Deep Learning

Masters in Electrical and Computer Engineering

Homework 1

Authors:

Duarte Cerdeira – 96195 duarte.cerdeira@tecnico.ulisboa.pt
Pedro Jacinto – 97352 pedro.m.l.jacinto@tecnico.ulisboa.pt

Group 53

Professor Mário Alexandre Teles de Figueiredo

2023/2024 – 1st Semester, P2

Contents

1 Q1

1.1

(a)

The final accuracy was 0.3422.

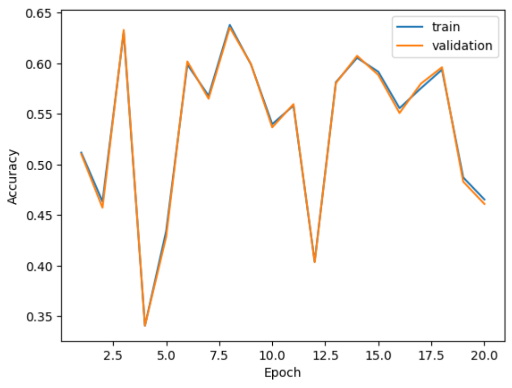
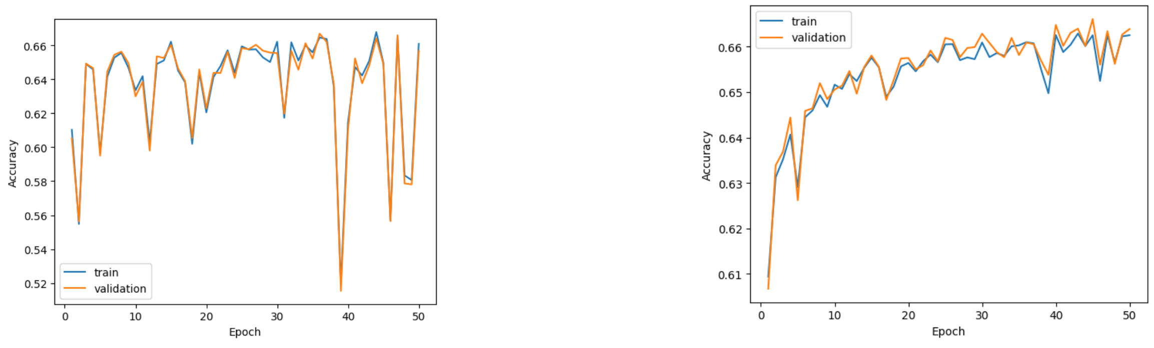


Figure 1: Train and validation accuracy

(b)

The final accuracy for both tests was 0.5784 when the learning rate was 0.01 and 0.5936 when the learning rate was 0.001.



(a) Train and validation accuracy using a learning rate of 0.01

(b) Train and validation accuracy using a learning rate of 0.001

Figure 2: Results using logistic regression

1.2

(a)

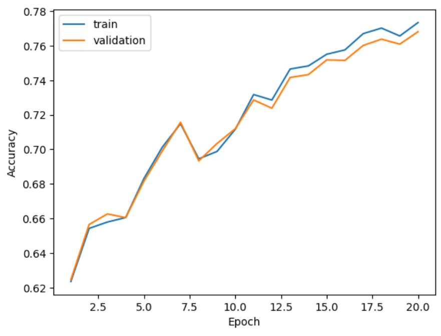
The claim is true if we are comparing a logistic regression model and multi-layer perceptron with ReLu activations. The expressiveness of a model is associated with the capacity to capture the existing relationships within the data points. Usually, an MLP with ReLu activations is more expressive than a logistic regression model that relies on pixel values as features.

Logistic regression is advantageous in terms of training simplicity because its optimization problem is convex and convex optimization problems have a single, globally optimal solution, making it easier to find and converge during the training process. Due to MLP having multiple parameters and non-linear activation functions, the optimization process is more complex and prone to getting stuck in suboptimal solutions.

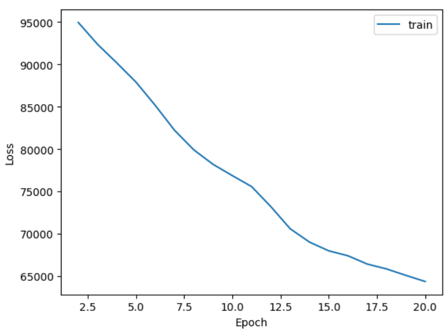
For relatively simple problems where interpretability is crucial and the relationships are mostly linear, logistic regression might be a pratical solution despite its lower expressiveness. In scenarios where the data exhibits complex patterns, an MLP with ReLu activation functions might be a better fit, despite the increased training complexity. That being said, the optimal choice always depends on the specific characteristics and requirements of the problem to be solved.

(b)

The final test accuracy was 0.7202.



(a) Accuracy



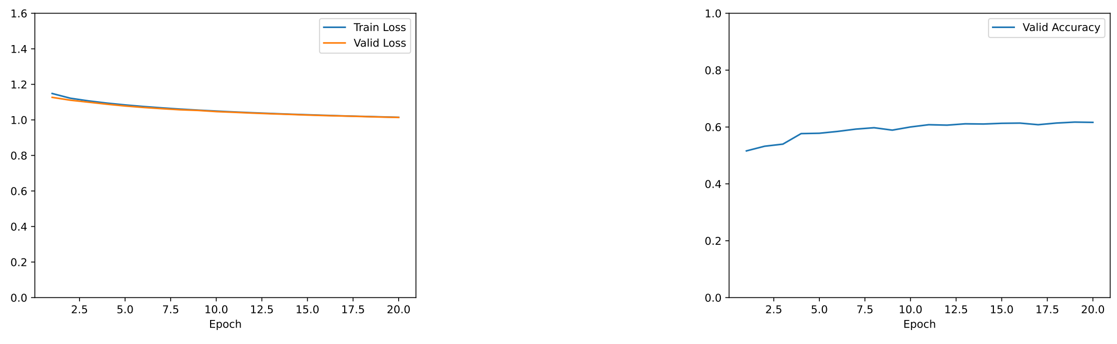
(b) Train loss

Figure 3: Results using a multi-layer perceptron

2 Q2

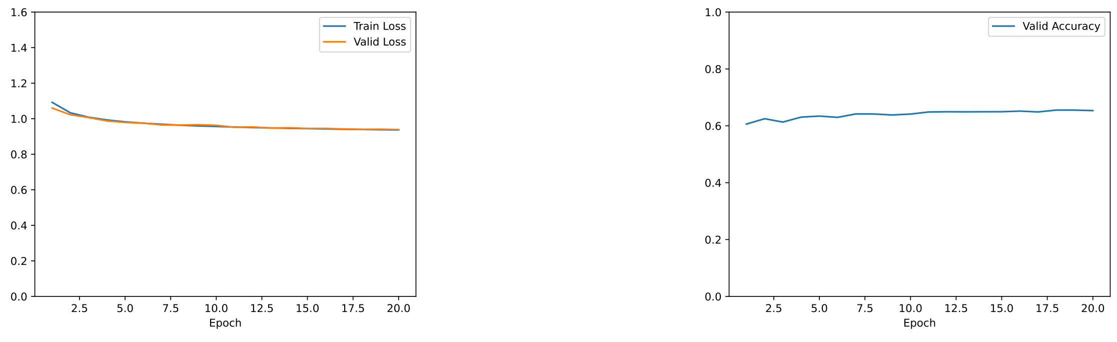
2.1

Using a learning rate of 0.1 the final accuracy was 0.5577, when using a learning rate of 0.01 the final accuracy was 0.620 and when using a learning rate of 0.001 the final accuracy was 0.6503.



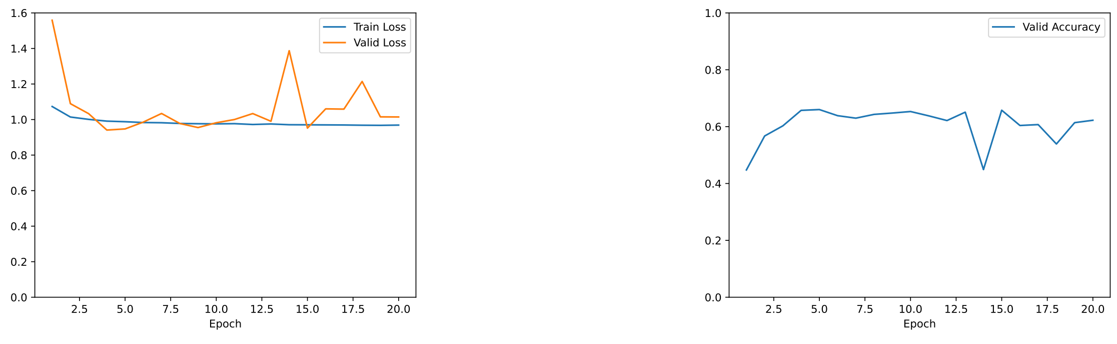
(a) Train and validation loss **(b)** Validation accuracy

Figure 4: Results using logistic regression using a learning rate of 0.001



(a) Train and validation loss **(b)** Validation accuracy

Figure 5: Results using logistic regression using a learning rate of 0.01



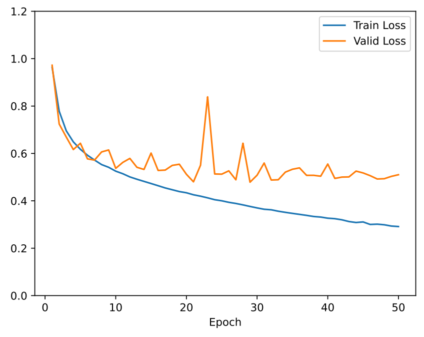
(a) Train and validation loss **(b)** Validation accuracy

Figure 6: Results using logistic regression using a learning rate of 0.1

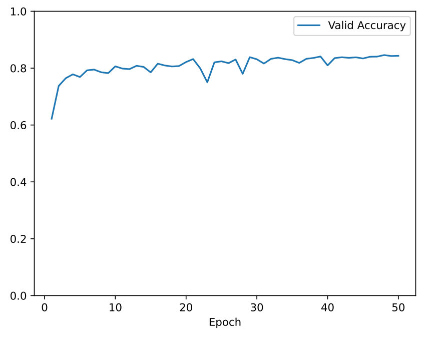
2.2

(a)

Using a batch of 16 we got a final accuracy of 0.7864 with an execution time of 9mins 7s.Using a batch of 1024 we got a final accuracy of 0.7524 with an execution time of 2mins 52s.

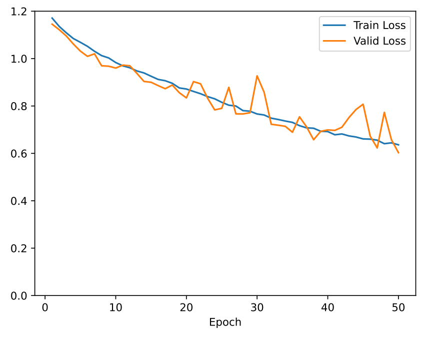


(a) Train and validation loss

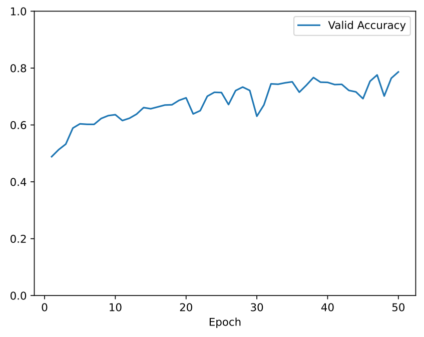


(b) Validation accuracy

Figure 7: Results using a feed-forward neural network using a batch size of 16



(a) Train and validation loss



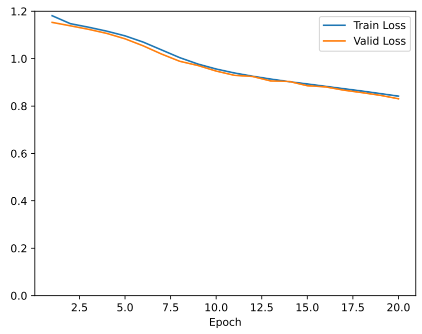
(b) Validation accuracy

Figure 8: Results using a feed-forward neural network using a batch size of 1024

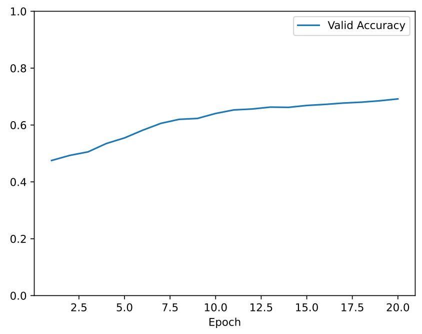
As expected the when using a batch of size 1024 the execution time drops drastically compared to the one with a batch size of 16, we can also note that it helps the model not overfit when using a bigger batch size. On the other hand, the bigger batch size had a lower final accuracy than the smaller batch size.

(b)

When using the default values and changing the learning rate to 0.001 we got a final accuracy of 0.7089 and an execution time of $3min46s$.



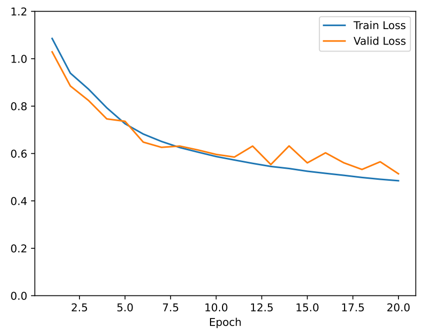
(a) Train and validation loss



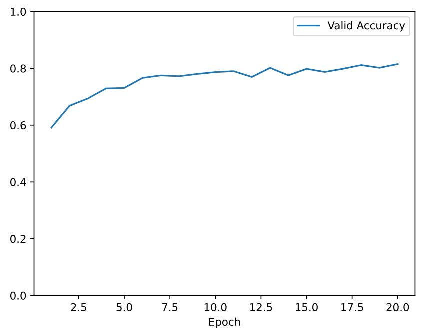
(b) Validation accuracy

Figure 9: Results using a feed-forward neural network using a learning rate of 0.001

When using the default values and changing learning rate to 0.01 we got a final accuracy of 0.7500 and an execution time of $3min44s$.



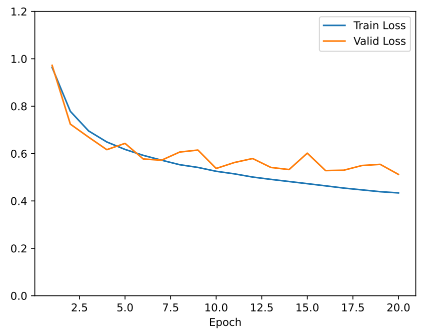
(a) Train and validation loss



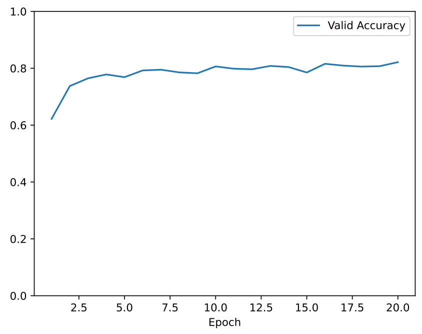
(b) Validation accuracy

Figure 10: Results using a feed-forward neural network using a learning rate of 0.01

When using the default values and changing `learning_rate` to 0.1 we got a final accuracy of 0.7675 and an execution time of $3min41s$.



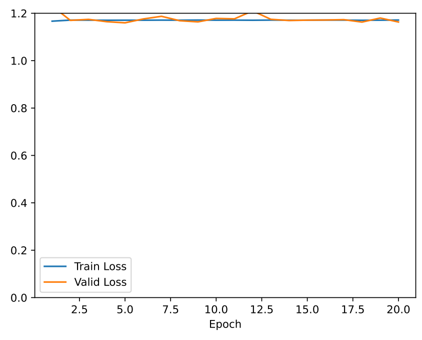
(a) Train and validation loss



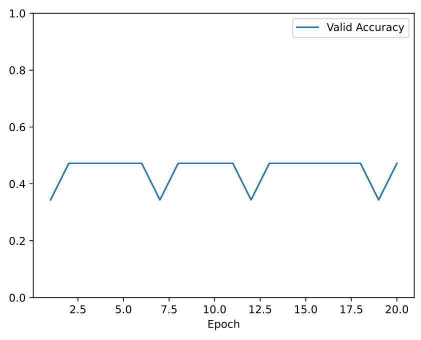
(b) Validation accuracy

Figure 11: Results using a feed-forward neural network using a `learning_rate` of 0.1

When using the default values and changing `learning_rate` to 1.0 we got a final accuracy of 0.4726 and an execution time of $3min35s$.



(a) Train and validation loss



(b) Validation accuracy

Figure 12: Results using a feed-forward neural network using a `learning_rate` of 0.1

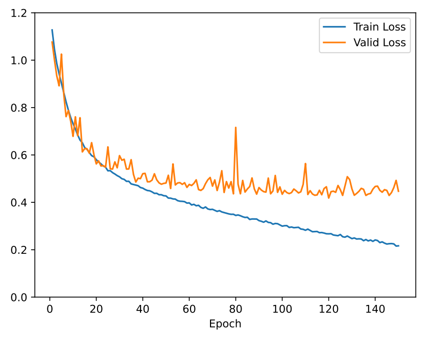
The best accuracy was when we used `learning_rate` = 0.1, we can see that when using that value the training loss converges faster than using the values 0.01 and 0.001. We can also note that when lowering the learning rate the convergence is smoother and less oscilatory.

From figure 12a we can se that the train and validation loss remains almost constant throught the epochs which suggest that the learning rate is too large for stable converge. From figure 12b we can see that the model is not learning from the training data, also due to the fact that the learning rate was too high.

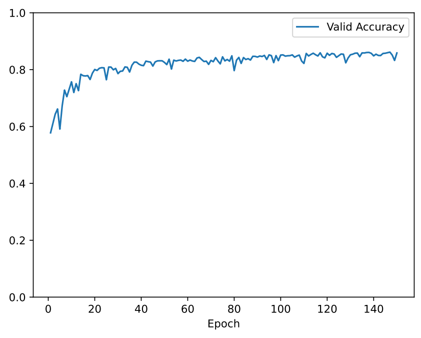
The execution time didn't suffer noticeble changes when changing the learning rate but we can see that the higher the learning rate the lower the execution time.

(c)

For the default values with a batch size of 256 we got a final accuracy of 0.7732 and an execution time of $9min30s$.



(a) Train and validation loss

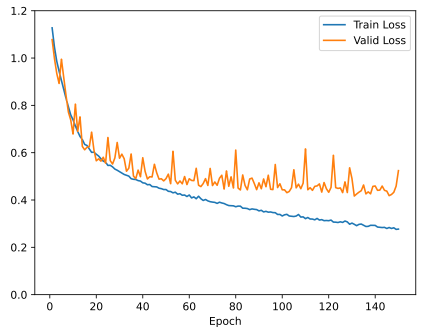


(b) Validation accuracy

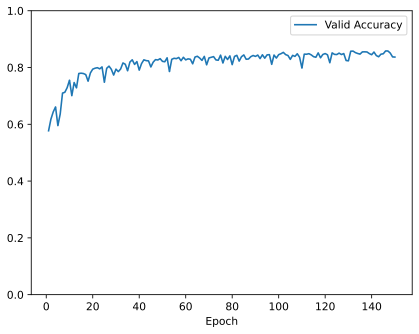
Figure 13: Results using a feed-forward neural network using default values and a batch size of 256

Using a batch size of 256 and 150 epochs we can se the model overfits, while the training loss decreases we notice that the validation loss plateaus at around 50 epochs, creating a large gap between them at the end of all epochs. For the next tests we continued with 150 epochs and a batch size of 256 changing the regularization parameter and the dropout value.

With the same batch size and all default values but setting the regularization parameter to 0.0001 we got a final accuracy of 0.7750 and an execution time of $10min0s$.



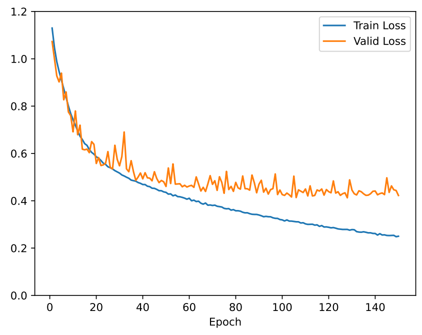
(a) Train and validation loss



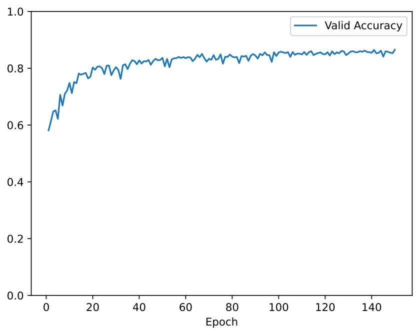
(b) Validation accuracy

Figure 14: Results using a feed-forward neural network using default values and with the L2 regularization parameter set as 0.0001
When using the L2 regularization parameter as 0.0001, we are adding a penalty term to the loss function. We observe a smoother decay in the training loss over time as well as an increased final accuracy of the model.

Sticking to the default values with a batch size of 256 we got a final accuracy of 0.7845 and an execectuion time of $9min43s$.



(a) Train and validation loss



(b) Validation accuracy

Figure 15: Results using a feed-forward neural network using default values and a dropout probability of 0.2

Setting the dropout probability to 0.2, we are randomly setting a fraction of input units to zero during training. This maskes sure the network is more robust, since we train different subsets of neurons on diferent parts of the data. We also note an improved final accuracy from the default values tested before.

3 Q3

3.1

$$f : \{-1, +1\}^D \rightarrow \{-1, +1\}$$

$$f(x) = \begin{cases} +1 \text{ if } \sum_{i=1}^{D} x_i \in [A, B], \\ -1 \text{ otherwise,} \end{cases} \tag{1}$$

(a)

Function 1 cannot be generally computed with a single perceptron. Simple perceptrons are only guaranteed to solve problems that are linearly separable, i.e. classes can be separated by a single hyperplane.

To prove this we take a simple 2-dimensional ($D = 2$) counter-example with $A = -1, B = 1$.

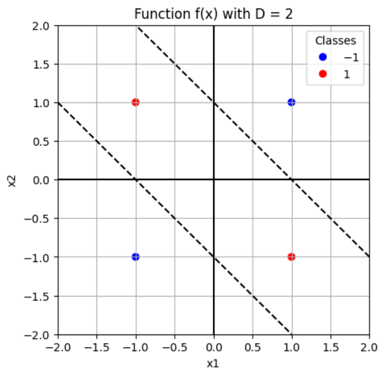


Figure 16: Decision boundaries for function $f(x)$ with $D = 2$

As we can see from figure 16, there is no single hyperplane that is able to clearly separate both classes, at least 2 hyperplanes are required to correctly compute function 1.

(b)

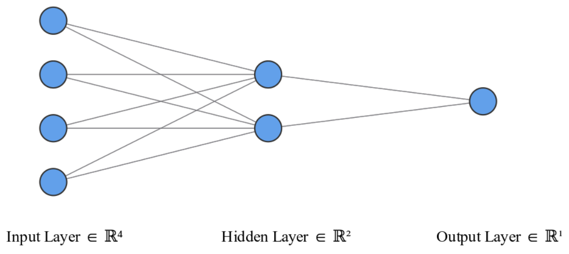


Figure 17: Model of the multilayer perceptron with 2 hidden layers

A multilayer perceptron with 2 hidden layers like the one shown in figure 17, with hard threshold activations can correctly compute function 1. Furthermore, with the correct choice of weights and biases it can also ensure robustness in presence of infinitesimal perturbations of the inputs.

The pre-activation input for each of the hidden units is given by a linear combination of the inputs:

$$z(x) = W^{(1)}x + b^{(1)}$$

The corresponding output of the hidden units, with hard activations is given by the following:

$$h(x) = g(z(x)) = \text{sign}(z(x)) = \text{sign}(W^{(1)}x + b^{(1)}) \tag{2}$$

Finally, the output layer takes as input a linear combination of the outputs of the hidden layer and produces the result of the hard activation:

$$y = f(x) = o(h(x)^T w^{(2)} + b^{(2)}) \tag{3}$$

$$y = sign(w^{(2)T} sign(W^{(1)}x + b^{(1)}) + b^{(2)}) \tag{4}$$

with $W^{(1)} \in \mathbb{R}^{2\times D}$, $b^{(1)} \in \mathbb{R}^2$, $w^{(2)} \in \mathbb{R}^2$, $b^{(2)} \in \mathbb{R}^1$.

Taking equation 4 with $W^{(1)} = \begin{bmatrix} 1 & \dots & 1 \\ 1 & \dots & 1 \end{bmatrix}$ and $b^{(1)} = \begin{bmatrix} -A \\ -B \end{bmatrix}$, $w^{(2)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $b^{(2)} = 1$ we get:

$$y = \text{sign}(h_1(x) - h_2(x) - 1)$$

with

$$h_1(x) = sign(\sum_{i=1}^{D} x_i - A)$$

$$h_2(x) = sign(\sum_{i=1}^{D} x_i - B)$$

This means that when the sum of all the inputs is greater than A and less than B we get $z_1 = 1, z_2 = -1$ and so $y = sign(1-(-1)-1) = 1$, but if any of those conditions is not verified, then $y = sign(1 - 1 - 1) = 0$ or $y = sign(-1 - (-1) - 1 = 0$. Since we want an output of -1 when the conditions aren't met we can use $b^(2) = -1.5$ instead to force the output to -1 in these situations. Furthermore, with $b^{(1)} = \begin{bmatrix} -(A+0.5) \\ -(B+0.5) \end{bmatrix}$, we can guarantee that when the sum of all the inputs is exactly equal to A or B then we also correctly classify the output as $+1$ as intended. This also guarantees robustness in the presence of small disturbances in the input.

(c)

Using rectified linear unit (ReLU) activations instead of hard activations, we need to modify equation 2 and will obtain:

$$h(x) = g(z(x)) = \text{relu}(z(x)) = \text{relu}(W^{(1)}x + b^{(1)}) \tag{5}$$

Then, combining 3 and 5 we obtain:

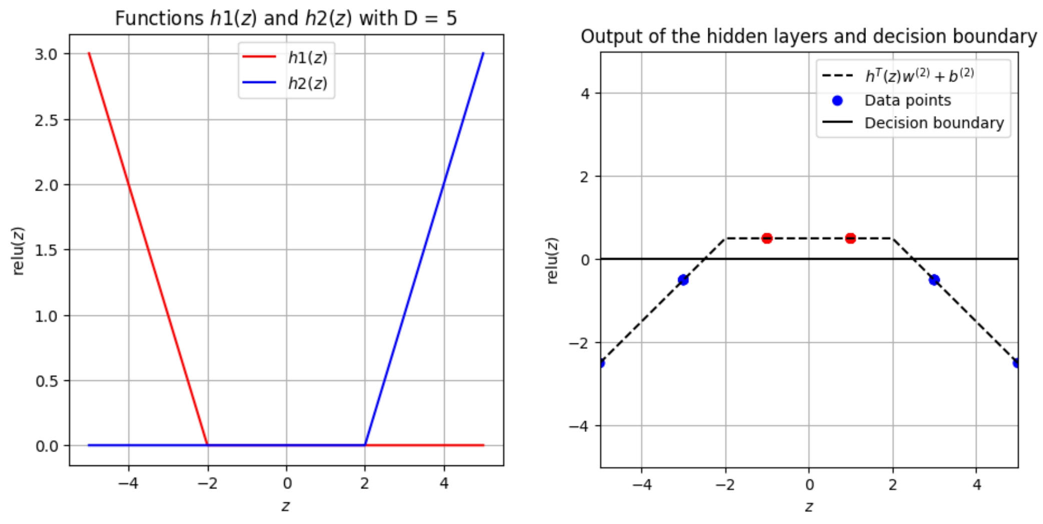
$$y = sign(w^{(2)T} relu(W^{(1)}x + b^{(1)}) + b^{(2)}) \tag{6}$$

In practice, this means that instead of $h(x) \in \{-1, 0, 1\}$ we have $h(x) \in \{0, b, w+b, 2w+b, \ldots, Dw+b\}$, assuming all weights are the same.

If we take $W^{(1)} = \begin{bmatrix} -1 & \ldots & -1 \\ 1 & \ldots & 1 \end{bmatrix}$ and $b^{(1)} = \begin{bmatrix} A \\ -B \end{bmatrix}$ we obtain:

$$h_1(x) = \begin{cases} -\sum_{i=1}^{D} x_i + A \text{ if } -\sum_{i=1}^{D} x_i < A, \\ 0 \text{ otherwise,} \end{cases}$$

$$h_2(x) = \begin{cases} \sum_{i=1}^{D} x_i - B \text{ if } \sum_{i=1}^{D} x_i < B, \\ 0 \text{otherwise} \end{cases}$$



(a) Hidden layer outputs

(b) Final layer inputs and decision boundary

Figure 18: Example with $D = 5$, $A = -2$ and $B = 2$

And with $w^{(2)} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$, $b^{(2)} = 0.5$ we ensure that

$$h(x)^T w^{(2)} + b^{(2)} > 0 \text{ iff } A <= \sum_{i=1}^{D} x_i <= B$$

and therefore function 1 is correctly computed and mantains robustness in the presence of minor disturbances on the input.

4 Contributions

The previous work was separated in the folowing way:

Pedro Jacinto: Exs 1.2, 2.2, 3(a)

Duarte Cerdeira: Exs 1.1, 2.1, 3(b), 3(c)