Exercise: 4

Meeting on January 26th / 28th

Problem 1: Maximin Strategy

In this problem you will look at various two-player-games with different reward and loss distributions. Using the maximin technique from the lecture, compute the respective maximin strategy for each of these games, where mixed strategies are allowed (aside from (d)). The tables are set up so that the first number of a cell belongs to the row player and the second number to the column player.

(a) First look at a simple coin toss, where both players choose a side of the coin simultaneously. Player B wins if they choose the same side, player A wins if the choose different sides.

	A: Heads	A: Tails
B: Heads	$1, -1$	$-1, 1$
B: Tails	$-1, 1$	$1, -1$

Table 1: Matching Pennies

(b) Now look at Two-Finger-Morra with different rewards:

	O: One	O: Two
E: One	$4, -4$	$-6, 6$
E: Two	$-4, 4$	$6, -6$

Table 2: Two-Finger-Morra variant

(c) Finally look at a variant of rock-paper-scissors. In this variant, gains and losses of rock are doubled and for scissors even tripled (i.e. if player A uses rock and player B uses paper, the loss of $A = -2$ the gain of $B = 1$). Model the game in its normal form first and then compute the maximin strategy, if only pure strategies are allowed.

(d) Again, list the normal form of rock-paper-scissors, this time with even gains and losses, and compute the maximin strategy for pure strategies.

Solution:

(a) Since the game is symmetric, we will only look at player B.

First we define p as the probability of player A choosing "heads" and analogous $(1 - p)$ for "tails". From table 1 we can compute the expected utilities based on the decision of A:

- A chooses "heads": $1 \cdot p - 1 \cdot (1-p) = 2p - 1$
- A chooses "tails": $-1 \cdot p + 1 \cdot (1-p) = 1 - 2p$

The minimum loss can be computed by setting the expressions equal and solve for p:

$$\begin{aligned} 2p - 1 &= 1 - 2p \\ 4p &= 2 \\ p &= 0.5 \end{aligned}$$

Inserted into one of the expressions above nets us an expected utility of $U_{A,B} = 0$ (and analogously $U_{B,A} = 0$, since the game is symmetrical). This strategy [0.5: Heads, 0.5: Tails] for both players is the **maximum equilibrium**.

(b) The computation follows the same rules as (a). The values in table 2 net us the following equations, starting with player E:

- O plays One: $4 \cdot p - 4 \cdot (1-p) = 8p - 4$
- O plays Two: $-6 \cdot p + 6 \cdot (1-p) = 6 - 12p$

Set equal and solve:

$$\begin{aligned} 8p - 4 &= 6 - 12p \\ 20p &= 10 \\ p &= \frac{10}{20} = \frac{1}{2} \end{aligned}$$

Hence $U_{E,O} = 0$.

For player O we get:

- E plays One: $-4 \cdot q + 6 \cdot (1-q) = 6 - 10q$
- E plays Two: $4 \cdot q - 6 \cdot (1-q) = 10q - 6$

Set equal and solve:

$$\begin{aligned} 6 - 10q &= 10q - 6 \\ 12 &= 20q \\ q &= \frac{12}{20} = \frac{3}{5} \end{aligned}$$

and inserted $U_{O,E} = 0$, i.e. no player is at an advantage in this variant.

(c) The normal form of rock-paper-scissors looks as follows:

B/A	Rock	Paper	Scissors
Rock	0,0	-2,1	$2, -3$
Paper	$1, -2$	$0, 0$	$-1, 3$
Scissors	-3,2	$3, -1$	0,0

For the best pure strategy, our goal is to maximize our minimum gain, i.e. you look at all worst-case scenarios and choose the strategy that nets the least bad worst-case. Since the game is symmetrical, we will only look at one player, here player A.

We look at each column separately, which represent our possible strategies.

- In the case of *scissors*, our possible rewards are $0, -3$ and 3, the worst case is hence -3.
- For *rock* our possibilities are $2, 0$ and -2, the worst-case is -2.
- For *paper* we gain $-1, 1$ and 0, the worst-case is then -1.

To minimize our losses (i.e. maximize our negative gain), we choose *paper* as our maximin strategy.

(d) The normal form of regular rock-paper-scissors looks as follows:

B/A	Rock	Paper	Scissors
Rock	0,0	-1,1	$1, -1$
Paper	$1, -1$	$0, 0$	$-1, 1$
Scissors	-1,1	$1, -1$	0,0

Since this game is symmetric as well, looking at only one player is sufficient. The probabilities of the mixed strategy are now p_1, p_2 und $(1-p_1-p_2)$ (since each action is of equal value, we refrain from assigning the variables to the actions explicitly). For each action we compute the expected utility U and receive three equations:

$$U = 0 - p_2 + (1 - p_1 - p_2) = -p_1 - 2p_2 + 1 \quad (1)$$
$$U = p_1 + 0 - (1 - p_1 - p_2) = 2p_1 + p_2 - 1 \quad (2)$$
$$U = -p_1 + p_2 + 0 \quad (3)$$

Subtract equation (3) from equation (2), (2) - (3):

$$\begin{aligned} U - U &= 2p_1 + p_2 - 1 - (-p_1 + p_2) \\ 0 &= 3p_1 - 1 \\ p_1 &= \frac{1}{3} \end{aligned}$$

Now subtract equation (2) from (1) and insert $p_1 = \frac{1}{3}$:

$$\begin{aligned} U - U &= -p_1 - 2p_2 + 1 - (2p_1 + p_2 - 1) \\ 0 &= -3p_1 - 3p_2 + 2 = -3p_2 + 1 \\ 3p_2 &= 1 \\ p_2 &= \frac{1}{3} \end{aligned}$$

Hence $U_{A,B} = 0$ and the maximin strategie is $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$.

Problem 2: Linear Regression (Theory) - Part 1

It was shown in the lecture that linear regression can be solved approximately with a Hill-Climbing-Algorithm.

(a) What is the linear modell for approximating the data $\{y_i, \vec{x}_i\}$?

(b) What is the equation for the error to be minimized by linear regression?

(c) What are the parameters that need to be optimized by the model?

(d) What property is given at the minimum?

(e) What is formula for the update rule of Gradient-Descent (Hill-Climbing)?

(f) Compute explicitly the derivative with respect to the optimization paramaters.

(g) Explain, how to integrate the bias w_0 into the parameter vector $\vec{w}$. What is the linear model then?

Solution:

(a) $y = \vec{w} \cdot \vec{x} + w_0$ (Linear model since all components of x only enter linearly.)

(b)

$$L = \frac{1}{N} \sum_i^N \ell_2(y_i, \vec{x}_i) = \frac{1}{N} \sum_i^N \left(y_i - (\vec{w} \cdot \vec{x}_i + w_0)\right)^2$$

(c) $\vec{w}$ and w_0, where w_0 is the bias (offset) and $\vec{w}$ the "normal vector" (see hyperplane equation)

(d) $\frac{\partial L}{\partial w_i} = 0 \; \forall \; w_i$

(e) $\Delta w_i = -\lambda \frac{\partial L}{\partial w_i}$

(f) Derivative of the loss function (Use: $\frac{d}{dx} f(x)^2 = 2 f(x) \frac{d}{dx} f(x)$)

$$\begin{aligned}
\frac{\partial L}{\partial w_0} &= -\frac{2}{N} \sum_i^N \left(y_i - (\vec{w} \cdot \vec{x}_i + w_0)\right) \\
\frac{\partial L}{\partial w_i} &= -\frac{2}{N} \sum_n^N \left(y_n - (\vec{w} \cdot \vec{x}_n + w_0)\right) (\vec{x}_n)_i, \text{ bzw.} \\
\nabla_{\vec{w}} L &= -\frac{2}{N} \sum_n^N \left(y_n - (\vec{w} \cdot \vec{x}_n + w_0)\right) \vec{x}_n
\end{aligned}$$

(g) all $\vec{x}$ are extended by a 0th component with value 1, the weight vector $\vec{w}$ receives the bias as a 0th component.

$$y = \vec{w} \cdot \vec{x}$$

(Be aware that w and x here are extended by the 0-component compared to the above equation.)

Problem 3: Linear Regression (Theory) - Part 2

Following, the bias will be integrated into the weight vector $\vec{w}$, which simplifies the following computations but keeps them exact. Linear regression is a special case of a differential equation which can be solved analyitically. This differential equation is:

$$\nabla_{\vec{w}} L = 0$$

(a) Compute the solution to the optimization problem analytically. Potentially, compute the problem first with only one component (i) of the gradient. (Solution: $\vec{w} = \vec{y}\bar{X}^{-1}$)

(b) Why does a large number of examples/data points pose a problem?

(c) Why does inserting a smart identity reduce the problem:

$$\vec{w} = \vec{y}\bar{X}^T \left(\bar{X}\bar{X}^T\right)^{-1}$$

Solution:

(a) First, the loss function can be simplified:

$$\begin{aligned}
L &= \frac{1}{N}\sum_i^N (y_i - \vec{w}\cdot\vec{x}_i)^2 \\
&= \frac{1}{N}\sum_i^N \left(y_i - (w_1, \cdots, w_M) \cdot \begin{pmatrix} x_{i1} \\ x_{i\cdots} \\ x_{iM} \end{pmatrix} \right)^2 \\
&= \frac{1}{N} \left\| (y_1, \cdots, y_N) - (w_1, \cdots, w_M) \cdot \begin{pmatrix} x_{11} & \cdots & x_{N1} \\ \cdots & \cdots & \cdots \\ x_{1M} & \cdots & x_{NM} \end{pmatrix} \right\|_2^2 \\
&= \frac{1}{N} \left\| \vec{y} - \vec{w}\bar{X} \right\|_2^2
\end{aligned}$$

Our goal is minimizing this.

- Set gradients to 0:

$$\nabla_{\vec{w}} L = \nabla_{\vec{w}} \frac{1}{N} \left\| \vec{y} - \vec{w}\bar{X} \right\|_2^2 = 0$$

- Solve the square of the absolute value:

$$\nabla_{\vec{w}} \left[\left(\vec{y} - \vec{w}\bar{X}\right)^T \left(\vec{y} - \vec{w}\bar{X}\right) \right] = 0$$

- Transpose within the parantheses:

$$\nabla_{\vec{w}} \left[\left(\vec{y}^T - \bar{X}^T\vec{w}^T\right) \left(\vec{y} - \vec{w}\bar{X}\right) \right] = 0$$

- Multiply and reposition (optional):

$$\nabla_{\vec{w}} \left[\bar{X}^T \vec{w}^T \vec{w} \bar{X} - \bar{X}^T \vec{w}^T \vec{y} - \vec{y}^T \vec{w} \bar{X} + \vec{y}^T \vec{y} \right] = 0$$

- Scalar is invariant under transposition: $\bar{X}^T \vec{w}^T \vec{y} = \vec{y}^T \vec{w} \bar{X}$:

$$\nabla_{\vec{w}} \left[\bar{X}^T \vec{w}^T \vec{w} \bar{X} - 2\vec{y}^T \vec{w} \bar{X} + \vec{y}^T \vec{y} \right] = 0$$

- Derive (w is "squared"):

$$2\bar{X}^T \vec{w}^T \bar{X} - 2\vec{y}^T \bar{X} = 0$$

- Move to other side:

$$\bar{X}^T \vec{w}^T \bar{X} = \vec{y}^T \bar{X}$$

- Multiply with inverse matrix $\left(\bar{X}^T\right)^{-1}$ and $\left(\bar{X}\right)^{-1}$:

$$\vec{w}^T = \left(\bar{X}^T\right)^{-1} \vec{y}^T$$

- Transpose

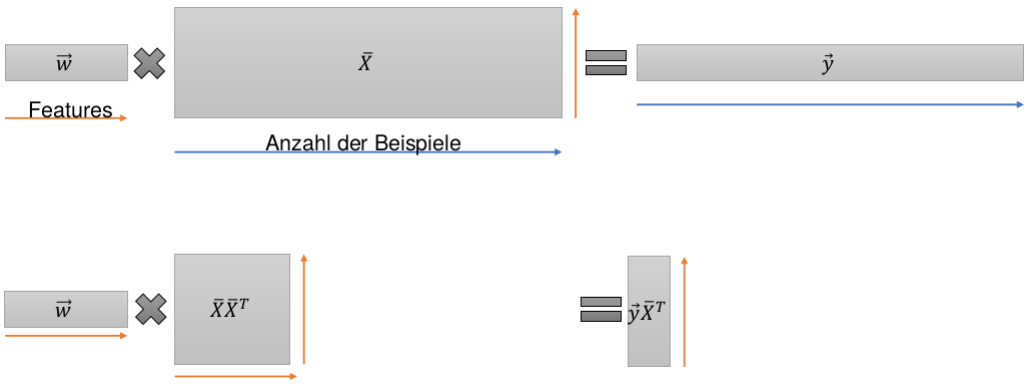
$$\vec{w} = \vec{y} \bar{X}^{-1}$$

Algebraically, minimzing means: If $\bar{X}$ is invertible, solve

$$\vec{y} = \vec{w} \cdot \bar{X} \Rightarrow \vec{w} = \vec{y} \bar{X}^{-1}$$

(b) Inverting the matrix is very costly ($\mathcal{O}(N^3)$)

(c) $\bar{X}\bar{X}^T$ only has the dimension of the feature vector, hence inverting is independent on the number of data points. Concretely:



Problem 4: Activation function

A Multilayer-Perceptron (MLP) is a kind of neural network and hence a simple, non-linear classifier, consiting of an input, a hidden and an output layer of "nodes"/vectors (see Fig. 1)

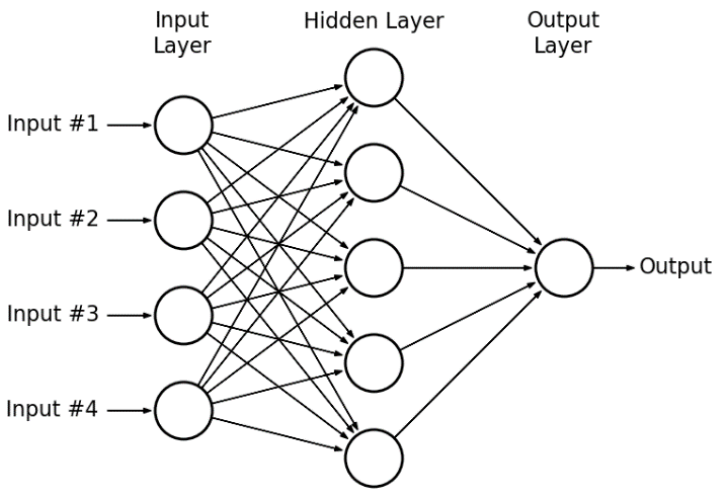


Figure 1: Beispiel für ein Multilayer-Perceptron

The hypothesis room $\mathcal{H}$ of MLPs is given by

$$y = f(\vec{x}) = \vec{w} \cdot r(\bar{V}\vec{x}) \; . \tag{4}$$

Where is r a non linear function, since the above equation could otherwise be reduced to a linear equation.

(a) Show this by setting $r(x) = x$. Following that, why is a non-linear activation function r necessary?

(b) Show that:

- $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ with $\sigma(x) = \frac{1}{1+e^{-x}}$
- $\tanh'(x) = 1 - \tanh^2(x)$ with $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- $\text{ReLU}'(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x > 0 \\ \text{not defined for } x = 0 \end{cases}$ with $\text{ReLU}(x) = \max(0, x)$

Solution:

(a) Converstion for $r(x) = x$:

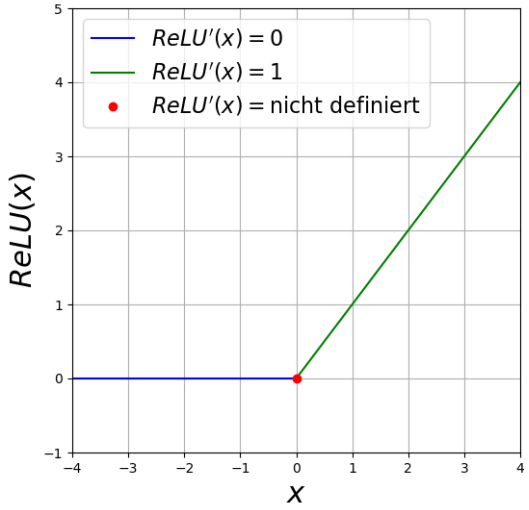
$$\begin{aligned} y &= \vec{w} \cdot \bar{V}\vec{x} \\ &= \underbrace{\vec{w} \cdot \bar{V}}_{\vec{w}'} \cdot \vec{x} \\ &= \vec{w}' \cdot \vec{x} \end{aligned}$$

Hence, the equation can be reformed to a linear equation.

The non-linear function is what allows the MLP to divide the space non-linearly. Without the activation function, the space is only linearly divisible.

(b)

- $\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} = \sigma(x) \cdot \frac{e^{-x}}{1+e^{-x}} = \sigma(x) \cdot \frac{1+e^{-x}-1}{1+e^{-x}}$
  $= \sigma(x) \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) = \sigma(x) \cdot (1 - \sigma(x))$
- $\tanh'(x) = \frac{(e^x+e^{-x})\cdot(e^x+e^{-x}) - (e^x-e^{-x})\cdot(e^x-e^{-x})}{(e^x+e^{-x})^2} = \frac{(e^x+e^{-x})^2}{(e^x+e^{-x})^2} - \left(\frac{e^x-e^{-x}}{e^x+e^{-x}}\right)^2$
  $= 1 - \tanh^2(x)$
- The given values for $\mathrm{ReLU}'(x)$ can be read from the graph of $\mathrm{ReLU}(x)$ directly:



Problem 5: Programming a perceptron

Given is a sequence of numbers $Z_i \in \{0,1\}$. Program and learn a multi-class perceptron with gradient descent as an optimizer, which predicts the next number from the sequence of the last 10 numbers. This means, the net has 10 input nodes with values $Z_{t-9} \rightarrow Z_t$ and should predict Z_{t+1}. Use softmax as the output function and one node for 0 and 1 respectively. The sequence for testing the algorithm is given by random numbers which are entered manually (by the command line) to the program.

Use the following sequence as a clue:

1. Initialize the saved sequence (i.e. last 10 elements) by e.g. random numbers.
2. Initialize the weight vector W of the perceptron with e.g. random numbers.
3. Repeat:
   (a) Compute the next prediction of the net p_{t+1}.
   (b) Read the next number z_{t+1} from the console.
   (c) Compare p_{i+1} and z_{i+1} and perhaps adjust W.
   (d) Save z_{t+1} into the sequence of the last 10 elements (input data).
   (e) Ouput of the percentage of all correctly predicted numbers.

What is the perceptron's accuracy? Hence, are you a reliable random number generator?

Solution:

Example code in Python:

```python
import numpy as np


def softmax(x):
    return np.exp(x) / np.sum(np.exp(x))


random_string = input("Type your random numbers {0, 1}: ")
random_string = random_string.replace(' ',"")

print(random_string)

# read string from shell
random_list = np.asarray(list(map(int, random_string)))

# init net
input_neurons = 10
layer_neurons = 2

# initialize weights W, biases b and buffer X with random numbers
W = np.random.rand(input_neurons, layer_neurons) - 0.5
b = np.random.rand(layer_neurons)
X = np.random.randint(0, 2, input_neurons)

# define the learning reate
learnrate = 0.2

# initialize counter of correct prediction
N = 0
correct = 0

# loop until user stops script (terminate)
for digit in random_list:
    # digit can be two output vectors corresponding to 1/0
    if digit > 0:
        digit = np.array([0, 1])
    else:
        digit = np.array([1, 0])

    # predict next, increment N
    N += 1
    h = X.dot(W) + b
    predicted = softmax(h)

    # check result and learn if required (see script)
    if np.argmax(predicted) == np.argmax(digit):
        correct += 1

    W -= learnrate * np.outer(X, (predicted - digit))
    b -= learnrate * (predicted - digit)

    # store to replay
```

```
53    X[:-1] = X[1:]
54    X[-1] = np.argmax(digit)
55
56# output results
57# print('Buffer %s' % X)
58# print('Predicted %d, Correct %d, Accuracy %f %%' % (predicted, digit, (correct * 100) /
      ↪ N))
59
60print('Accuracy %f %%' % ((correct * 100) / N))
```