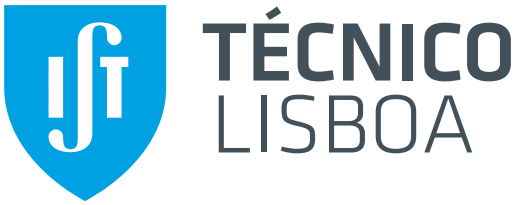


Instituto Superior Técnico

Homework 1

Deep Learning (P2, 2023/24)

Authors	N	Group	Course
Francisco Martins	99068	68	MECD
Wanghao Zhu	95764	68	MECD

Lisboa, December 15, 2023

Table of contents

1 Contributions

Francisco: Did all of question 3, also all of the code for question 1 except 1.2(b) which was done in conjunction with Wanghao. Also wrote up the answer to 1.2(a). Wanghao: Commented on all of the plots, did all of the code for question 2. Also did in conjunction with Francisco the code for question 1.2(b).

2 Answers to questions

Question 1

1.(a)

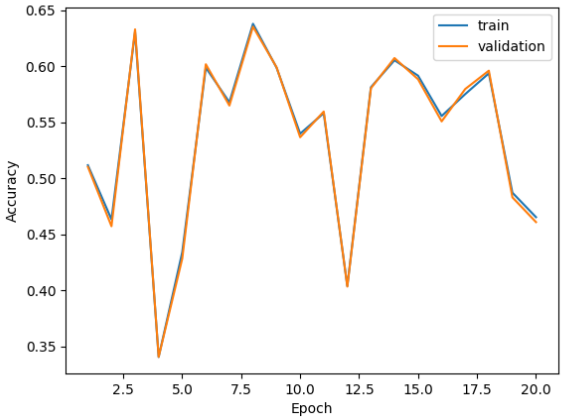
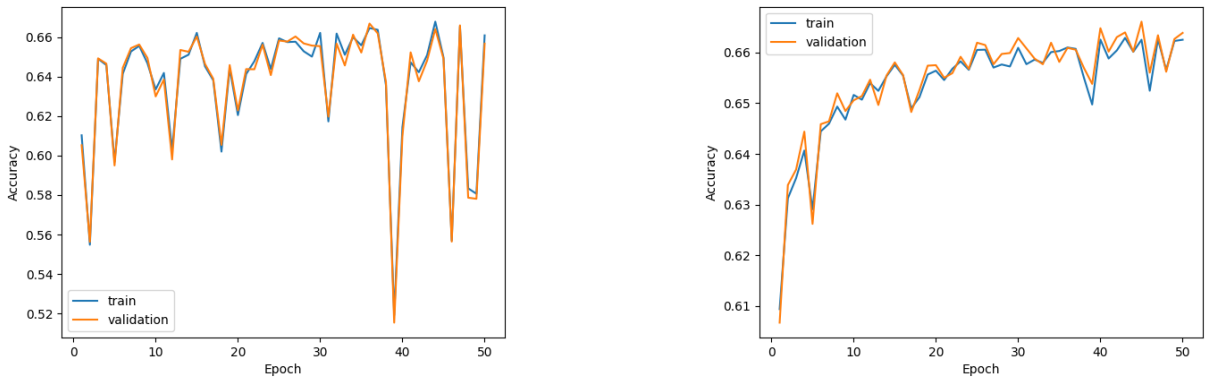


Figure 1: Training and Validation Accuracies per Epoch for the Perceptron Model.

The model achieved a final test accuracy of 34.22%, which is significantly lower than the training and validation accuracies (46.54% and 46.10%), as shown in figure 1, this discrepancy indicates a potential overfitting to the training data or a mismatch between the training/validation set distribution and the test set distribution.

1.(b)



(a) LR=0.01 (b) LR=0.001

Figure 2: Comparative Training and Validation Accuracies for Different Learning Rates

The findings suggest that a learning rate of 0.001 for logistic regression offers a better balance between learning stability and model performance, exhibited a more stable and consistent increase in accuracies over epochs, as it achieved a higher test accuracy (57.84% vs 59.36%) with less variance in accuracy over epochs.

2.(a)

There are two things mentioned here that will discussed - expressiveness and training complexity.

Expressiveness:

Logistic Regression: This model is a linear classifier. It is using pixel values as features, and attempts to separate classes using a linear decision boundary. Its expressiveness is limited to representing linear relationships in the data. It cannot model more complex patterns between the pixel values such as correlations between pixels.

Multi-Layer Perceptron with ReLU Activations: The ReLU activation function introduces non-linearity into the network which, combined with multiple layers, allows the MLP to learn more complex patterns and correlations between the pixel values that logistic regression can't. Essentially, it is capable of learning an internal representation that simplifies the separation of the dataset classes in the final layer.

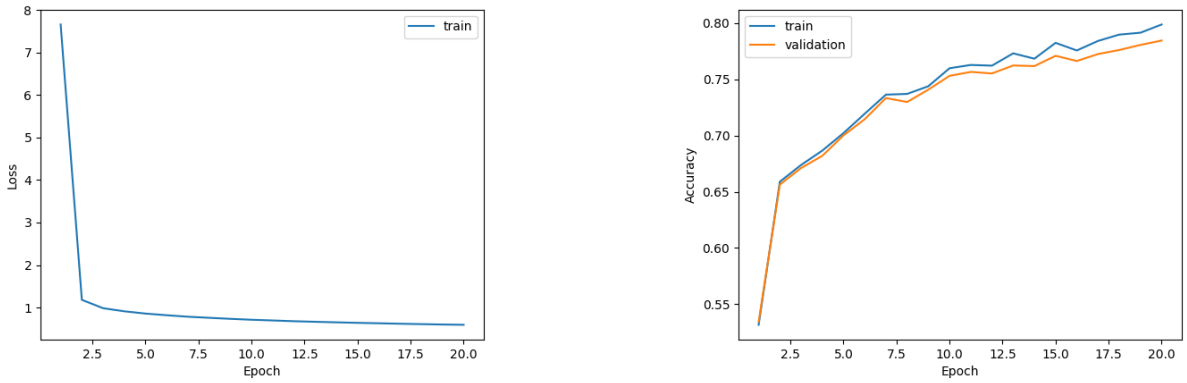
Training Complexity:

Logistic Regression: The optimization problem in logistic regression is convex, meaning there is a single global minimum. Gradient descent methods are guaranteed to converge to this global minimum. This makes training logistic regression models relatively straightforward and computationally less intensive.

Multi-Layer Perceptron: Training MLPs is more complex. The presence of multiple layers and non-linear activations turns the optimization problem into a non-convex one. The optimization landscape is a lot more complicated. This means there can be multiple local minima, and gradient descent methods might not necessarily converge to the global minimum. Training MLPs requires careful tuning of parameters (like learning rates, initialization, etc.) and is more computationally intensive.

2.(b)

The model achieved a final test accuracy of 74.86%.



(a) Training loss

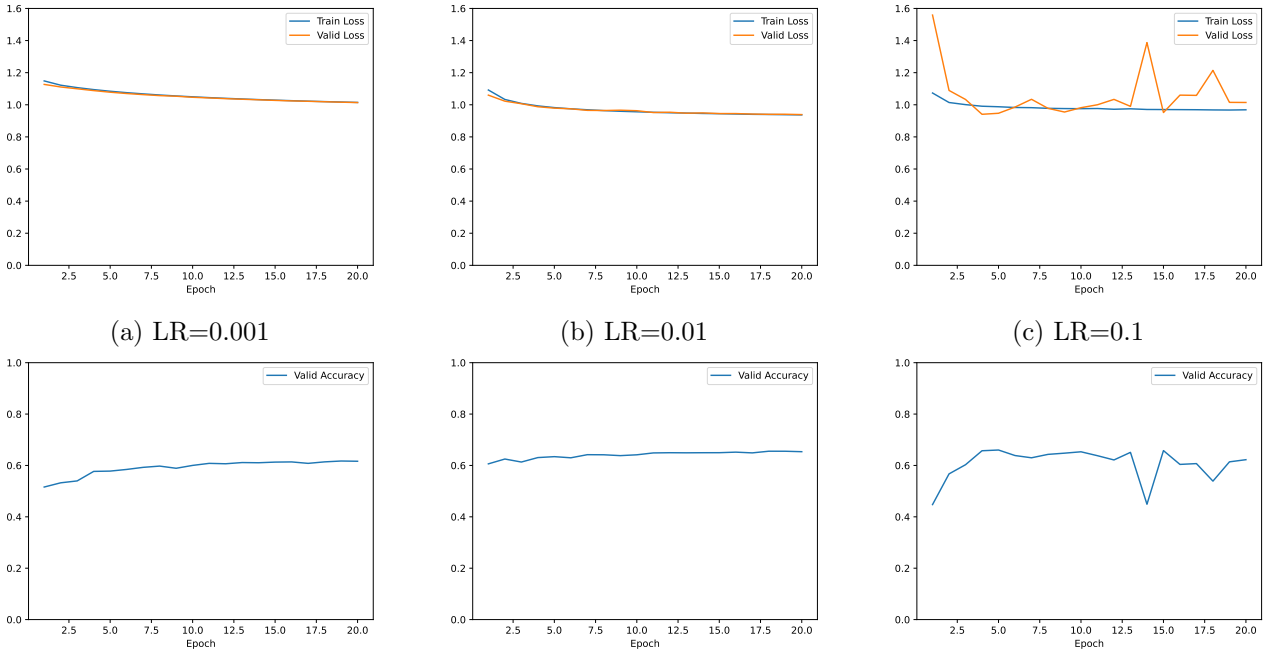
(b) Training and validation accuracies

Figure 3: Training loss and training and validation accuracies of the multi-layer perceptron across 20 epochs

Question 2

1.

We trained three logistic regression models with distinct learning rates to examine their impact on the performance of medical image classification. The learning rates tested were 0.001, 0.01, and 0.1, with each model trained for 20 epochs using a batch size of 16.



(a) LR=0.001 (b) LR=0.01 (c) LR=0.1

(d) LR=0.001 (e) LR=0.01 (f) LR=0.1

Figure 4: Comparative performance of logistic regression models with different learning rates. The top row shows the training loss and the bottom row the validation accuracy across 20 epochs.

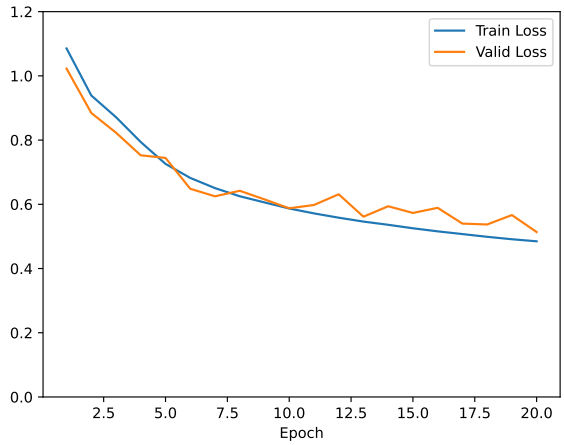
Table 1: Final test accuracy for each learning rate.

Learning Rate	0.001	0.01	0.1
Final Validation Accuracy	61.63%	65.35%	62.24%
Final Test Accuracy	65.03%	62%	55.77%

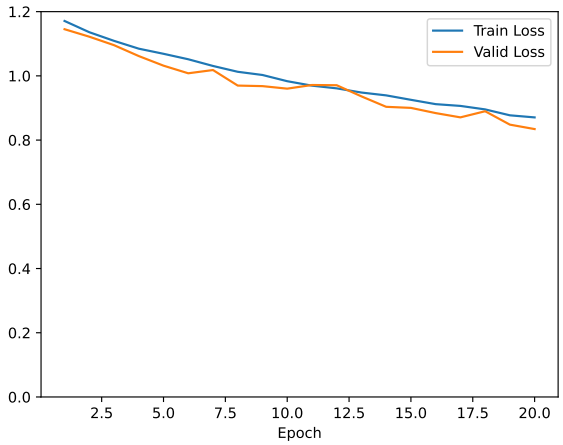
The model with a learning rate of 0.001 and 0.01 exhibited consistent improvement throughout the training epochs, reflecting steady learning and ended with finals validation accuracies of 61.63% and 65.35%. A learning rate of 0.01 had the highest one. Whichs final test accuracy 62%. This suggests that the smaller updates in model parameters allowed for a more nuanced approach to finding a minimum in the loss landscape.

The model using a learning rate of 0.1 demonstrated the most significant variations in validation accuracy. This is often characteristic of larger learning rates that can cause the optimizer to overshoot minima. This model achieved a final test accuracy of 55.77%, the lowest among the three.

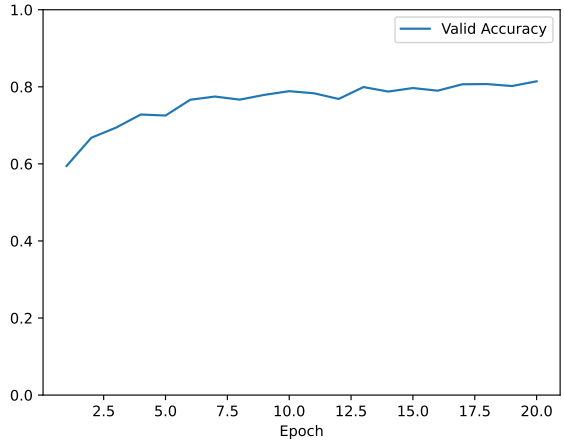
2.(a)



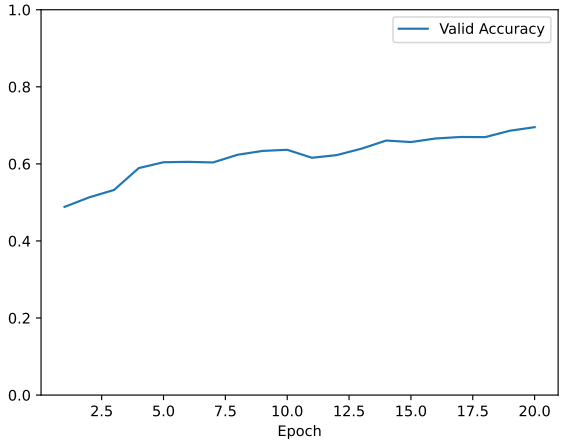
(a) Batch sizes=16



(b) Batch sizes=1024



(c) Batch sizes=16



(d) Batch sizes=1024

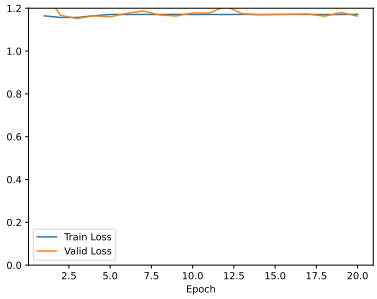
Figure 5: Comparative performance of MLP models with different batch sizes. The top row shows the training loss and the bottom row the validation accuracy across 20 epochs.

The highest test accuracy, 75.05%, was achieved with the model using a batch size of 16. This model displayed quicker convergence, which can be attributed to the increased randomness and variability introduced by smaller batch sizes. When updates are based on smaller data subsets, they tend to be more responsive to individual data points, as opposed to larger batch sizes that promote weight adjustments that are more generalized and stable but less responsive to individual variations.

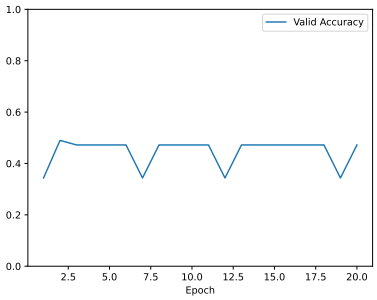
On the other hand, training with a batch size of 1024 was significantly faster compared to a batch size of 16. This efficiency stems from processing larger data sets simultaneously, reducing the need for frequent data reloading during each training iteration. However, it's important to note that larger batch sizes demand more memory.

The time taken for each approach was 3 minute and 34.193 seconds for the batch size of 16, and 45.569 seconds for the batch size of 1024, demonstrating the trade-offs between training speed and the granularity of learning.

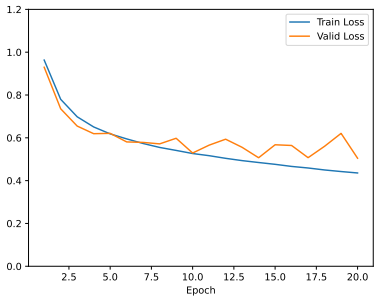
2.(b)



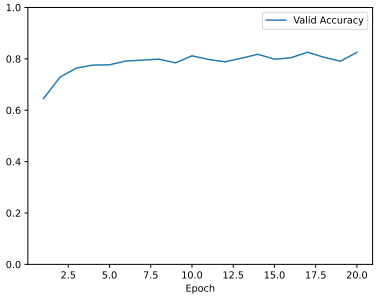
(a) Training Loss (LR=1.0)



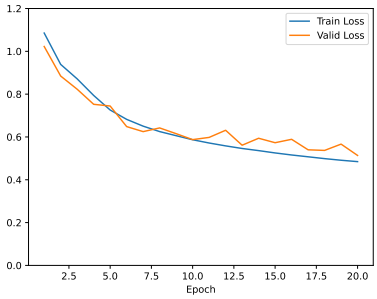
(b) Validation Accuracy (LR=1.0)



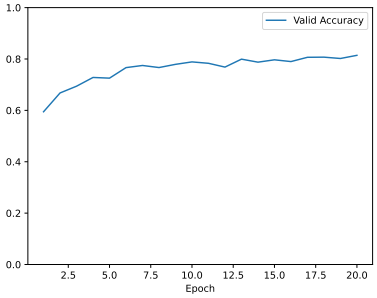
(c) Training Loss (LR=0.1)



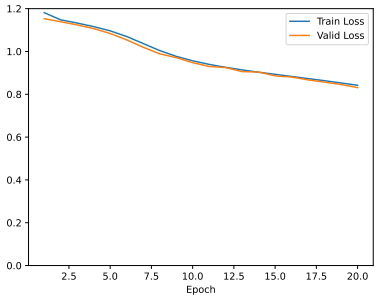
(d) Validation Accuracy (LR=0.1)



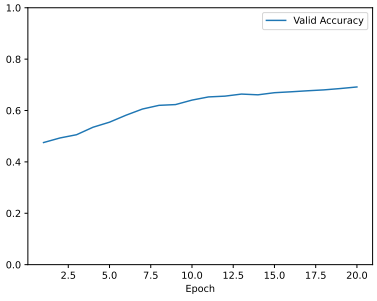
(e) Training Loss (LR=0.01)



(f) Validation Accuracy (LR=0.01)



(g) Training Loss (LR=0.001)



(h) Validation Accuracy (LR=0.001)

Figure 6: Comparative performance of the MLP model with different learning rates. Training loss and validation accuracy are shown for each learning rate over 20 epochs.

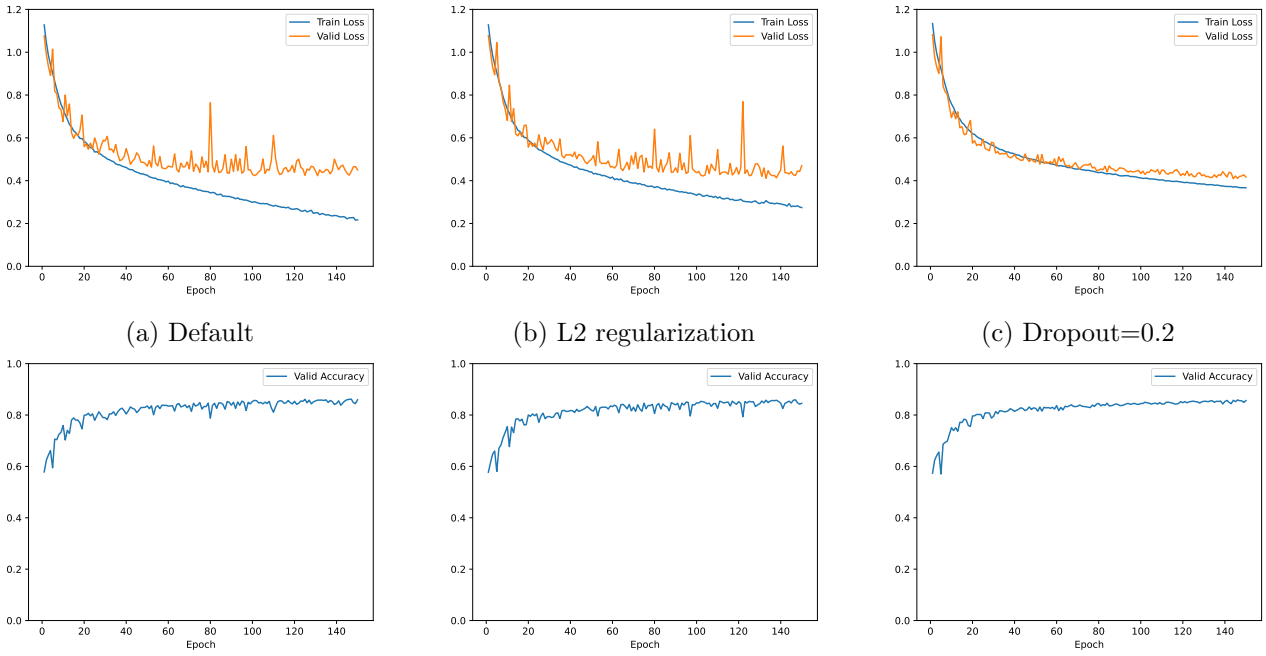
Table 2: Final test and validation accuracy for each learning rate.

Learning Rate	1	0.1	0.01	0.001
Final Test Accuracy	47.26%	75.05%	76.73%	71.46%

The best and the worst configurations in terms of validation accuracy was achieved with a learning rate of 0.01 and 1.

The differences in performance between the configurations can be attributed to how the learning rate affects the model's ability to converge to a good set of weights. A learning rate of 1 was too high, causing unstable training and poor model performance. This is evident from the higher training and validation losses and lower accuracy. Conversely, a learning rate of 0.01 offered a more controlled and steady convergence, leading to better training dynamics and higher accuracy on both validation and test sets. The performance with learning rates 0.1 and 0.001 fell in between these extremes, illustrating the sensitivity of model training to the choice of learning rate.

2.(c)



(a) Default (b) L2 regularization (c) Dropout=0.2

(d) Default (e) L2 regularization (f) Dropout=0.2

Figure 7: Comparative performance of models with different regularization techniques. The top row shows the training loss and the bottom row the validation accuracy across 150 epochs.

The first model, run with a batch size of 256 for 150 epochs, showed good convergence without significant signs of overfitting. This is evidenced by stable training and validation loss trends and a consistent accuracy improvement over epochs. The best and worst configuration (in terms of validation accuracy) are the model with dropout and the model with L2 regularization.

Table 3: Comparison of test accuracy for different regularization and dropout techniques.

	Default	L2	Dropout
Test Accuracy	76.73%	76.94%	78.45%

The highest test accuracy recorded was 78.45%, achieved by the model with dropout. This indicates that dropout was more effective in enhancing the model's ability to generalize on unseen data, compared to L2 regularization in this specific scenario.

Dropout as a regularization technique seemed to be more effective for this particular model and dataset. By randomly deactivating neurons during training, it prevented the network from overfitting and promoted the development of more robust features.

L2 Regularization aims to reduce model complexity by penalizing large weights. However, in this case, it was less effective than dropout, which might be due to the specific characteristics of the dataset or the model architecture.

Question 3

1. (a)

One such function is the logical XOR which in this formulation corresponds to the values D=2, A=B=0, i.e.:

$$XOR(x) = \begin{cases} 1 & if \ \sum_{i=1}^{2} x_i \in [0,0] \\ -1 & otherwise, \end{cases}$$

When both components have opposite signs the function outputs 1, otherwise it outputs -1.

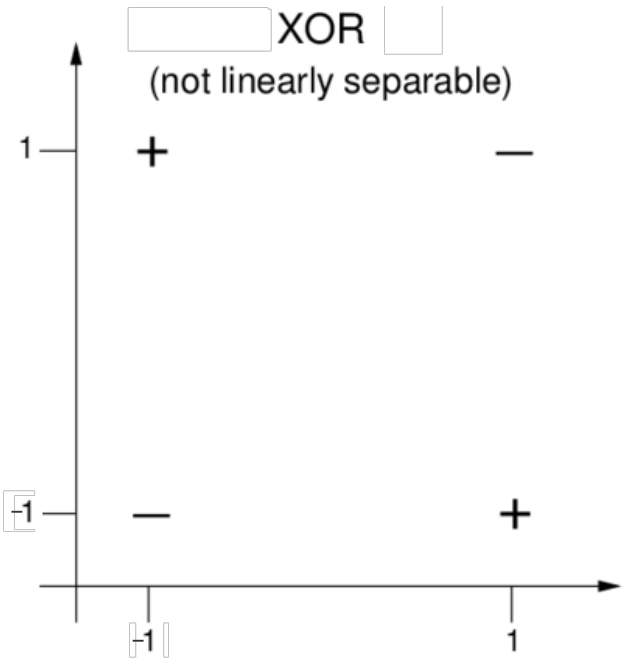


Figure 8: The XOR function is not linearly separable

1. (b)

Given D, A and B we want to specify a multi-layer perceptron that computes the corresponding Boolean function of D variables, $f\colon \{-1,1\}^D \to \{-1,1\}$, defined as:

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

There are many different Boolean functions and for each the weights and biases of the MLP will be different. For the general case, given values of D, A and B we can always specify an MLP that computes the corresponding Boolean function, henceforth referred to as $Boolean_{A,B,D}$.

First, let's define the following quantities that will be used in the multi layer perceptron specification. Given A and B, α and β will be this:

$$\alpha = \begin{cases} A-1 & if \quad A \equiv D \pmod 2 \\ A & otherwise, \end{cases}, \quad \beta = \begin{cases} B+1 & if \quad B \equiv D \pmod 2 \\ B & otherwise, \end{cases}$$

The reason for using α and β in the MLP will become apparent later.

For a given $x \in \{-1,1\}^D$, let $\tilde{x} = x + tv$ such that $|\sum_{i=1}^{D} tv_i| < 1$, so this is a small perturbation of the input. Then $Boolean_{A,B,D}(x) = 1$ if and only if (see proof in the Appendix):

$$\begin{cases} \sum_{i=1}^{D} \tilde{x}_i \geq \alpha \iff \sum_{i=1}^{D} \tilde{x}_i - \alpha \geq 0 \iff sign(w_1^T \tilde{x} - \alpha) = 1 \\ \sum_{i=1}^{D} \tilde{x}_i \leq \beta \iff \beta - \sum_{i=1}^{D} \tilde{x}_i \geq 0 \iff sign(w_2^T \tilde{x} + \beta) = 1 \end{cases} \tag{1}$$

where,

$$\mathbb{R}^{D\times 1} \ni w_1^{(1)} = [1 \quad 1 \quad \cdots \quad 1]^T, \quad w_2^{(1)} = -w_1^{(1)}$$

and

$$sign\colon \mathbb{R} \to \{-1,1\}$$

is defined as:

$$sign(x) = \begin{cases} 1 & if\ x \geq 0 \\ -1 & otherwise, \end{cases}$$

Because of the "if and only if" above, this also means that if either of these conditions doesn't hold for x then $Boolean_{A,B,D}(x) \neq 1$, so $Boolean_{A,B,D}(x) = -1$.

Based on this we specify for the first layer the following weights and biases:

$$W^{(1)} = \begin{bmatrix} w_1^{(1)^T} \\ w_2^{(1)^T} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ -1 & -1 & \cdots & -1 \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} -\alpha \\ \beta \end{bmatrix}$$

The hidden layer's output is $h^{(1)} = sign(W^{(1)}\tilde{x} + b^{(1)}) \in \{-1,1\}^{2\times 1}$.

So, h is an indicator variable: its first component conveys whether $\sum_{i=1}^{D} \tilde{x}_i \geq \alpha$ and its second component conveys whether $\sum_{i=1}^{D} \tilde{x}_i \leq \beta$.

Given what we saw in (1), we want the final output as a function of the hidden layer's output to be:

$$\hat{y} = \begin{cases} 1 & if \quad h^{(1)} = [1 \quad 1]^T \\ -1 & otherwise, \end{cases}$$

Since then both conditions in (1) will bet met.

So now, the second layer is just an AND gate, which is a linearly separable function. It is very simple to implement with a single layer perceptron. Notice in the Boolean function's formulation,

we were kind of using an AND gate. This together with using a single hidden layer with *two* hidden units hinted at this sort of final layer.

Here are the second layer weights and bias:

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

The only way for $W^{(2)}h^{(1)} + b^{(2)}$ to be positive is if $h_1^{(1)} = 1$ and $h_2^{(1)} = 1$. Which is what we want.

One interesting thing to notice is (4) (see Appendix) exactly implies that the function h resulting from the MLP satisfies $\lim_{t\to 0} h(x + tv) = h(x) = Boolean_{A,B,D}(x)$. Since h only cares about the value of the summation of its input, we can think just about the value of $\sum_{i=1}^{D} x_i + tv_i$.

$$\lim_{t\to 0} \sum_{i=1}^{D} x_i + tv_i = \sum_{i=1}^{D} x_i$$

This means:

$$\exists_\delta : \quad | \, t \, | < \delta \Rightarrow | \sum_{i=1}^{D} x_i + tv_i - \sum_{i=1}^{D} x_i \, | \quad = \quad | \sum_{i=1}^{D} tv_i \, | < 1$$

.

This together with eq. (4) and the way we constructed the MLP proves that the function is robust to perturbations.

Let's give some intuition to how using α and β instead of the original A and B grants robustness to small perturbations of the inputs to the MLP. We want the range boundaries to not be on a possible value for the summation. So we give ourselves more leeway by increasing the range, by changing the range limits A and/or B. But only when doing that won't make us mispredict other xs. In the end, this amounts to decreasing/increasing A or B by 1 (respectively) only when they are equal to a possible value for the summation. The change always increases the range, meaning $\alpha \leq A$ and $\beta \geq B$. For more detail look at the proof of (1) in the appendix. Let's look at examples:

- For $D = 2$, the values the summation can take are: $\{-2, 0, 2\}$. Notice they are all even values. Let's take, for example, $A = 0$, $B = 1$ and $x = [0 \quad 0]^T$. Let $v = [0 \quad -1]^T$, $t = 10^{-4}$ and $\tilde{x} = x + tv$. Then $-10^{-4} = \sum_{i=1}^{D} \tilde{x}_i < A < B$. So, if we use A directly in the MLP instead of α, the value of $h_1^{(1)} = sign(\sum_{i=1}^{D} x_i - A)$ is 1 if the input to the MLP is x, and -1 if it's $\tilde{x}$. Meanwhile, $h_2^{(1)}$ is 1 for both x and $\tilde{x}$. This means $1 = MLP(x) \neq MLP(\tilde{x}) = -1$. We conclude that this MLP function is not robust to small perturbations. However, if we use $\alpha = A - 1 = -1$ then $\sum_{i=1}^{D} \tilde{x}_i > \alpha$ which means $h_1^{(1)} = 1$ then, $1 = MLP(\tilde{x}) = MLP(x) = Boolean_{A,B,D}(x)$. In fact, as long as $| \sum_{i=1}^{D} tv_i | < 1$, the MLP output will always be robust to the perturbation. Since we are expanding our allowed range we could be mispredicting xs as 1 when really $Boolean_{A,B,D}(x) = -1$. However, α is such that $\forall x \sum_{i=1}^{D} x_i \neq -1 = \alpha$ so we don't catch any new possible values for the summation of x. B is 1 so it's already in the sweet spot between the possible values attained by the summation and so $\beta = B$. So α and β will always be in the integer spaces between the possible values for the summation which gives us robustness to perturbations without skewing the predictions.

- For $D = 3$, the values the summation can take are: $\{-3, -1, 1, 3\}$. Notice they are all odd values. In that case, if A is odd we subtract one from it, increasing our range. That way, it's not on a value attained by the summation, which makes our function robust to perturbations. Similarly, for B, if it is odd, we add one to it, increasing the range, but not incurring any misclassifications as a result.

Finally, notice the possible values for the summation are odd if D is odd and even if D is even (proof in the Appendix). This concludes our explanation for the intuition behind α and β and the need to use them instead of the original A and B.

Next, for a recap of all of these ideas and clarity, we provide a function in python that returns a function corresponding to the MLP counterpart to the corresponding Boolean function defined by the given A, B and D.

```
def mlp(A, B, D):
    def mlp_aux(x, A, B, D):
        alpha = A
        beta = B
        if A % 2 == D % 2:
            alpha -= 1
        if B % 2 == D % 2:
            beta += 1

        W1 = np.vstack((np.ones(D), -1 * np.ones(D)))
        b1 = np.array([-alpha, beta]).reshape(2,1)
        W2 = np.ones((1, 2))
        b2 = -1

        z1 = W1 @ x + b1
        h1 = sign(z1)
        z2 = W2 @ h1 + b2
        out = sign(z2)
        return out[0,0]
    return lambda x: mlp_aux(
        np.array(x).reshape((D,1)), A, B, D
        )
```

Example usage:

```
h = mlp(-2,1,2)
x1 = [1,-1]
x2 = [-1,-1]
print(h(x1), h(x2))
```

1. (c)

For this question we will again use the previously discussed values - α and β - for robustness to small perturbations and the additional perk of not being on the values for the summation, meaning:

$$\forall_x \quad \sum_{i=1}^{D} \tilde{x}_i \neq \alpha \wedge \sum_{i=1}^{D} \tilde{x}_i \neq \beta \tag{2}$$

The first layer will be exactly the same as before:

$$W^{(1)} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ -1 & -1 & \cdots & -1 \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} -\alpha \\ \beta \end{bmatrix}$$

Let $z^{(1)} = W^{(1)}\tilde{x} + b^{(1)}$. The hidden layer's output is $h^{(1)} = relu(z^{(1)})$.

Notice that:

$$h_1^{(1)} = 0 \iff relu(\sum_{i=1}^{D} \tilde{x}_i - \alpha) = 0 \iff \sum_{i=1}^{D} \tilde{x}_i \leq \alpha$$

$$h_2^{(1)} = 0 \iff relu(-\sum_{i=1}^{D} \tilde{x}_i + \beta) = 0 \iff \sum_{i=1}^{D} \tilde{x}_i \geq \beta$$

But because of (2):

$$h_1^{(1)} = 0 \iff \sum_{i=1}^{D} \tilde{x}_i < \alpha$$

$$h_2^{(1)} = 0 \iff \sum_{i=1}^{D} \tilde{x}_i > \beta$$

These two results mean we want the final output as a function of the hidden layer output to be:

$$\hat{y} = \begin{cases} -1 & if \quad h_1^{(1)} = 0 \vee h_2^{(1)} = 0 \\ 1 & otherwise, \end{cases}$$

It will be simpler to first specify the second layer, and then explain why it is like that.

$$W^{(2)} = \begin{bmatrix} -1 & -1 \end{bmatrix}, \quad b^{(2)} = \beta - \alpha$$

Notice that $W^{(2)}h^{(1)}$ is minus the sum of the components of $h^{(1)}$. Let's look at what this equals for different cases:

- Case 1: $\sum_{i=1}^{D} \tilde{x}_i > \alpha$ and $\sum_{i=1}^{D} \tilde{x}_i < \beta$, which means:

$$h_1^{(1)} = relu(\sum_{i=1}^{D} \tilde{x}_i - \alpha) = \sum_{i=1}^{D} \tilde{x}_i - \alpha$$

$$h_2^{(1)} = relu(\beta - \sum_{i=1}^{D} \tilde{x}_i) = \beta - \sum_{i=1}^{D} \tilde{x}_i$$

  which in turn means

$$-h_1^{(1)} - h_2^{(1)} = \alpha - \beta$$

- Case 2: $\sum_{i=1}^{D} \tilde{x}_i < \alpha$ and $\sum_{i=1}^{D} \tilde{x}_i < \beta$, which means:

$$h_1^{(1)} = relu(\sum_{i=1}^{D} \tilde{x}_i - \alpha) = 0$$

$$h_2^{(1)} = relu(\beta - \sum_{i=1}^{D} \tilde{x}_i) = \beta - \sum_{i=1}^{D} \tilde{x}_i$$

  which in turn means

$$-h_1^{(1)} - h_2^{(1)} = \sum_{i=1}^{D} \tilde{x}_i - \beta < \alpha - \beta$$

- Case 3: $\sum_{i=1}^{D} \tilde{x}_i > \alpha$ and $\sum_{i=1}^{D} \tilde{x}_i > \beta$, which means:

$$h_1^{(1)} = relu(\sum_{i=1}^{D} \tilde{x}_i - \alpha) = sum_{i=1}^{D} \tilde{x}_i - \alpha$$

$$h_2^{(1)} = relu(\beta - \sum_{i=1}^{D} \tilde{x}_i) = 0$$

which in turn means

$$-h_1^{(1)} - h_2^{(1)} = \alpha - \sum_{i=1}^{D} \tilde{x}_i < \alpha - \beta$$

- Case 4: $\sum_{i=1}^{D} \tilde{x}_i > \alpha$ and $\sum_{i=1}^{D} \tilde{x}_i < \beta$, which is not possible since $\alpha < \beta$

Finally, we can summarize all of this by saying if $h_1^{(1)} = 0 \vee h_2^{(1)} = 0$ then $-h_1^{(1)} - h_2^{(1)} < \alpha - \beta \iff W^{(2)}h^{(1)} + b^{(2)} < 0 \iff sign(W^{(2)}h^{(1)} + b^{(2)}) = -1$ and if $h_1^{(1)} \neq 0 \wedge h_2^{(1)} \neq 0$ then $-h_1^{(1)} - h_2^{(1)} = \alpha - \beta \iff W^{(2)}h^{(1)} + b^{(2)} = 0 \iff sign(W^{(2)}h^{(1)} + b^{(2)}) = 1$. So that as desired:

$$\hat{y} = \begin{cases} -1 & if \quad h_1^{(1)} = 0 \vee h_2^{(1)} = 0 \\ 1 & otherwise, \end{cases}$$

Finally, notice the same analysis that was done in (b) can be done here to show our MLP with relu is robust to small perturbations as defined in the homework since the first layer's weights are the same.

```
def mlp_relu(A, B, D):
    def mlp_relu_aux(x, A, B, D):
        alpha = A
        beta = B
        if A % 2 == D % 2:
            alpha -= 1
        if B % 2 == D % 2:
            beta += 1

        W1 = np.vstack((np.ones(D), -np.ones(D)))
        b1 = np.array([-alpha, beta]).reshape(2,1)
        W2 = -1 * np.ones((1, 2))
        b2 = beta - alpha

        z1 = W1 @ x + b1
        h1 = np.maximum(0, z1)
        z2 = W2 @ h1 + b2
        out = sign(z2)
        return out[0,0]
    return lambda x: mlp_relu_aux(
        np.array(x).reshape((D,1)), A, B, D
        )
```

Appendix

Proof that D has the same parity as $\sum_{i=1}^{D} x_i$

Let $\{-1, 1\}^D \ni x^{1\ldots1} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}$ then $\sum_{i=1}^{D} x_i^{1\ldots1} = D$. Finally, notice each x_i can only switch from 1 to -1 and vice-versa, for an oscillation in the summation of $\pm$ 2.

Summing or subtracting 2 from an integer doesn't change its parity so this means for any x that $\sum_{i=1}^{D} x_i$ has the same parity as the summation of $x^{1\ldots1}$. So it has the same parity as D. In more compact notation:

$$\forall x, \qquad \sum_{i=1}^{D} x_i \equiv D \pmod 2 \tag{3}$$

Proof of (1)

For a given $x \in \{-1, 1\}^D$, let $\tilde{x} = x + tv$ such that $|\sum_{i=1}^{D} tv_i| < 1$, so this is a small perturbation of the input. Then we want to prove that:

$$Boolean_{A,B,D}(x) = 1 \iff \sum_{i=1}^{D} \tilde{x}_i \geq \alpha \wedge \sum_{i=1}^{D} \tilde{x}_i \leq \beta \tag{4}$$

Let's first prove (4) in the direction ($\Rightarrow$):
Assume $Boolean_{A,B,D}(x) = 1$, then:

$$\sum_{i=1}^{D} x_i \geq A \wedge \sum_{i=1}^{D} x_i \leq B$$

$$\sum_{i=1}^{D} \tilde{x}_i = \sum_{i=1}^{D} x_i + \sum_{i=1}^{D} tv_i > A - 1 = \alpha$$

$$\sum_{i=1}^{D} \tilde{x}_i = \sum_{i=1}^{D} x_i + \sum_{i=1}^{D} tv_i < B + 1 = \beta$$

■

Let's now prove (4) in the direction ($\Leftarrow$). We can do a proof by contrapositive, meaning we switch from "if A, then B" to "if not B, then not A". So proving it in the direction ($\Leftarrow$) is the same as proving:

$$Boolean_{A,B,D}(x) = -1 \Rightarrow \sum_{i=1}^{D} \tilde{x}_i < \alpha \vee \sum_{i=1}^{D} \tilde{x}_i > \beta$$

Assume $Boolean_{A,B,D}(x) = -1$, this implies:

$$\sum_{i=1}^{D} x_i < A \vee \sum_{i=1}^{D} x_i > B$$

Let's break it down into cases:
Case 1: Both occur simultaneously
They can't both occur simultaneously as $A < B$.
Case 2: $\sum_{i=1}^{D} x_i < A \leq B$:
Notice $\sum_{i=1}^{D} x_i, A \in \mathbb{N}$ so $\sum_{i=1}^{D} x_i \leq A - 2 \vee \sum_{i=1}^{D} x_i = A - 1$
If $\sum_{i=1}^{D} x_i \leq A - 2$, then:

$$\sum_{i=1}^{D} \tilde{x}_i = \sum_{i=1}^{D} x_i + \sum_{i=1}^{D} tv_i < (A - 2) + 1 = A - 1 = \alpha$$

If $\sum_{i=1}^{D} x_i = A - 1$, then:
Equation (3) shows that D has the same parity as $\sum_{i=1}^{D} x_i$, this then means $A - 1 \equiv D \pmod 2$.
By the definition of α we get that $\alpha = A$.

So,

$$\sum_{i=1}^{D} \tilde{x}_i = \sum_{i=1}^{D} x_i + \sum_{i=1}^{D} tv_i < (\alpha - 1) + 1 = \alpha$$

Gathering everything, we just showed $\sum_{i=1}^{D} x_i < A \Rightarrow \sum_{i=1}^{D} \tilde{x}_i < \alpha$.
Case 3: $\sum_{i=1}^{D} x_i > B$
Very similar to the previous proof. It can be shown that $\sum_{i=1}^{D} x_i > B \Rightarrow \sum_{i=1}^{D} \tilde{x}_i > \beta$.

■

Since we proved it in both direction we are done. ■