Deep Learning

Homework 1

Group 71 members:

- Luis Jose De Macedo Guevara (95621)

- Vincent Jakl (108529)

Contributions of each member

- Luis Jose De Macedo Guevara (95621):

- Vincent Jakl (108529):

Question 1

1. a)

The single perceptron was not the best choice for this task. As can be seen in the figure below, the perceptron was not able to fit to the high dimensional data also with no sign of improvement. With the 20 epochs, it ended up with a 0.3422 test accuracy which is slightly higher than 0.25 that would happen with random selection for 4 classes. Also the learning rate was left at the value of 1 so this might cause the problem as well.
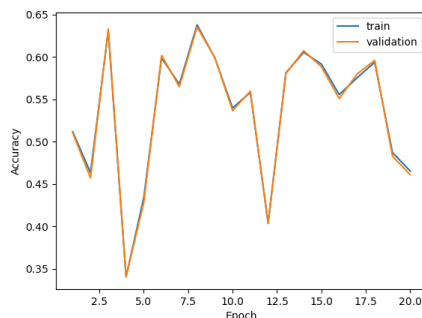

Figure 1: Single perceptron

1. b)

The logistic regression performed better than the single perceptron. As can be seen in the figure below, the logistic regression was able to fit to the data better. There was however a big difference between the two learning rates. Where the model with the learning rate of 0.01 jumped faster to the higher accuracy, it did not improve on a regular basis. The 0.001 lr model was a bit slower to get up to higher accuracy, but it was able to consistently improve. In the end the 0.01 lr model ended up with 0.5784 test accuracy and the 0.001 lr model ended up with 0.5936 test accuracy. Running these models for more epochs would possibly improve the accuracy since the training was still showing signs of improvement even with the evaluation metric.

2. a)

We do believe that the statement is correct. The multiplayer network can be more expressive than single layer logistic regression. The logistic regression is
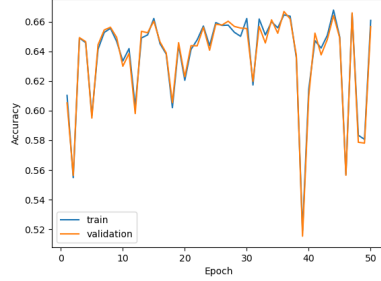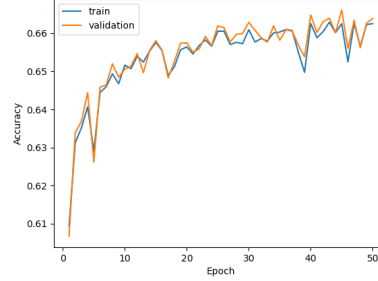
(a) lr 0.01   (b) lr 0.001

Figure 2: Logistic regression

a linear model and therefore it can only learn linear decision boundaries. The multi layer network can learn non-linear decision boundaries. However with it being non-linear there can be a problem with local minima for example, which we will not have with the linear model.

2. b)

As seen Figure 3, we can see that the multilayer perceptron, as specified, is a better fit for the data, not only its accuracy on both training and validation data largely improves with each epoch, so does its training loss. It also shows a good test accuracy of 0.7580 after 20 epochs. Looking at the trend in both graphs, we can expect a better accuracy on the training/validation data with more epochs and possibly, taking care to avoid overfitting, a better testing accuracy.
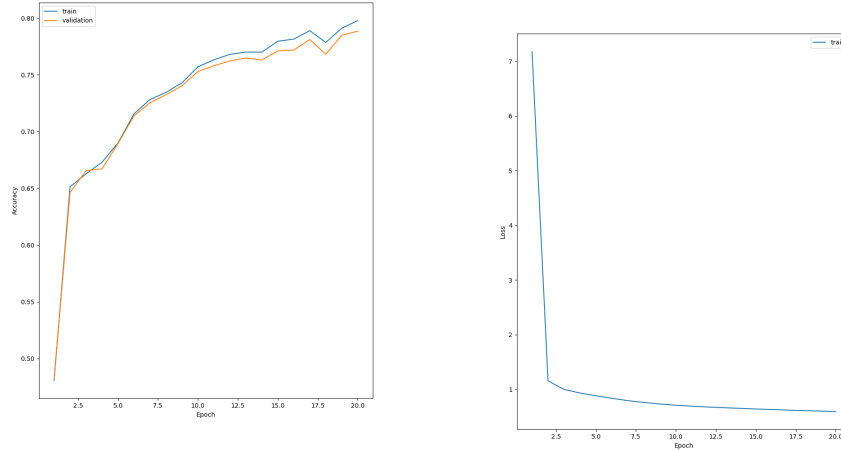
(a) Training/Validation accuracy per epoch

(b) Loss as a function of epoch number

Figure 3: MLP w/o Pytorch

Question 2

1.

After implementing the logistic regression with Pytorch, the best performing learning rate was 0.001 with the final test accuracy of 0.6503. All the test accuracies are as follows 0.5577, 0.6200, 0.6503 in the order of learning rate 0.1, 0.01, 0.001. Again here it looks like 20 training epochs were not enough to get the best performance out of the model. In the graphs below, it is visible that the model was still improving on the training data. Also it is noticable, how the 0.1 learning rate model had maybe too high of a learning rate since on the graph, the validation accuracy is jumping all around.

2. a)

When training the MLP with Pytorch, the batch size of 16 trained slower than the batch size of 1024. The 16 batch size finished in 678 seconds whereas the 1024 batch size finished in 353 seconds. However, since the 16 batch size trained on overall more batches, it was able to perform a little bit better The final testing accuracy of the 16 batch size was 0.7372 whereas the 1024 batch size was 0.7183. Once again we can see here, that more epoch would
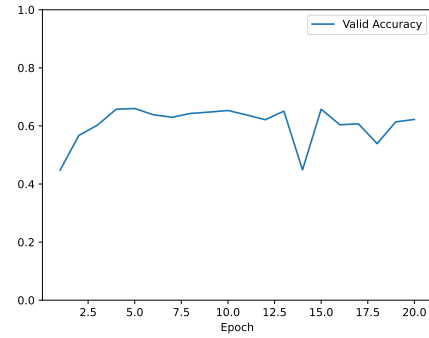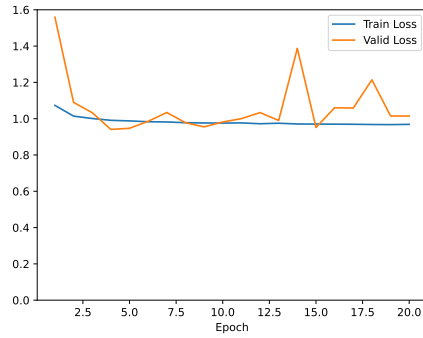
(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch
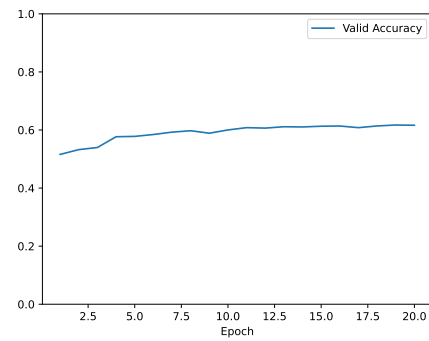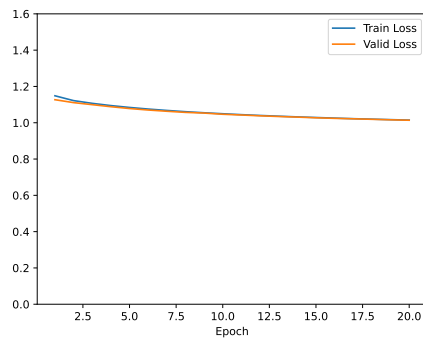
Figure 4: Logistic regression lr 0.1
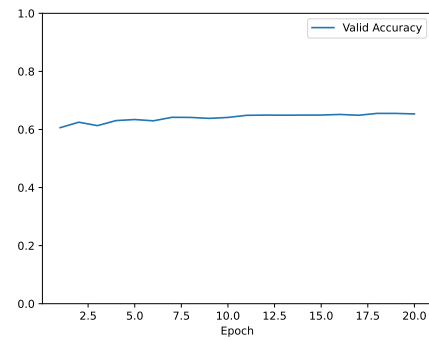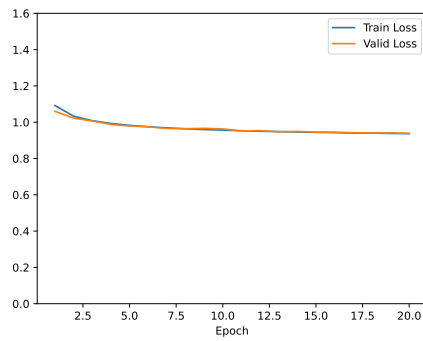
(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch

Figure 5: Logistic regression lr 0.01

(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch

Figure 6: Logistic regression lr 0.01
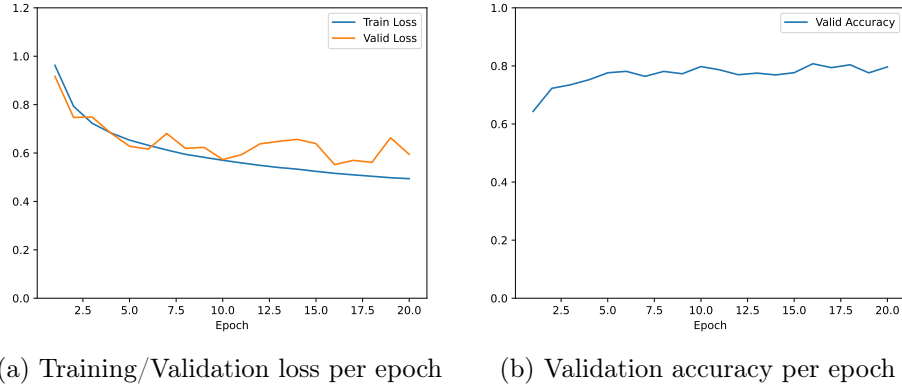
5

probably improve the performance of the model.

(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch

Figure 7: MLP batch size 16

(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch
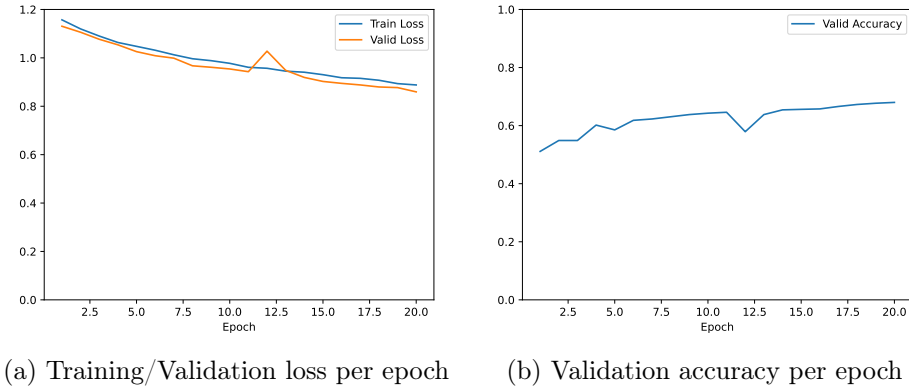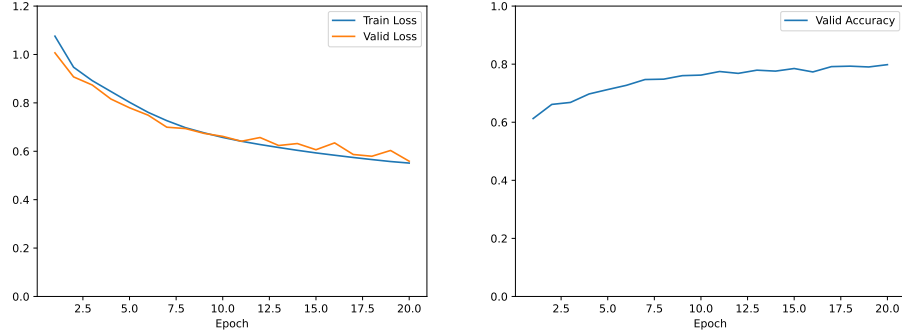
Figure 8: MLP batch size 1024

2. b)

Out of the four learning rates, the worst performing one was lr 1. This makes sense, since the gradient descent probably overshoots the minimum. The best performing learning rate was 0.01 with the final test accuracy of 0.7580, whereas the learning rate of 1 had the final test accuracy of 0.4726.

(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch

Figure 9: MLP lr 0.01
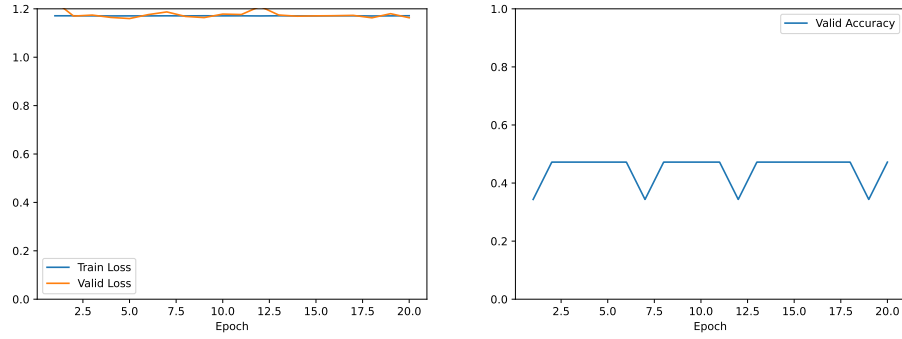
(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch

Figure 10: MLP lr 1

2. c)

When training the network for 150 epochs and 256 batch size, the model gets to higher accuracies than the previous models. However in the trainig, validation loss graph we can see, that the trainig and validation loss start to diverge after around 60 epochs. This is a sign of overfitting. When comparing the L2 regularization and droupout regularization, the better final accuracy test is better with the L2 regularization. However the dropout regularization train/validation loss graph looks like it would still benefit from more training, since the training and validation loss are both still improving, and not diverging. The final test accuracy for the L2 regularization was 0.7864 and for the dropout regularization it was 0.7845. This makes both the models, the best performing models out of all the models trained in this

7

question.
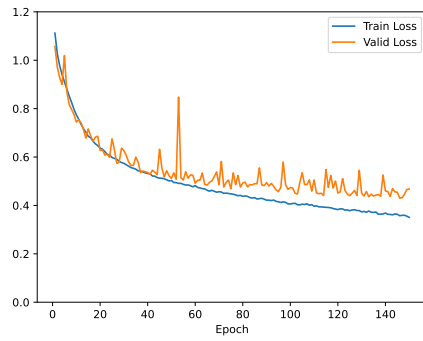
(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch

Figure 11: MLP overfitting
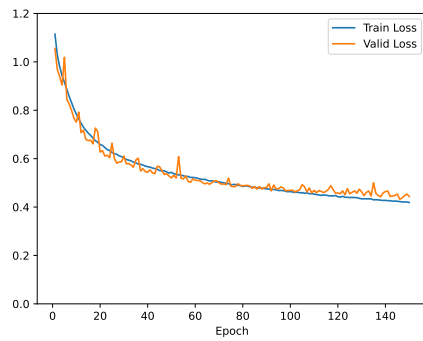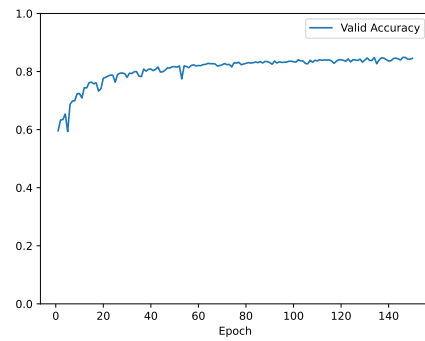
(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch

Figure 12: MLP dropout
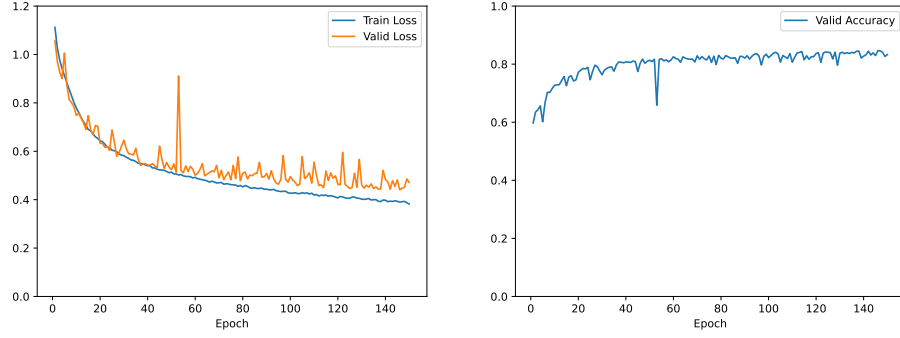
(a) Training/Validation loss per epoch   (b) Validation accuracy per epoch
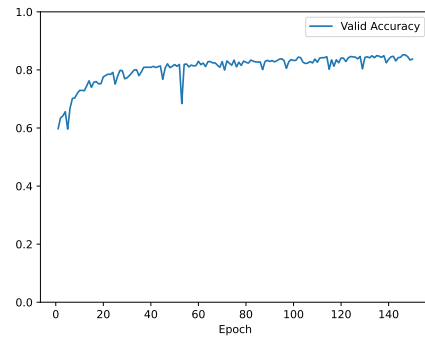
Figure 13: MLP L2 refularization

Question 3

1.

a)

Let

- $D = 2$

- $A = B = 0$

- $\boldsymbol{x}^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\boldsymbol{x}^{(2)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $\boldsymbol{x}^{(3)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ and $\boldsymbol{x}^{(4)} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$,

We have $f(\boldsymbol{x}^{(1)}) = -1$, $f(\boldsymbol{x}^{(2)}) = +1$, $f(\boldsymbol{x}^{(3)}) = +1$ and $f(\boldsymbol{x}^{(4)}) = -1$

9
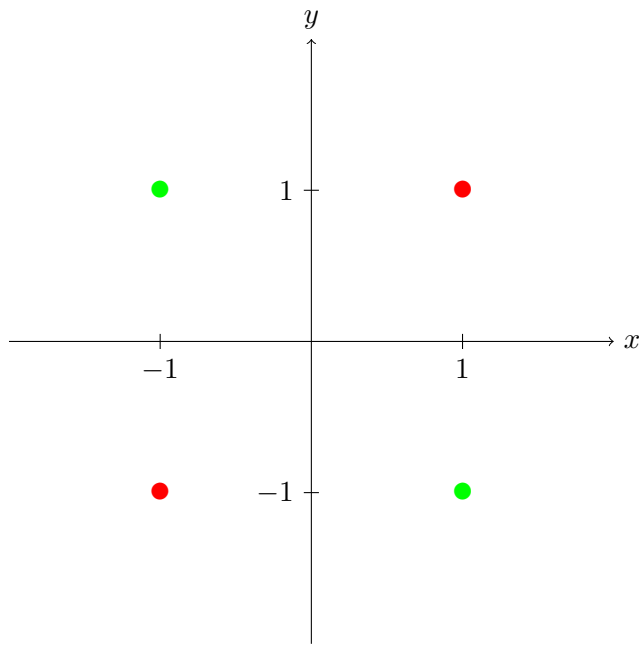
Which we can see is not linearly separable and therefore a perceptron cannot learn a separating hyperplane.

b)
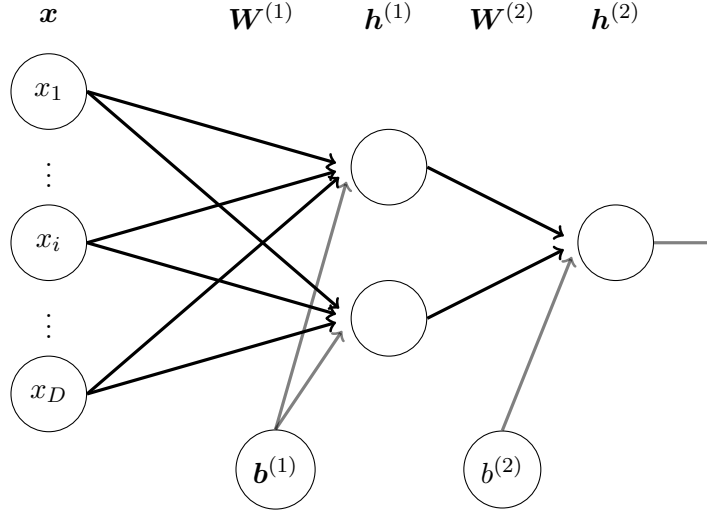
$$\boldsymbol{W}^{(1)} = \underbrace{\begin{bmatrix} - & \boldsymbol{w}_1^{(1)} & - \\ - & \boldsymbol{w}_2^{(1)} & - \end{bmatrix}}_{2\times D}, \qquad\qquad \boldsymbol{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_1^{(2)} & w_2^{(2)} \end{bmatrix}, \qquad\qquad b^{(2)}$$

We have that $A \le \sum_{i=1}^{D} x_i \le B$ iff

- $\sum_{i=1}^{D} x_i \le A \iff \sum_{i=1}^{D} x_i - A \ge 0$ and

- $\sum_{i=1}^{D} x_i \ge B \iff B - \sum_{i=1}^{D} x_i \ge 0.$

Knowing that

$$\boldsymbol{h}^{(1)} = \mathrm{sign}\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) = \begin{bmatrix} \mathrm{sign}\left(\boldsymbol{w}_1^{(1)}\boldsymbol{x} + b_1^{(1)}\right) \\ \mathrm{sign}\left(\boldsymbol{w}_2^{(1)}\boldsymbol{x} + b_2^{(1)}\right) \end{bmatrix}$$

We can set $\boldsymbol{W}^{(1)} = \begin{bmatrix} 1 & \cdots & 1 \\ -1 & \cdots & -1 \end{bmatrix}$ and $\boldsymbol{b}^{(1)} = \begin{bmatrix} -A \\ B \end{bmatrix}$ to get

$$\boldsymbol{h}^{(1)} = \begin{bmatrix} \mathrm{sign}\left(\sum_{i=1}^{D} x_i - A\right) \\ \mathrm{sign}\left(B - \sum_{i=1}^{D} x_i\right) \end{bmatrix}$$

11

which results in

$$
\boldsymbol{h}^{(1)} = \begin{cases} \begin{bmatrix} +1 \\ +1 \end{bmatrix} & \text{if } A \leq \sum_{i=1}^{D} x_i \leq B \\[2ex] \begin{bmatrix} -1 \\ +1 \end{bmatrix} & \text{if } \sum_{i=1}^{D} x_i < A \leq B \\[2ex] \begin{bmatrix} +1 \\ -1 \end{bmatrix} & \text{if } A \leq B < \sum_{i=1}^{D} x_i \end{cases}
$$

In order to learn the function f, we just need to choose $\boldsymbol{W}^{(2)}$ and $b^{(2)}$ such that

$$
\boldsymbol{h}^{(2)} = \begin{cases} +1 & \text{if } \boldsymbol{h}^{(1)} = \begin{bmatrix} +1 \\ +1 \end{bmatrix} \\ -1 & \text{if } \boldsymbol{h}^{(1)} \neq \begin{bmatrix} +1 \\ +1 \end{bmatrix} \end{cases}
$$

equivalently

- $\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + b^{(2)} \geq 0$ when $\boldsymbol{h}^{(1)} = \begin{bmatrix} +1 \\ +1 \end{bmatrix}$

- $\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + b^{(2)} < 0$ when $\boldsymbol{h}^{(1)} \neq \begin{bmatrix} +1 \\ +1 \end{bmatrix}$

Which we can achieve by

1. setting $\boldsymbol{W}^{(2)} = \begin{bmatrix} 1 & 1 \end{bmatrix}$, from which we get $\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)}$ equal to 2, 0 and 0 for $\begin{bmatrix} +1 \\ +1 \end{bmatrix}$, $\begin{bmatrix} -1 \\ +1 \end{bmatrix}$ and $\begin{bmatrix} +1 \\ -1 \end{bmatrix}$ respectively, and

2. setting $b^{(2)} = -1$, from which we get $\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + b^{(2)}$ equal to 1, -1 and -1 for $\begin{bmatrix} +1 \\ +1 \end{bmatrix}$, $\begin{bmatrix} -1 \\ +1 \end{bmatrix}$ and $\begin{bmatrix} +1 \\ -1 \end{bmatrix}$ respectively.

and so we learn our desired function with the weights and biases:

$$
\boldsymbol{W}^{(1)} = \underbrace{\begin{bmatrix} 1 & \cdots & 1 \\ -1 & \cdots & -1 \end{bmatrix}}_{2 \times D}, \qquad\qquad \boldsymbol{b}^{(1)} = \begin{bmatrix} -A \\ B \end{bmatrix}
$$

$$
\boldsymbol{W}^{(2)} = \begin{bmatrix} 1 & 1 \end{bmatrix}, \qquad\qquad b^{(2)} = -1
$$

c)
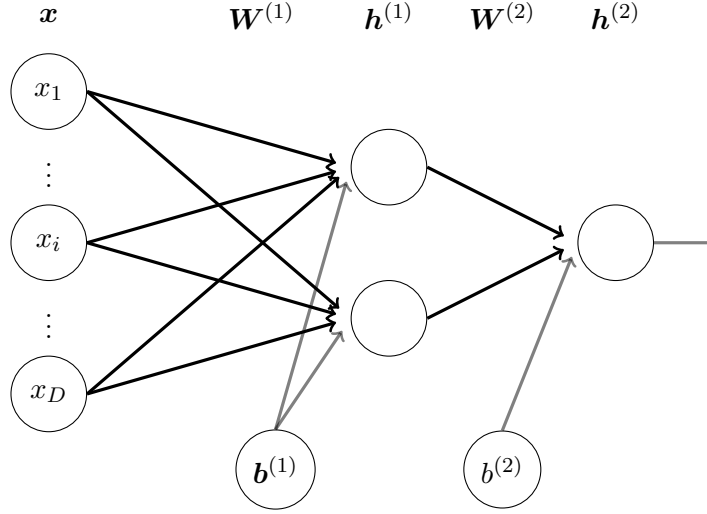
$$\boldsymbol{W}^{(1)} = \underbrace{\begin{bmatrix} - & \boldsymbol{w}_1^{(1)} & - \\ - & \boldsymbol{w}_2^{(1)} & - \end{bmatrix}}_{2 \times D}, \qquad \boldsymbol{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_1^{(2)} & w_2^{(2)} \end{bmatrix}, \qquad b^{(2)}$$

We have that $A \leq \sum_{i=1}^{D} x_i \leq B$ iff

- $\sum_{i=1}^{D} x_i \leq A \iff A - \sum_{i=1}^{D} x_i \geq 0$ and

- $\sum_{i=1}^{D} x_i \geq B \iff \sum_{i=1}^{D} x_i - B \geq 0$.

Knowing that

$$\boldsymbol{h}^{(1)} = \text{ReLU}\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) = \begin{bmatrix} \text{ReLU}\left(\boldsymbol{w}_1^{(1)}\boldsymbol{x} + b_1^{(1)}\right) \\ \text{ReLU}\left(\boldsymbol{w}_2^{(1)}\boldsymbol{x} + b_2^{(1)}\right) \end{bmatrix}$$

We can set $\boldsymbol{W}^{(1)} = \begin{bmatrix} -1 & \cdots & -1 \\ 1 & \cdots & 1 \end{bmatrix}$ and $\boldsymbol{b}^{(1)} = \begin{bmatrix} A \\ -B \end{bmatrix}$ to get

$$\boldsymbol{h}^{(1)} = \begin{bmatrix} \text{ReLU}\left(A - \sum_{i=1}^{D} x_i\right) \\ \text{ReLU}\left(\sum_{i=1}^{D} x_i - B\right) \end{bmatrix}$$

13

which results in

$$\boldsymbol{h}^{(1)} = \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \end{bmatrix} \text{ where } \begin{cases} h_1^{(1)} = 0 & \text{if } \sum_{i=1}^{D} x_i \geq A \\ h_1^{(1)} > 0 & \text{if } \sum_{i=1}^{D} x_i < A \\ h_2^{(1)} = 0 & \text{if } \sum_{i=1}^{D} x_i \leq B \\ h_2^{(1)} > 0 & \text{if } \sum_{i=1}^{D} x_i > B \end{cases}$$

In order to learn the function, we just need to choose $\boldsymbol{W}^{(2)}$ and $b^{(2)}$ such that

$$\boldsymbol{h}^{(2)} = \begin{cases} +1 & \text{if } \boldsymbol{h}^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ -1 & \text{otherwise} \end{cases}$$

equivalently

- $\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + b^{(2)} \geq 0$ when $\boldsymbol{h}^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

- $\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + b^{(2)} < 0$ when $\boldsymbol{h}^{(1)} \neq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Knowing that $h_1^{(1)} \geq 0$ and $h_2^{(1)} \geq 0$ we just need to set $\boldsymbol{W}^{(2)} = \begin{bmatrix} -1 & -1 \end{bmatrix}$ and $b^{(2)} = 0$, from which we get $\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + b^{(2)}$ equal to 0 when $\boldsymbol{h}^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, and negative otherwise.

and so we learn our desired function with the weights and biases:

$$\boldsymbol{W}^{(1)} = \underbrace{\begin{bmatrix} -1 & \cdots & -1 \\ 1 & \cdots & 1 \end{bmatrix}}_{2 \times D}, \qquad \boldsymbol{b}^{(1)} = \begin{bmatrix} A \\ -B \end{bmatrix}$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} -1 & -1 \end{bmatrix}, \qquad b^{(2)} = 0$$