# Deep Learning

MEEC

## Homework 1

**Authors:**

Rodrigo Faria (100081) rodrigo.faria@tecnico.ulisboa.pt
Ricardo Silva (100071) ricardo.querido@tecnico.ulisboa.pt

$\boxed{\textbf{Group 35}}$

**2023/2024 – 1º Semester, P2**

The workload was evenly distributed among the group elements. To get acquainted with the tools used on the tasks, both group elements did the problems separately, sharing their insights at the end of each question. In the end, we created this document expressing our solutions for the problems presented to us.

## 1 Question 1 - Medical image classification with linear classifiers and neural networks

### 1.1 Problem 1 - Linear Classifiers

#### 1.1.1 Problem 1a - Perceptron

The Perceptron is one of the simplest neural network architectures, it forms the basis for more advanced models. It works pretty well on linear separable problems since the algorithm is guaranteed to find a separating hyperplane. When this condition isn't true, the model performs poorly because it can only learn linear decision boundaries, struggling when the data isn't linearly separable.

For this task, we implemented a Perceptron for Multi-Class classification and trained it for 20 epochs with a learning rate of 1.0 on the training set (gray-scale 28x28 images that fall under four categories). We can check the performance of the model in Table 1 and Figure 2. The outcome shows that the model couldn't find an optimal solution, and so it reached a state of divergence during the optimization process. Therefore, we conclude that the data isn't linearly separable.

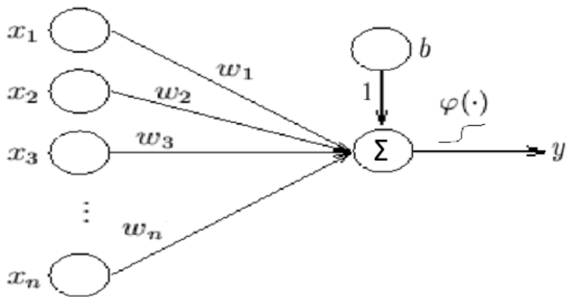


Figure 1: Perceptron Model

| Train Acc | Val Acc | Test Acc |
|---|---|---|
| 0.4654 | 0.4610 | 0.3422 |

Table 1: Perceptron's final accuracies

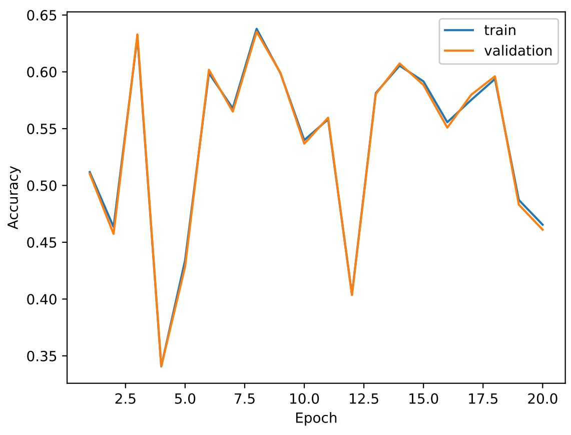


Figure 2: Perceptron's performance

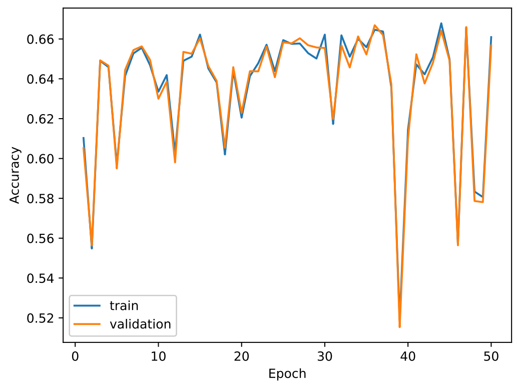
#### 1.1.2 Problem 1b - Logistic Regression

Similarly to the Perceptron, the Logistic Regression belongs to the family of linear classifier models, it is also discriminative, but instead of being non-probabilistic, it maximizes conditional likelihood. The loss function is the negative log-likelihood, also known as cross-entropy, and is described by the following equation:

$$L(\boldsymbol{W};(x;y)) = log \sum_{y'} exp(w_{y'}^T \phi(x)) - w_y^T \phi(x) \tag{1}$$

Since it doesn't have a closed-form solution we have to find the parameters that minimize the loss function. One great thing about the Logistic Regression is that it is strictly convex, which means that any local minimum is a global minimum. And so, by using Gradient methods such as gradient descent we will always reach the global optimum **W** with the appropriate choice of step-size $\eta_k$.

In this task, we trained the LR model for 50 epochs with stochastic gradient descent as our training algorithm and used learning rates of 0.01 and 0.001.



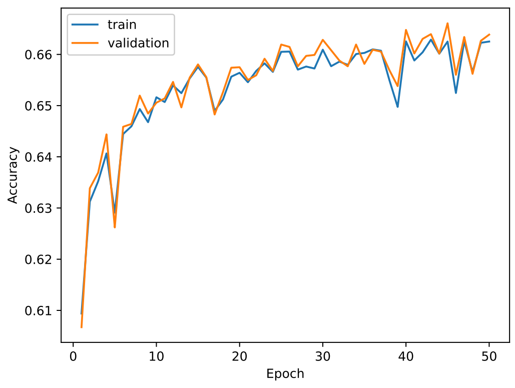


Figure 3: LR performance with $\eta = \{0.01, 0.001\}$, respectively

By analyzing Figure 3, we can see huge differences in results. On the left plot, we verify that the model diverged during the optimization algorithm. This may have happened because the step size was too high, leading to overshooting on the update step. Contrarily, on the right plot, the model seems to converge, meaning that a smaller step size may be beneficial since there are reduced oscillations in the parameter space. This reinforces the idea that choosing an appropriate step size is crucial for the success of the model.

Even though the graphs present huge differences, there are still similarities. For instance, the noisy profile of both plots derives from SGD, since it approximates the gradient with a noisy, unbiased version based on a single sample (we will delve deeper into this concept in a future section). Moreover, the fact that the train and validation accuracies are practically overlapped may indicate overfitting of the model, which is confirmed when we look at the discrepancy between the final test and train/validation accuracies in Table 2. This also happens in the Perceptron model.

| $\eta$ | Train Acc | Val Acc | Test Acc |
|---|---|---|---|
| 0.01 | 0.6609 | 0.6568 | 0.5784 |
| 0.001 | 0.6625 | 0.6639 | 0.5936 |

Table 2: Logistic Regression final accuracies

## 1.2 Problem 2 - Non-linear Classifiers

### 1.2.1 Problem 2a - Logistic Regression vs Multi-layer Perceptron

**Claim**

”A logistic regression model using pixel values as features is not as expressive as a multi-layer perceptron using relu activations. However, training a logistic regression model is easier because it is a convex optimization problem.”

The present claim is true. Since the Logistic Regression is a linear classifier, it can only model linear decision boundaries, making it less expressive than an MLP with ReLU activations, which is a non-linear classifier, allowing it to model more complex relationships between features and the target variable.

Even though the MLP with ReLU activations has an upper hand on expressiveness, it is way harder to train because of the non-linearities introduced by ReLU. Being a non-convex problem means that local minima may not be global minima, and therefore, the optimization landscape may have regions where the gradient information is not reliable for finding the global minimum. Whereas, the Logistic Regression is a convex optimization problem, and so any local minimum is also a global minimum, meaning that gradient-based optimization algorithms such as SGD are guaranteed to converge to the global minimum.

### 1.2.2 Problem 2b - Multi-layer Perceptron

The Multi-Layer Neural Network, also called Feed-forward neural network is a complex model that uses the Perceptron as its building blocks. Each hidden layer computes a representation of the input and propagates it forward. If the activation functions of the hidden layers are non-linear, the network is capable of learning complex, non-linear mappings from input features to output predictions. Without non-linear activation functions, the model would be limited to representing linear transformations of the input features. Training this model involves using a supervised learning approach called backpropagation, where the algorithm calculates the gradient of the loss function concerning the parameter space and updates them using optimization techniques, such as gradient descent. As stated by the Universal Approximation Theorem, a feed-forward neural network with a single layer containing a sufficient number of neurons can approximate any continuous function, highlighting the expressiveness power of MLPs.

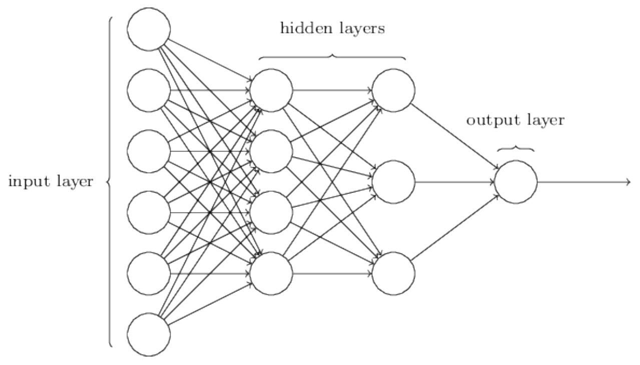


Figure 4: Multi-layer Perceptron Model

To test this model on the presented dataset, we used the same training algorithm of the previous task for 20 epochs with a learning rate of 0.001. The neural network architecture had one hidden layer with 200 units, a ReLU activation for the hidden layer, and a cross-entropy loss in the output layer. The initial biases were set to zero and the weight matrices to $w_{ij} \sim \mathcal{N}(0.1,\, 0.1^2)$ .

The total train loss was 57795.7751 and the model's performance on the different data sets can be seen in Table 3. We can also see the evolution of the model during the training in Figure 5. It seems that until epoch 2.5 the model was converging smoothly, and from then on further adjustments had diminishing returns. Furthermore, the variance of the accuracies in the difference datasets is smaller than in the previous model, deriving from the generalization capability of non-linear classifiers.

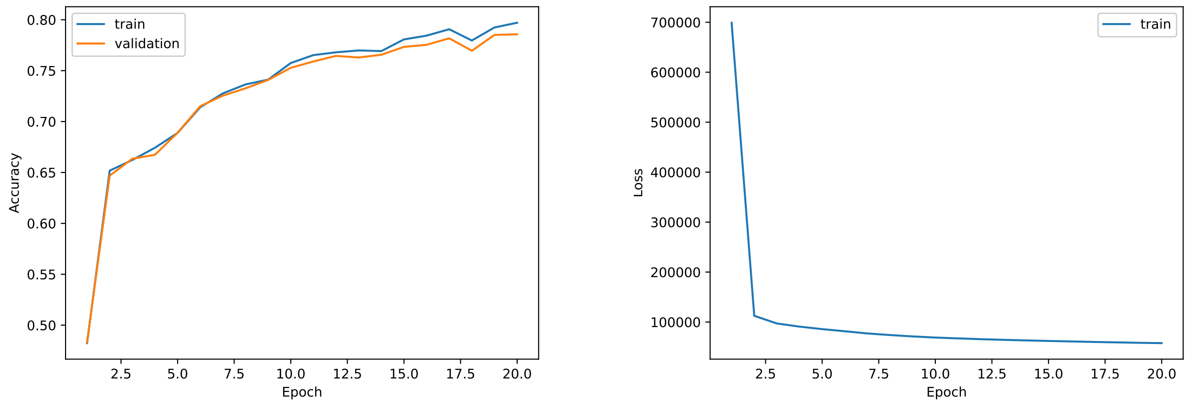


Figure 5: MLP's accuracy and loss plots

Comparing the MLP performance to the Perceptron and the Logistic Regression, it becomes evident that non-linear classifiers succeed when we don't know if the data is linearly separable. It comes back to the claim presented in section 1.2.1, the expressiveness power of a MLP with ReLU activations on the hidden layer is much bigger than a linear classifier's, but the fine-tuning of optimal hyperparameters is much harder.

| Model | Train Acc | Val Acc | Test Acc |
|---|---|---|---|
| Perceptron | 0.4654 | 0.4610 | 0.3422 |
| Logistic Regression | 0.6609 | 0.6568 | 0.5784 |
| MLP | 0.7970 | 0.7857 | 0.7448 |

Table 3: Performance of the difference models

## 2 Question 2 - Medical image classification with an autodiff toolkit

### 2.1 Problem 1 - Logistic Regression

As we already referred to in section 1.1.2, the learning rate is a fundamental hyperparameter for the convergence of the optimization algorithm. We trained a LR model with three different learning rates to study its influence on the model performance. These results are present in Table 4 and Figure 6. Even though, a learning rate of 0.01 reported a better validation accuracy, it got a worse test accuracy when compared to the configuration with a learning rate 10 times smaller. This phenomenon may reflect overfitting of the model, but it can also mean that the learning speed is too high, and so, the model converges too quickly, potentially overshooting the optimal parameter values and leading to poor generalization. Sometimes the effect of learning too quickly may even lead to divergence.

| $\eta$ | Train loss | Val Acc | Test Acc |
|---|---|---|---|
| 0.1 | 0.9687 | 0.6224 | 0.5577 |
| 0.01 | 0.9370 | 0.6543 | 0.6200 |
| 0.001 | 1.0143 | 0.6169 | 0.6503 |

Table 4: Learning rate influence in performance

### 2.2 Problem 2 - Neural Networks

#### 2.2.1 Problem 2a - Batch size influence

The batch size is another important hyperparameter that highly influences the performance of machine learning models. The choice of batch size can impact characteristics such as convergence speed, generalization, computational efficiency, etc. On the one hand, larger batch sizes promote faster convergence because the model makes less frequent parameter updates, more efficient training because they allow for parallelization of computations, and can provide a more stable and smoother optimization process.

On the other hand, smaller batch sizes usually take more time to converge since they provide more frequent updates but sometimes this phenomenon might help the model escape local minima and converge quicker. Moreover, smaller batch sizes introduce more noisy but less biased estimates of the gradient, which can improve generalization.

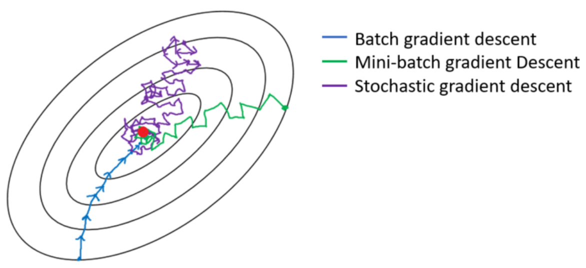


Figure 7: Otimization methods based on batch size

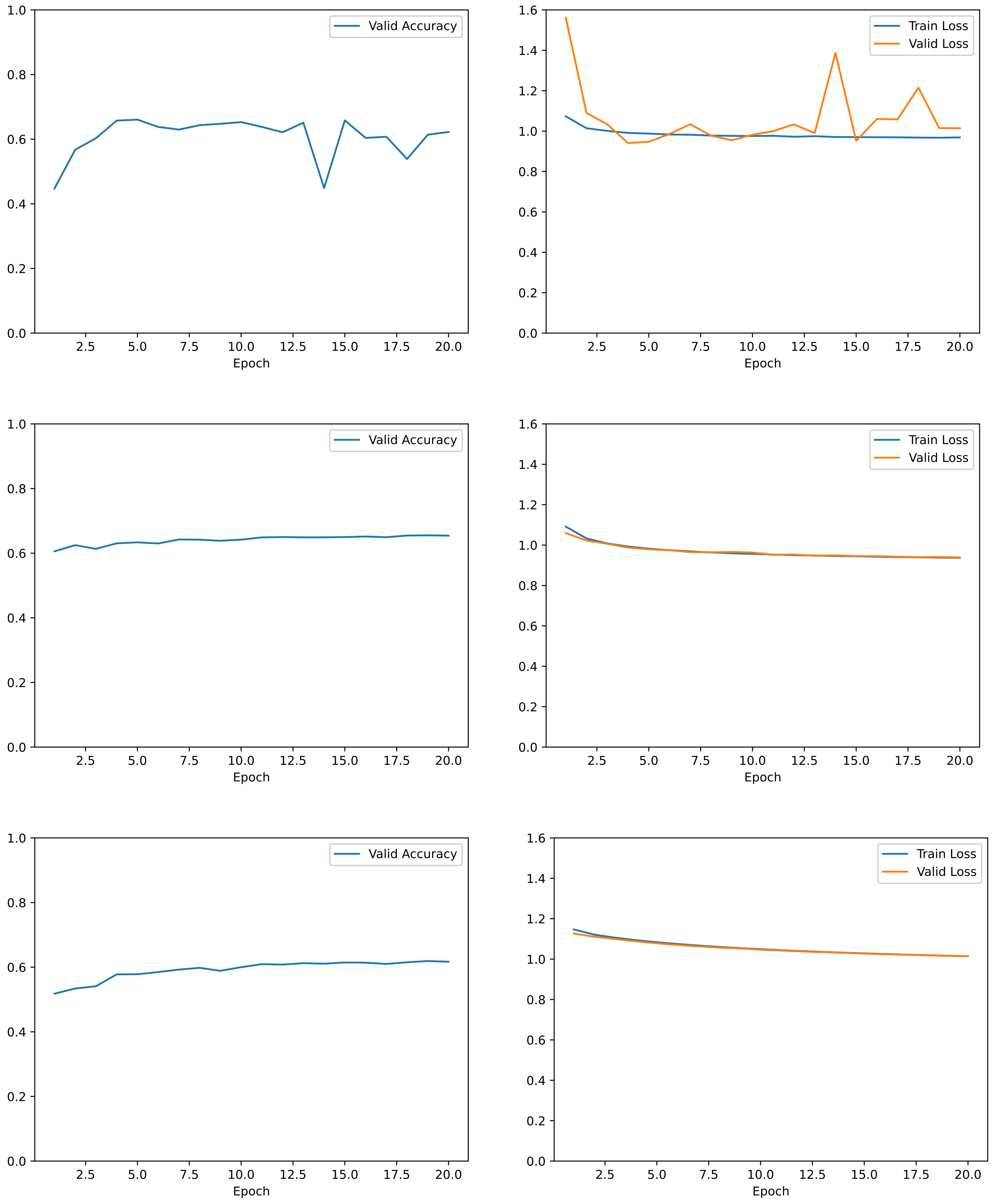


Figure 6: Performance with $\eta = \{0.1, 0.01, 0.001\}$, respectively

| batch size | Train loss | Val Acc | Test Acc | Time |
|---|---|---|---|---|
| 16 | 0.4809 | 0.8169 | 0.7524 | 159s |
| 1024 | 0.8787 | 0.6907 | 0.7221 | 15s |

Table 5: Batch size influence in the model's performance

Looking at Table 5 and Figure 8, we can see that a smaller batch size leads to longer training time and a better generalization of the model. Whereas, a larger batch size generates higher loss values but a smoother loss curve, indicating a more stable convergence during training.

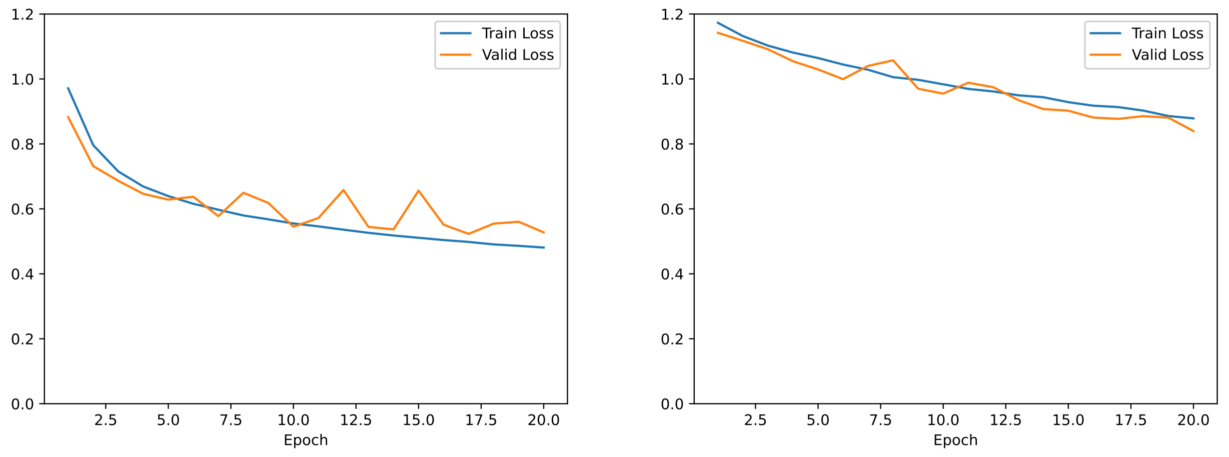


Figure 8: Loss with batch size 16 and 1024, respectively

### 2.2.2 Problem 2b - Learning rate influence

Similarly to what happened in section 2.1, the learning rate, $\eta$, highly influenced the performance of the model since the optimization process is transversal to both linear and non-linear classifiers. In both cases, we used stochastic gradient descent, which can be described by the following equation 2.2.2, with $\theta = \{(\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)})\}_{l=1}^{L+1}$ and $j$ representing a single training sample.

$$\begin{cases} \mathcal{L}_j(\theta) = \lambda\Omega(\theta) + L(\boldsymbol{f}(\boldsymbol{x_j};\boldsymbol{\theta}), y_j) \\ \theta^{(l+1)} \leftarrow \theta^{(l)} - \eta\boldsymbol{\nabla}_\theta\mathcal{L}_j(\theta^{(l)}) \end{cases} \tag{2}$$

Looking at Table 6, we conclude that the best configuration was the one with $\eta = 0.1$ in terms of validation accuracy but the one with $\eta = 0.01$ had the best performance on the test dataset, reflecting a better generalization and therefore, less overfitting of the model (we can notice some overfitting of the model from the epoch 7.5 on, in Figure 9. The configuration with $\eta = 1.0$ had a learning rate too high, leading to divergence in the optimization process.

| $\eta$ | Train loss | Val Acc | Test Acc |
|---|---|---|---|
| 1.0 | 1.1714 | 0.4721 | 0.4726 |
| 0.1 | 0.4333 | 0.8246 | 0.7505 |
| 0.01 | 0.4855 | 0.8157 | 0.7694 |
| 0.001 | 0.8421 | 0.6922 | 0.7127 |

Table 6: Learning rate influence in the model's performance

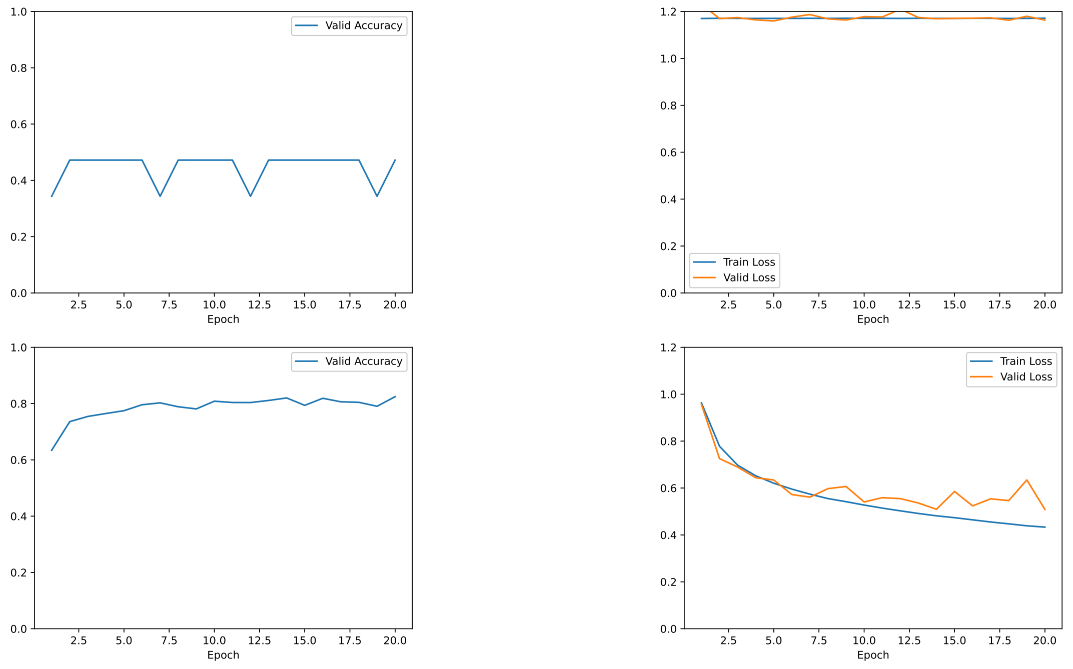


Figure 9: Performance with $\eta = \{1.0, 0.1\}$, respectively

### 2.2.3 Problem 2c - L2 Regularization vs Dropout

Before analyzing the final results of our model, let's try to understand both regularization methods to build some intuition on the matter. Both techniques are commonly used to prevent overfitting in machine learning models. The ***L*2 Regularizer** is defined by equation 2.2.3.

$$\Omega(\theta) = \frac{1}{2}\sum_{l=1}^{L+1} ||\boldsymbol{W}^{(l)}||^2 \tag{3}$$

When we apply Lasso, a.k.a $L2$, the SGD turns into equation 2.2.3. From there we conclude that Lasso creates a weight decay effect, penalizing large weights and leading to sparse solutions, which can be interpreted as feature selection.

$$\begin{aligned} \boldsymbol{W}^{(l+1)} &\leftarrow \boldsymbol{W}^{(l)} - \eta \boldsymbol{\nabla}_{\boldsymbol{W}^{(l)}} \mathcal{L}_i(\theta^{(l)}) \\ &= \boldsymbol{W}^{(l)} - \eta(\lambda\Omega(\theta) + L(\boldsymbol{f}(\boldsymbol{x_i};\boldsymbol{\theta}), y_i)) \\ &= (1 - \eta\lambda)\boldsymbol{W}^{(l)} - \eta \boldsymbol{\nabla}_{\boldsymbol{W}^{(l)}} L(\boldsymbol{f}(\boldsymbol{x_i};\boldsymbol{\theta}), y_i) \end{aligned} \tag{4}$$

**Dropout** sets the output of hidden units to 0 with probability $p$, which prevents hidden units from co-adapting to other units, forcing them to be more generally useful. At test time, keeps all units, with the outputs multiplied by $1 - p$. This method is categorized as a form of adaptive regularization. Since Dropout introduces randomness by temporarily removing a random subset of neurons during training, this can be beneficial when dealing with non-linear relationships in the data, preventing the model from relying too much on specific neurons and encouraging the network to learn more robust features. It can also be viewed as a form of ensemble learning since it trains multiple models with shared parameters but different dropout patterns, leading to improved generalization by averaging the predictions of all ”sub-models” during prediction.

| Regularization Method | Train loss | Val Acc | Test Acc |
|---|---|---|---|
| None | 0.2164 | 0.8608 | 0.7675 |
| L2($\lambda = 0.0001$) | 0.2701 | 0.8535 | 0.7694 |
| Dropout($p = 0.2$) | 0.3659 | 0.8575 | 0.7845 |

Table 7: Regularization influence in the model’s performance

When we look at Table 7, we may be fooled by the values of train loss and validation accuracy of the configurations with regularization when comparing to the one without, but as soon as we look at the Test accuracies, we understand the generalization power that regularization methods bring to the table. L2 with a decay coefficient equal to 0.0001 didn’t improve the initial model that much, but Dropout did succeed on this task. It becomes apparent the presence of overfitting on the initial model when looking at Figure 10, where the train loss keeps decreasing and the validation loss starts to stagnate. In the model with Dropout’s plot the train and validation loss stay pratically overlapped during all training.

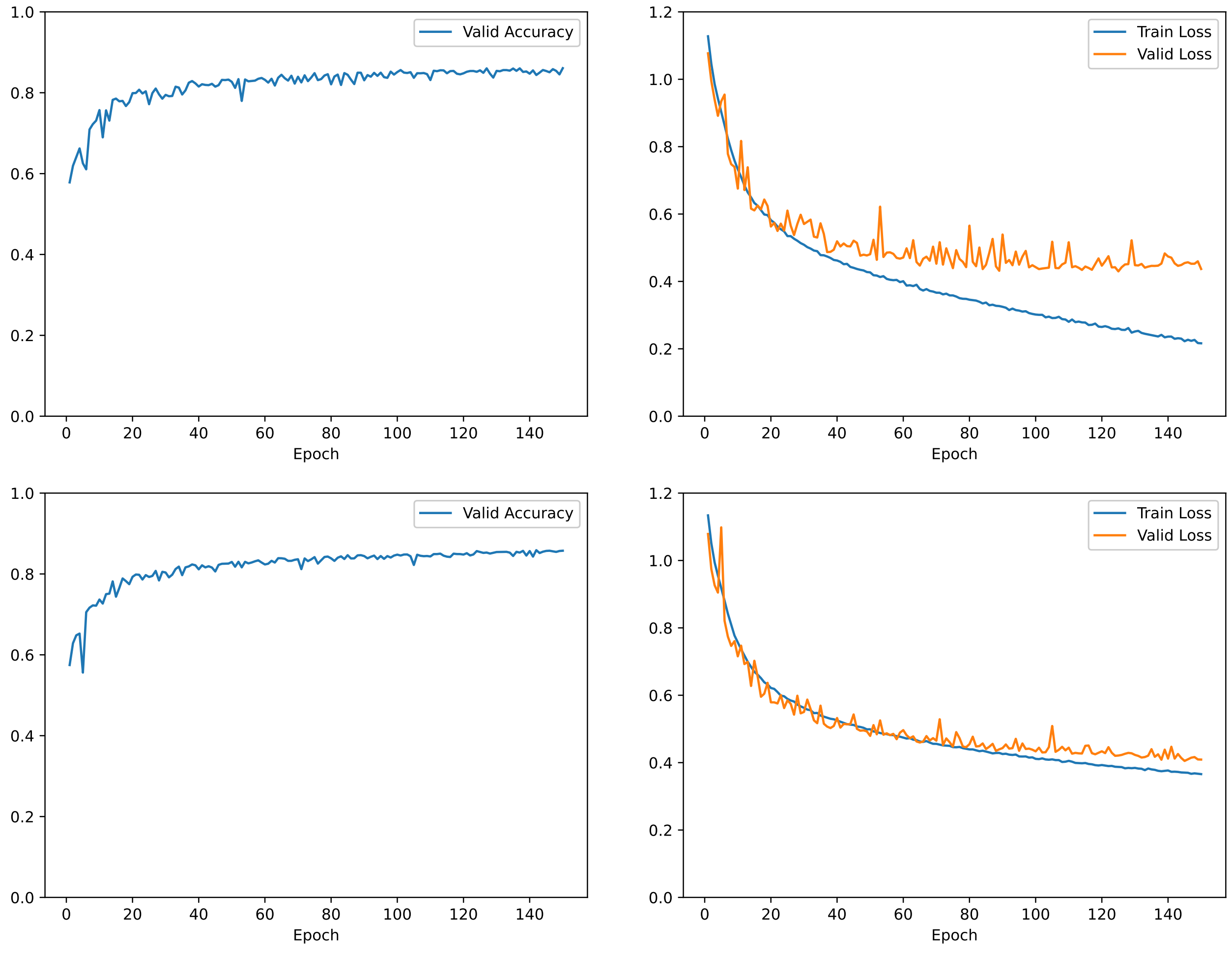


Figure 10: MLP’s performance without regularization and with Dropout, respectively

# 3 Question 3 - Multi-layer Perceptron that computes a Boolean function

## 3.1 Problem 1

### 3.1.1 Problem 1a

A Perceptron is commonly employed to classify linearly separable problems by partitioning the input space with a hyperplane. This hyperplane is characterized by the weights and bias incorporated into the calculation of the Perceptron's pre-activation output, as expressed by the equation:

$$z(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = 0 \tag{5}$$

The activation function then aims to categorize the data based on its location in the sub-spaces divided by this hyperplane. Given the binary nature of the classification problem, an activation function $h : \mathbb{R} \rightarrow \{-1, +1\}$ can be employed.

Assuming that points above the hyperplane belong to class +1 and those below belong to class -1, the activation function can be defined as:

$$\hat{y} = h(z) = \text{sign}(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases} \tag{6}$$

We can attempt to represent the actual Boolean function $f(\boldsymbol{x})$ by initializing all the weights with ones, obtaining the following expressions:

$$z(\boldsymbol{x}) = \sum_{i=1}^{D} x_i + b \tag{7}$$

$$\hat{y} = \begin{cases} +1 & \text{if } \sum_{i=1}^{D} x_i \geq -b \\ -1 & otherwise \end{cases} \tag{8}$$

However, this representation can only handle one bound of the interval $[A, B]$, which may suffice when the interval encompasses one extreme (i.e. $A = -D$ or $B = D$), but, for other cases, correct classification becomes unattainable. This limitation underscores how a single Perceptron excels only in classifying linearly separable classes.

Figure 11 illustrate an example with input data having two features as input dimensions and an interval $[A, B] = [-1, 1]$. Despite the attempt made by the Perceptron in Figure 12 to separate the data with a single straight line, it is evident that the data is not linearly separable leading to unsatisfactory results.

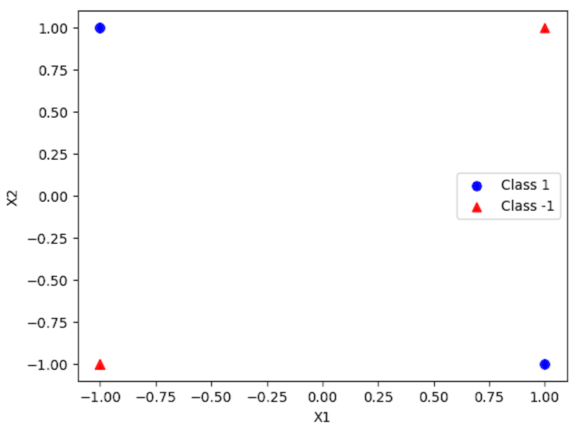


Figure 11: Class distribution for $D = 2$ and $[A, B] = [-1, 1]$

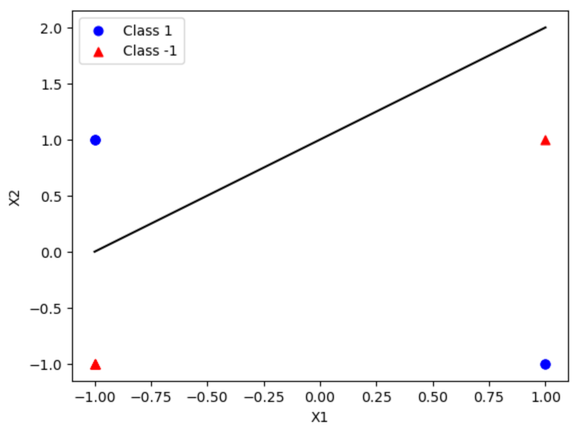


Figure 12: Attempted line separation by Perceptron

### 3.1.2 Problem 1b

We can address the previous problem with a multilayer perceptron utilizing a single hidden layer with two hidden nodes. Since a single Perceptron attempts to separate the data with a single hyperplane, and given that our data can be contained in a linearly connected subspace (in this case the data that represents class +1), we aim to define that subspace using two hyperplanes, where each hyperplane establishes the bounds of the subspaces that define the class of the input data.

As shown on 8, a Perceptron can represent a single bound of the interval. Considering this, we can delineate the two dividing bounds of the interval using two separate perceptrons, with their joint sets constituting the intended interval.

- For $\sum_{i=1}^{D} x_i \geq A$, using as weights $\boldsymbol{w_1} = [\mathbf{+1}]$ and bias $b_1 = -A$

$$h_1(z_1) = \begin{cases} +1 & \text{if } \sum_{i=1}^{D} x_i \geq A \\ -1 & otherwise \end{cases} \tag{9}$$

- For $\sum_{i=1}^{D} x_i \leq B$, using as weights $\boldsymbol{w_2} = [\mathbf{-1}]$ and bias $b_2 = B$

$$h_2(z_2) = \begin{cases} +1 & \text{if } \sum_{i=1}^{D} x_i \leq B \\ -1 & otherwise \end{cases} \tag{10}$$

The output of the multilayer perceptron is then defined as the intended function $f(\boldsymbol{x})$:

$$\hat{y} = \begin{cases} +1 & \text{if } h_1(z_1) + h_2(z_2) = 2 \\ -1 & otherwise \end{cases} = \begin{cases} +1 & \text{if } A \leq \sum_{i=1}^{D} x_i \leq B \\ -1 & otherwise \end{cases} = f(\boldsymbol{x}) \tag{11}$$

The multilayer perceptron remains robust to minor input variations if the given hyperplanes have a margin between them and their closest data points. The margins can be assessed with the following expression:

$$M = \frac{1}{\|\boldsymbol{w}\|_2} = \frac{1}{\sqrt{D}} \tag{12}$$

Thus, there is a margin of at least $2M$ between the closest points inside the subspace defining class +1 and the points in the subspaces belonging to the other class -1, allowing for some input noise.

However, this assertion does not hold in certain scenarios, such as when the hyperplanes coincide, exemplified by the case $[A, B] = [0, 0]$. In this situation, where both hyperplanes overlap due to identical lower and upper limits in the interval, the margin between the hyperplane and the nearest data points becomes negligible as the points coincide with the hyperplane. To address this particular challenge, we examine all potential values resulting from the summation $\sum_{\boldsymbol{x}} x_i$ in the function $f(x)$.

$$\sum_{i=1}^{D} x_i \in \{-D, -(D+2), -(D+4), ..., D-4, D-2, D\} \tag{13}$$

Upon careful consideration of the presented information, it becomes evident that for cases where the bounds fall within the specified set of values, we have the flexibility to extend these bounds by one value each without sacrificing generality ($A' = A - 1$ and $B' = B + 1$). This extension effectively addresses the margin issue, resulting in a more robust model that accommodates input variations across all use cases.

Subsequently, the selection of appropriate hyperplanes is crucial, and this is achieved by choosing the bias term in accordance with the given intervals. The following expressions provide the means to determine the correct bias term based on the number of features and the specified problem intervals:

$$b_1 = \begin{cases} 1 - A & \text{if } (-1)^{D+A} = 1 \\ -A & \textit{otherwise} \end{cases} = \frac{1 + (-1)^{D+A}}{2} - A \tag{14}$$

$$b_2 = \begin{cases} 1 + B & \text{if } (-1)^{D+B} = 1 \\ B & \textit{otherwise} \end{cases} = \frac{1 + (-1)^{D+B}}{2} + B \tag{15}$$

It is important to consider that in cases of overlapping where the intervals fall outside the 13 set, the hyperplanes remain unchanged. This, however, does not pose an issue as in such instances it is not feasible to assign data to class +1.

Applying the obtained multilayer perceptron to the previous example yields the separation lines of Figure 13, correctly classifying the data into their respective classes (with a margin $M = 0.71$):

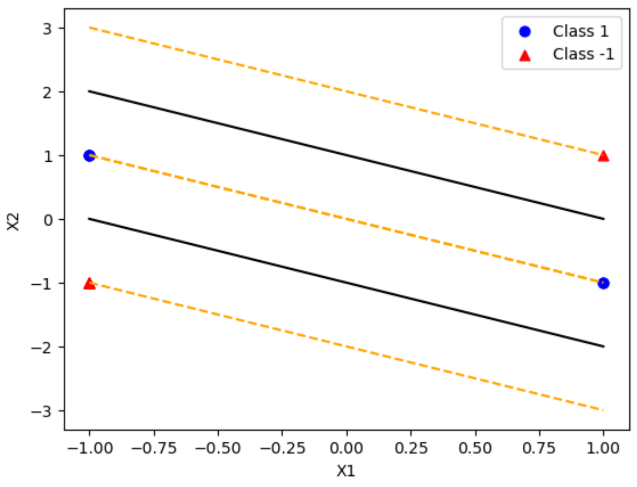


Figure 13: Hyperplanes obtained using a MLP with two hidden nodes and sign activation

### 3.1.3 Problem 1c

In this problem we use the rectified linear unit (ReLU) as the activation function, defined as:

$$h(z) = ReLU(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & otherwise \end{cases} \tag{16}$$

However, this choice introduces a new challenge: determining how to accurately assess the activation of both hidden perceptrons. Considering the pre-activation of the Multi-Layer Perceptron (MLP) output expressed as:

$$z_{out} = w_1^{(out)} h_1(z_1) + w_2^{(out)} h_2(z_2) + b^{(out)} \tag{17}$$

There are two approaches to determine the activation of hidden nodes. The first involves utilizing the same hidden weights and bias from the previous problem and checking if the output of both nodes is non-zero. However, this method introduces complexity, making it challenging to ascertain whether both nodes were activated at the MLP output. Even in the case of a positive output, it remains unclear whether it originates from a single node or both, necessitating intricate expressions for differentiation.

Alternatively, a simpler approach involves inverting the decision planes. This results in a zero output at the hidden layer if the points belong to the specified class and a positive value otherwise. By adopting this strategy, we can easily verify at the MLP output whether the received parameters equate to zero. The revised expressions for the activations of the hidden layers would then be:

- For $\sum_{i=1}^{D} x_i \geq A$, using as weights $\boldsymbol{w_1} = [\boldsymbol{-1}]$ and bias $b_1 = A - \frac{1+(-1)^{D+A}}{2}$

$$h_1(z_1) = \begin{cases} 0 & \text{if } \sum_{i=1}^{D} x_i \geq A - \frac{1+(-1)^{D+A}}{2} \\ z_1 & otherwise \end{cases} \tag{18}$$

- For $\sum_{i=1}^{D} x_i \leq B$, using as weights $\boldsymbol{w_2} = [\mathbf{+1}]$ and bias $b_2 = -(B + \frac{1+(-1)^{D+B}}{2})$

$$h_2(z_2) = \begin{cases} 0 & \text{if } \sum_{i=1}^{D} x_i \leq B + \frac{1+(-1)^{D+B}}{2} \\ z_2 & \textit{otherwise} \end{cases} \tag{19}$$

The output of the multilayer perceptron is then defined as:

$$\hat{y} = \begin{cases} +1 & \text{if } h_1(z_1) + h_2(z_2) = 0 \\ -1 & \textit{otherwise} \end{cases} = \begin{cases} +1 & \text{if } A - \frac{1+(-1)^{D+A}}{2} \leq \sum_{i=1}^{D} x_i \leq B + \frac{1+(-1)^{D+B}}{2} \\ -1 & \textit{otherwise} \end{cases} \tag{20}$$

As illustrated in the previous problem, the provided intervals are equivalent to those selected by the function $f(x)$, only improving the hyperplane margins.

Using as example the extra problem given ($D = 2$ and $A = B = 0$), the resulting MLP computes the following expression:

$$h(x) = \begin{cases} +1 & \text{if } -1 \leq \sum_{i=1}^{D} x_i \leq 1 \\ -1 & \textit{otherwise} \end{cases} = f(x) \tag{21}$$

The separation lines and class classification obtained align with those previously depicted in Figure 13.