Exercise 2 - Neural Networks

Alexander Bigalke, Arthur Heimbrecht, Robin Rombach

May 16, 2018

I. Classification Capacity

- see attached handwritten notes beginning next page -

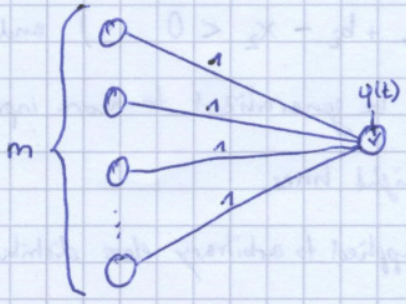
Sheet 2

exercise 1.1)

1. logical or :

To perform a logical OR operation on an input vector $z \in \{0,1\}^m$, we choose the weight vector $w \in \mathbb{R}^m$ to be a vector of 1s ($w = \underbrace{(1,1,1,\dots,1,1)}_{m}$) and the activation function to be a step function

$$\varphi(t) = \begin{cases} 0 & \text{if } t < 0.5 \\ 1 & \text{if } t \geq 0.5 \end{cases}$$



2. check if input $z \in \{0,1\}^m$ exactly coincides with $c \in \{0,1\}^m$

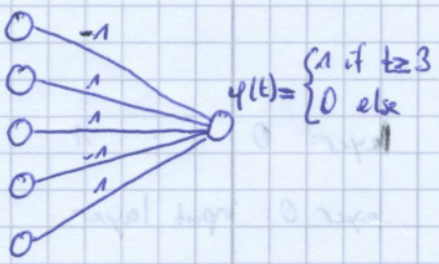
Here, we choose our weight vector $w \in \mathbb{R}^m$ to be

$$w_i = \begin{cases} 1 & \text{if } c_i = 1 \\ -1 & \text{if } c_i = 0 \end{cases} \quad \text{or} \quad w = 2c - 1$$

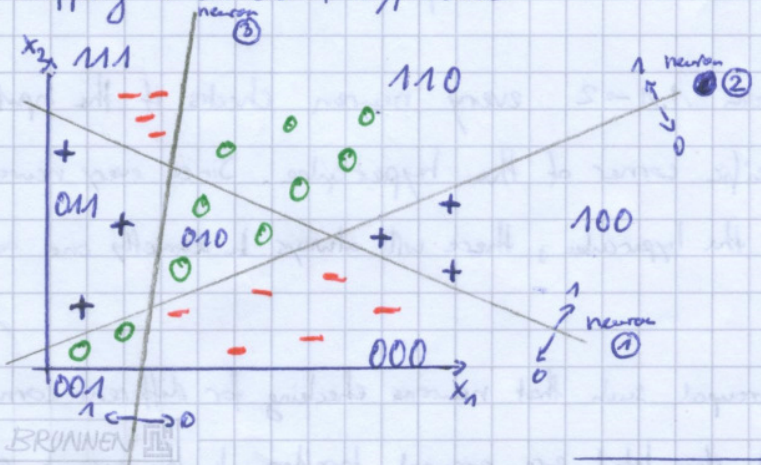
and again a step function as activation

$$\varphi(t) = \begin{cases} 1 & \text{if } t \geq \|c\|_1 \\ 0 & \text{else} \end{cases} \qquad \text{with the 1-norm } \|.\|_1$$

For $c = (0,1,1,0,1)$, our network e.g. looks like:



3. mapping to corners of hypercube



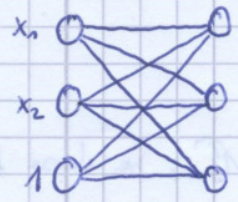
The decision boundary of each neuron can be described by a straight line

$$f_i(x_1) = a_i x_1 + b_i .$$

Given the parameters a_i, b_i $(i = 1,2,3)$ the neural network which

Note that input data cannot be mapped to the corner 101 for this choice of decision boundaries.

maps the input data to the corner of a hypercube looks like



with the weight matrix $W = \begin{pmatrix} a_1 & -1 & b_1 \\ a_2 & -1 & b_2 \\ a_3 & -1 & b_3 \end{pmatrix}$

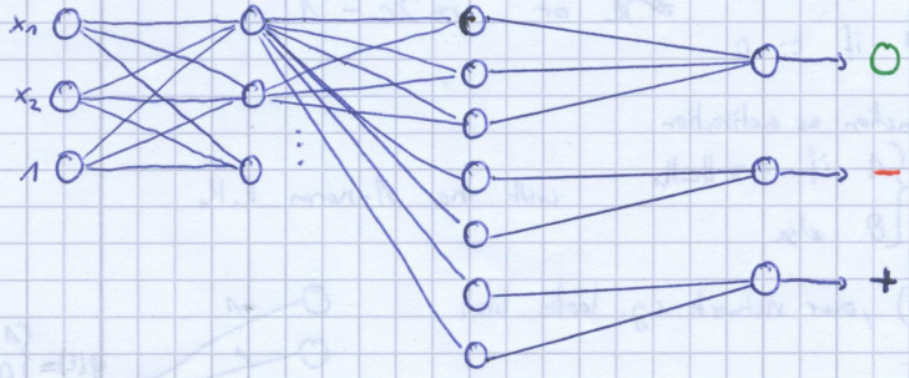
and the activationfunction $\varphi(t) = \begin{cases} 1 & \text{if } t \leq 0 \\ 0 & \text{if } t > 0 \end{cases}$

In words, that means, that a neuron outputs "1" if the input is above its decision boundary ($a_i \cdot x_1 + b_i - x_2 < 0$) and "0" if the input is below.

This can easily be generalized to more input dimensions by using hyperplanes as decision boundaries instead of straight lines.

It can also be applied to arbitrary class distributions by using more decision boundaries, i.e. more neurons.

exercise 1.2) For the given example, the 3-layer network would look as follows:



layer: 0 1 2 3

layer 0: input layer

layer 1: maps the input data to the corners of a hypercube (according to 1.1→3). The number of neurons is given by the required number of decision boundaries to separate the data.

layer 2: using the architecture from 1.1→2 every neuron checks if the input has been mapped to a specific corner of the hypercube. Since every neuron checks for a different corner of the hypercube, there will always be exactly one active neuron in this layer.

The neurons should be grouped such that neurons checking for different corners with the same associated class label are grouped together. In the present case,

the first 3 neurons check for $(0,0,1)$, $(0,1,0)$, $(1,1,0)$, neurons 4,5 check for $(1,1,1)$, $(0,0,0)$ and neurons 6,7 check for $(0,1,1)$, $(1,0,0)$.

In general, the number of neurons in this layer is given by $H_2 = 2^{H_1}$.

In this case, we only have $H_2 = 7 = 2^{H_1} - 1$, since the corner $(1,0,1)$ "does not exist".

layer 3: In this layer, there is one neuron for each class label (here for $-, 0, +$).

~~and each neuron checked~~ Each of these neurons is then only connected to the neurons of the previous layer which checked for corners of the hypercube associated with the corresponding class label.

Using the logical OR from $1.1 \to 1$, each neuron then detects if one of its preneurons in layer 2 is active, i.e. if the data has been mapped to a corner which is associated with its class label ~~and then respond~~ and responds a "1" in this case and a "0" else.

problems)

- the number of neurons in layer 2 grows exponentially with the number of required decision boundaries
- the generalization to an independent test set only works, if the data are grouped in the same way as the training data, i.e. the test set must have the same decision boundaries

  → the network tends to overfitting

II. Linear Activation Function

In a feed-forward network, the output of each layer l (denoted $Z_l \in \mathbb{R}^{m_l}$; with m_l some integer specifying the number of features in layer l.) is computed iteratively by multiplying the weight matrix $B_l \in \mathbb{R}^{m_l \times m_{l-1}}$ with the output of layer $l-1$ and subsequently applying the activation function φ_l to this product:

$$\tilde{Z}_l = B_l \cdot Z_{l-1} \tag{1}$$
$$Z_l = \varphi_l(\tilde{Z}_l) \tag{2}$$

where the input Z_0 is given as

$$Z_0 = \begin{pmatrix} 1 \\ \mathbf{X} \end{pmatrix}$$

with $\mathbf{X}$ denoting the input (training) data.

Let us assume that the activation function is a linear map, i.e.

$$Z_l = \varphi_l(\tilde{Z}_l) = \Phi_l \cdot \tilde{Z}_l \tag{3}$$

such that Φ_l is a quadratic matrix with $\Phi_l \in \mathbb{R}^{m_l \times m_l}$.
Combining equations (1) and (3) and exploiting associativity of matrix multiplication then yields the following output for layer l:

$$Z_l = \Phi_l \cdot \tilde{Z}_l = \Phi_l \cdot B_l \cdot Z_{l-1} \equiv \mathcal{B}_l \cdot Z_{l-1} \,. \tag{4}$$

Hence,

$$\mathbb{R}^{m_l \times m_{l-1}} \ni \mathcal{B}_l = \Phi_l \cdot B_l$$

Expanding (4) gives:

$$Z_l = \underbrace{\mathcal{B}_l \cdot \mathcal{B}_{l-1} \cdot \mathcal{B}_{l-2} \cdot \cdots \cdot \mathcal{B}_1}_{\equiv \mathbf{B}_l^0 \in \mathbb{R}^{m_l \times m_0}} \cdot Z_0 \tag{5}$$

Thus, in the case of a linear activation function and by contracting the product $\mathcal{B}_l \cdot \mathcal{B}_{l-1} \cdot \mathcal{B}_{l-2} \cdot \cdots \cdot \mathcal{B}_1$, the whole procedure of mapping the input layer 0 to layer l via several intermediate layers $l-i$ ($i \in [0, l]$) reduces to mapping layer 0 directly to layer l by

$$Z_l = \mathbf{B}_l^0 \cdot Z_0 \tag{6}$$

with $\mathbf{B}_l^0 \in \mathbb{R}^{m_l \times m_0}$.

III. Application

The completed code can be found in the file `network.py`.

We report results for the following cases:

i) `layer_sizes = [2,2, n_classes]`
ii) `layer_sizes = [3,3, n_classes]`
iii) `layer_sizes = [5,5, n_classes]`
iv) `layer_sizes = [30,30, n_classes]`
with `n_classes = 2`.

For the other hyper-parameters, we use:
`n_epochs = 500`; `batch_size = 200`; `learning_rate = 0.0001`

The task is to classify a dataset provided by `sklearn.datasets.make_moons`. A visualized example-set is provided in Figure 1.
We account for the stochastic nature of the model (induced by the stochastic gradient descent used here) by repeating each classification run (for a fixed set of hyper-parameters) $n = 10$ times and subsequently computing mean, median and standard deviation of the resulting error rate(s). In the following, let γ_t

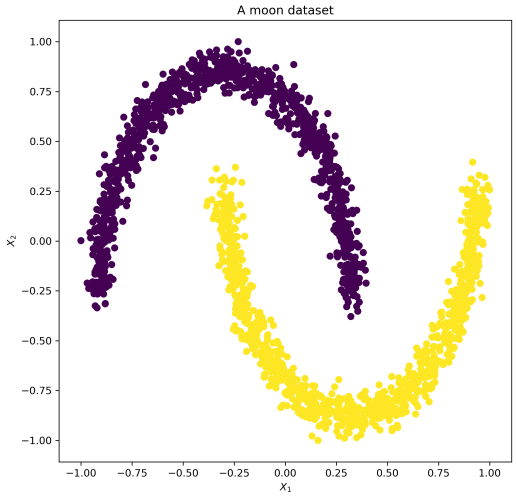
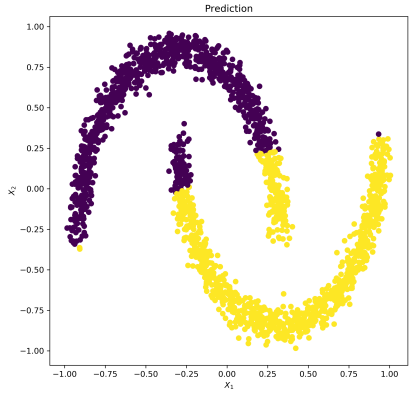


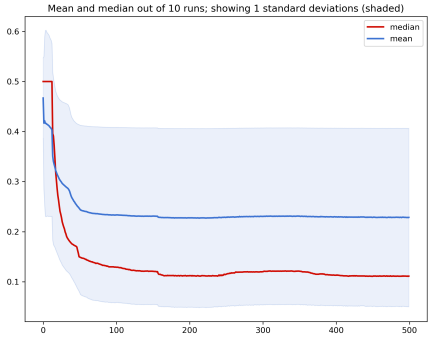
Figure 1: An instance of `sklearn.datasets.make_moons` with 2 classes.

denote the error rate on the test set.

i): `layer_sizes = [2,2, n_classes]`



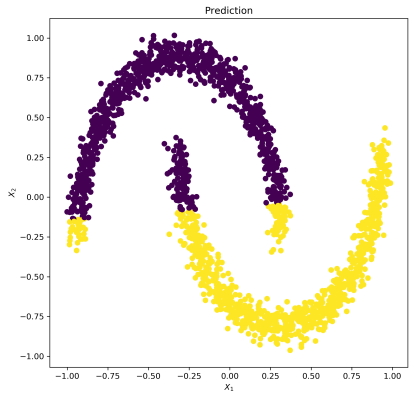
Classification outcome on test set



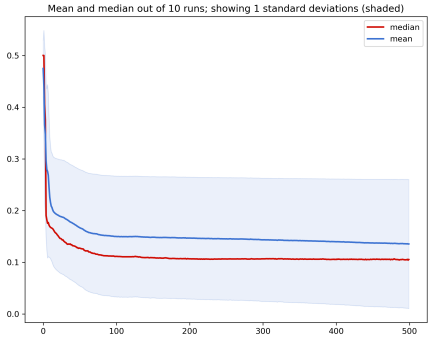
Training set error rate: mean and median with standard deviation; composed out of 10 runs.

Figure 2: Error rate on test set: $\gamma_t = 0.1195$. NN is not able to form sufficient decision boundaries.

ii): `layer_sizes = [3,3, n_classes]`



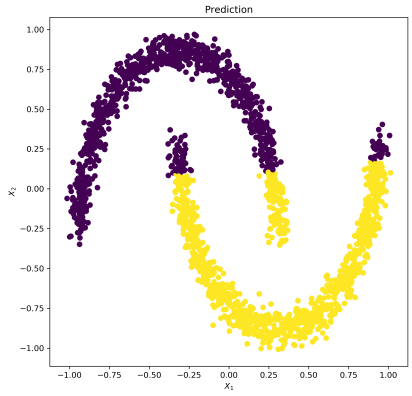
Classification outcome on test set



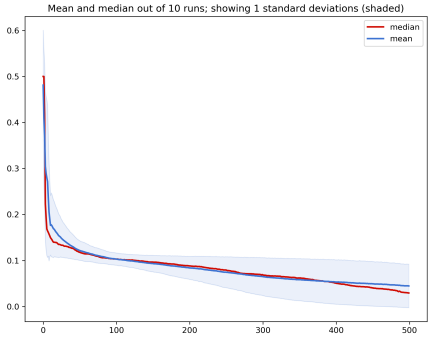
Training set error rate: mean and median with standard deviation; composed out of 10 runs.

Figure 3: Error rate on test set: $\gamma_t = 0.104$. NN is still not able to form sufficient decision boundaries.

iii): `layer_sizes = [5,5, n_classes]`



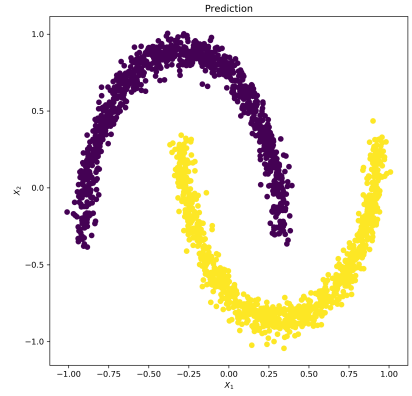
Classification outcome on test set



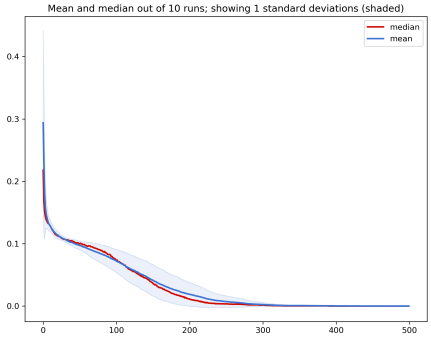
Training set error rate: mean and median with standard deviation; composed out of 10 runs.

Figure 4: Error rate on test set: $\gamma_t = 0.0985$. NN begins to form sufficient decision boundaries.

iv): `layer_sizes = [30,30, n_classes]`



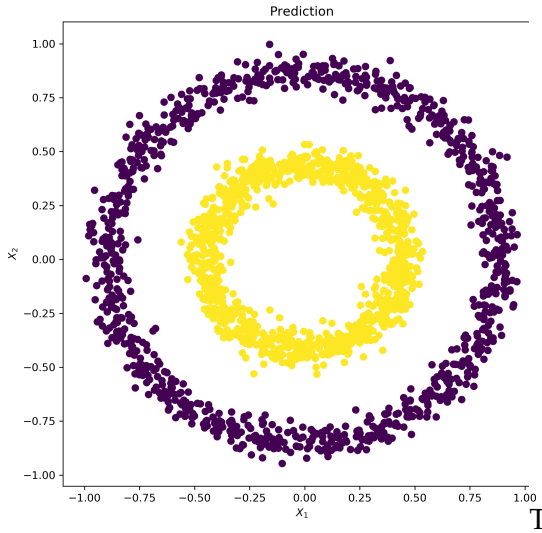
Classification outcome on test set



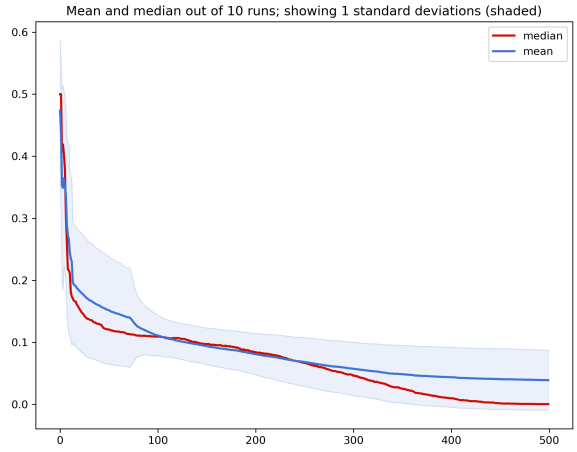
Training set error rate: mean and median with standard deviation; composed out of 10 runs.

Figure 5: Error rate on test set: $\gamma_t = 0.0$. NN is able to classify the system flawlessly.

Additionally, it is worth mentioning that the NN is also able to classify other datasets, such as `sklearn.datasets.make_circles` (see Figure 6) flawlessly.



Classification outcome on test set



Training set error rate: mean and median with standard deviation; composed out of 10 runs.

Figure 6: *Circles* dataset. Error rate on test set: $\gamma_t = 0.0$. NN is able to classify the system flawlessly.