can separate the two classes (0 and 1) correctly. This is because XOR is not linearly separable.

Exercise 5 : Multilayer Perceptron

Consider the minimum multilayer perceptron that can handle the XOR problem (slide ML:IV-59), but with an activation function f (z) = (f(z1), . . . , f(zd))T instead of Heaviside(). This is a two-layer perceptron with p = 2 attributes, a hidden layer with l = 2 units, and an output layer with k = 1 units.
Therefore, as per slide ML:IV-82:

$$\mathbf{y}(\mathbf{x}) = \mathbf{f}\left(W^{\mathsf{o}}\begin{pmatrix}1\\ \mathbf{f}(W^{\mathsf{h}}\mathbf{x})\end{pmatrix}\right) = \mathbf{f}\left((w^{\mathsf{o}}_{10}, w^{\mathsf{o}}_{11}, w^{\mathsf{o}}_{12})\begin{pmatrix}1\\ \mathbf{f}\left(\begin{pmatrix} w^{\mathsf{h}}_{10} & w^{\mathsf{h}}_{11} & w^{\mathsf{h}}_{12}\\ w^{\mathsf{h}}_{20} & w^{\mathsf{h}}_{21} & w^{\mathsf{h}}_{22}\end{pmatrix}\begin{pmatrix}1\\ x_1\\ x_2\end{pmatrix}\right)\end{pmatrix}\right)$$

Confirm for yourself that thus

$$\mathbf{y}(\mathbf{x}) = f(w^{\mathsf{o}}_{10} + w^{\mathsf{o}}_{11}\cdot f(w^{\mathsf{h}}_{10} + w^{\mathsf{h}}_{11}\cdot x_1 + w^{\mathsf{h}}_{12}\cdot x_2) + w^{\mathsf{o}}_{12}\cdot f(w^{\mathsf{h}}_{20} + w^{\mathsf{h}}_{21}\cdot x_1 + w^{\mathsf{h}}_{22}\cdot x_2))$$

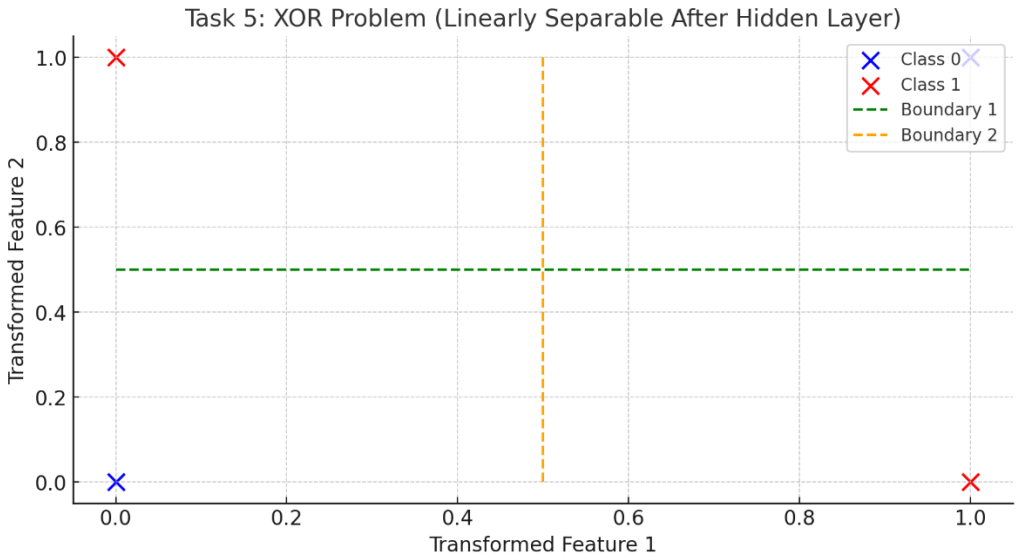
1. Understand the architecture:
   XOR can be solved by a 2-layer perceptron:
   - Input layer: Two neurons for A and B.
   - Hidden layer: Two neurons to map the problem to a linearly separable. representation.
   - Output layer: One neuron to combine results from the hidden layer.

2. Activation function:
   The problem states the activation function is the identity function f(z) = z .

3. Key steps:
   Write the equations for the hidden layer:
   $h\_1 = f(w\_\{10\} + w\_\{11\}A + w\_\{12\}B)$
   $h\_2 = f(w\_\{20\} + w\_\{21\}A + w\_\{22\}B)$

   Write the equation for the output layer:
   $y = f(w\_\{o0\} + w\_\{o1\}h\_1 + w\_\{o2\}h\_2)$

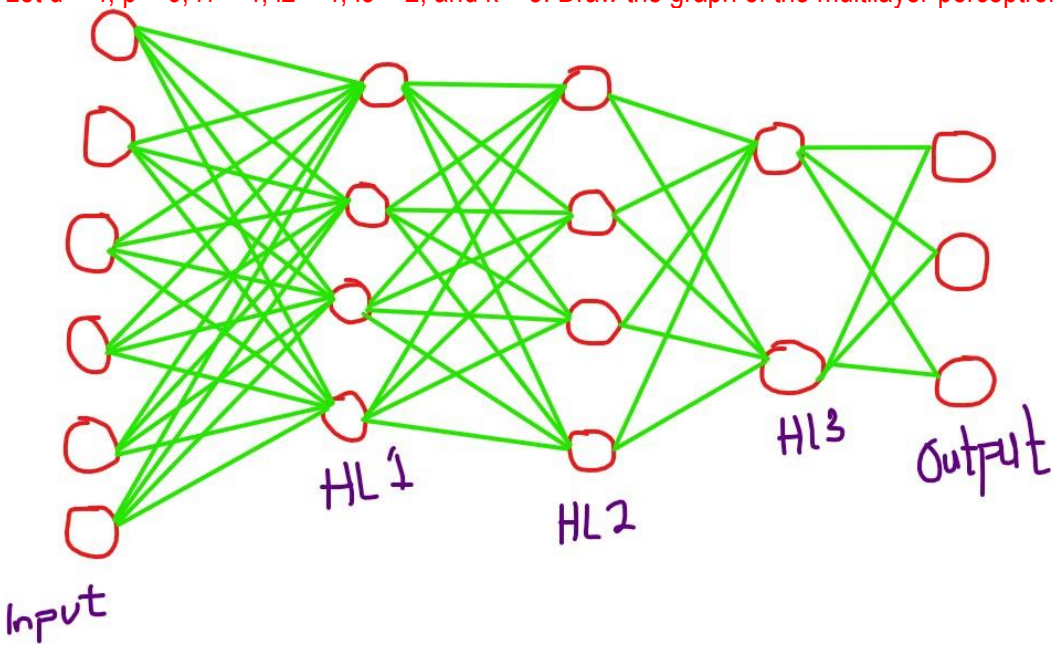
4. Proof for failure with f(z) = z :
   - Show that with the identity activation, the perceptron behaves linearly.
   - XOR still requires non-linear boundaries, so the network fails.



Graph Transformation:
The hidden layer transforms the input space so that the points become separable in a new space. The final output layer combines these to classify XOR correctly.

Exercise 6 : Parameters of the Multilayer Perceptrons (2+1+0+0=3 Points)
In this exercise, you analyze the number of weights (parameters) of multilayer perceptrons. We use the notation from the lecture (e.g., slide ML:IV-104), where multilayer perceptrons have d layers, p attributes, hidden layer i with li units, and an output layer with k units.

a. Let d = 4, p = 6, l1 = 4, l2 = 4, l3 = 2, and k = 3. Draw the graph of the multilayer perceptron.



b. Calculate the number of weights in the multilayer perceptron of (a).
   - Input – HL1 = 6 x 4 = 24
   - HL 1 – HL 2 = 4 x 4 = 16

- HL 2 – HL 3 = 4 x 2 = 8
- HL3 – Output = 3 x 2 = 6

Total = 24 + 16 + 8 + 6 = 54

c. Calculate the number of weights in the multilayer perceptron of (a) but with each li doubled, i.e., l1 = 8, l2 = 8, l3 = 4. Has the number of weights doubled as well?

- Input – HL1 = 6 x 8 = 48
- HL 1 – HL 2 = 8 x 8 = 64
- HL 2 – HL 3 = 8 x 4 = 32
- HL3 – Output = 3 x 4 = 12

Total = 48 + 64 + 32 + 12 = 156

The number of weights has not doubled because if the number of neurons in each layer is increased, this is a multiplication relationship that exists between the layers meaning that the weights exceed double.

d. Let f(p, l1, . . . , ld−1, k) be a function that computes the number of weights in the general case. Write down an expression for f.

- Weights of the First layer: $p \times l_1$
- Weights of Hidden Layers: $l_1 \times l_2 + l_2 \times l_3 + \cdots \ldots \ldots \ldots + l_{d-2} \times l_{d-1}$

  $= \sum_{l=1}^{d-2} l_i \times l_{i+1}$
- Weights of Output Layers: $l_{d-1} \times k$

Hence, the resulting expression is written as

$$f = p \times l_1 + \sum_{l=1}^{d-2} l_i \times l_{i+1} + l_{d-1} \times k$$

Exercise 7: P Classification with Neural Networks (1+1+1+1+1+1=6 Points)

In this exercise, you will implement a two-layer perceptron for predicting whether a given text was written by a human or generated by a language model. Download and use these files from Moodle (the tsv files are the same as in the last sheet):

- features-train.tsv: Feature vectors for each example in the training set.
- features-test.tsv: Feature vectors for each example in the test set.

- labels-train.tsv: Labels for each example in the training set indicating the class is_human (C = {True, False})
- programming_exercise_neural_networks.py: Template containing function stubs for the exercise. Use it with the files above as follows:
- python3 programming_exercise_neural_networks.py
- features-train.tsv labels-train.tsv features-test.tsv
- predictions-test.tsv
- requirements.txt: Requirements file for the template; can be used to install dependencies.

a. Implement a function encode_class_values to encode class values as vectors $c \in \{0, 1\}^k$ (see slide ML:IV-100).
b. Implement a function predict_probabilities that, given Wh and Wo, predicts the class probabilities for each example of a dataset.1
c. Implement a function predict that, given Wh and Wo, predicts the class for each example of a dataset (as the one with the highest probability).
d. Implement a function train_multilayer_perceptron that fits a multilayer perceptron using the IGD Algorithm (slide ML:IV-100). Like in the last exercise sheet, make sure to return the misclassification rate on both training and validation set for plotting, but this time also return the weights after each iteration (for task (e)).1
e. Select the model (iteration) that achieved the best misclassification rate on the validation set and use it to predict the label for each example in the test set (features-test.tsv). Write the predictions as one column to a file predictions-test.tsv and submit that along with your other solutions.
f. Which lines of your code would you need to change from two classes to all three? Mark each of them with the comment "# change for 3 classes".

Check for the answers to this question in the folder "Final Program" you wil find in it all necessary files -> Like the code and "predictions-test.tsv"