1.5em 0pt

Deep Learning

Deep Learning - Homework 1 report

Autores:

Tiago Costa (100094) tiagomascosta@tecnico.ulisboa.pt
Miguel Luís Rente Lourenço (100044) miguel.rente.l@tecnico.ulisboa.pt

Group 20

2022/2023 – 1st Semester, P2

Contents

1 Question 1

1 a)

In this question the objective was to implement the update weights method in a single perceptron, using only the numpy library, just plain linear algebra.

Algorithm 1 Perceptron Algorithm

```
$$\hat{y}_b = \text{argmax}_k \left(\mathbf{w}^{(k)}\right)^T \phi(\mathbf{x}) \quad (1)$$
for y $\in \mathcal{Y}$ do
    if $\hat{y}_b \neq y$ then
        $$\text{Update } \mathbf{w}_y^{(k+1)} = \mathbf{w}_y^{(k)} + \phi(\mathbf{x}_n) \quad (2)$$
        $$\text{Update } \mathbf{w}_{\hat{y}_b}^{(k+1)} = \mathbf{w}_{\hat{y}_b}^{(k)} - \phi(\mathbf{x}_n) \quad (3)$$
    end if
end for
```

This perceptron algorithm is the base on which we did our code, we implemented this logic, updating the weights of the perceptron.

```python
class Perceptron(LinearModel):
    def update_weight(self, x_i, y_i, **kwargs):
        """
        x_i (n_features): a single training example
        y_i (scalar): the gold label for that example
        other arguments are ignored
        """
        y_hat = self.predict(x_i.reshape(1,-1))

        if y_hat != y_i:
            self.W[y_i] += x_i
            self.W[y_hat] -= x_i
```

Listing 1: Perceptron class update_weight method

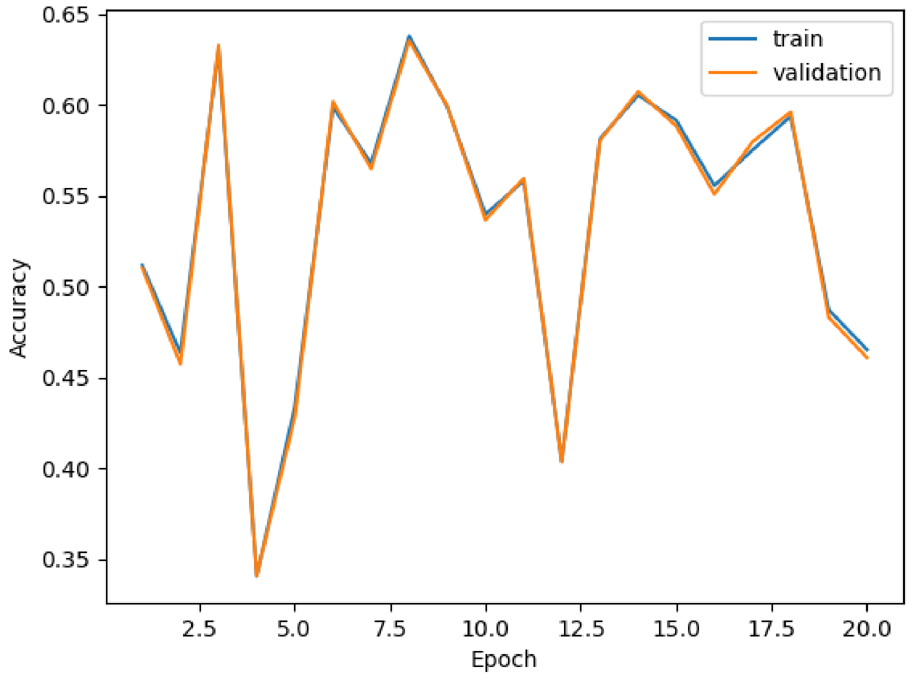


Figure 1: Train vs. Validation accuracy for the Perceptron

The perceptron class is a binary classifier, but the provided code implements it in a one vs all fashion. The perceptron has certain limitations to it, meaning that it can solve linearly separable problems, but it can't solve non-linearly separable problems. This means that the perceptron is very limited when it concerns the OCTMNIST dataset, resulting in both low training and validation accuracies. The final test accuracy was 0.3422, which is very low, meaning that this is not a very good predictive model for this dataset. We should use more complex models that can solve non-linearly separable problems.

1 b)

This exercise consisted of updating the update weight function, this time of the Logistic Regression class, using SGD(Stochastic gradient descent) as the training algorithm. This means that we use a softmax transformation on the scores computed to define the conditional probability, then we update the weights using stochastic gradient descent to minimize the loss function gradually.

Algorithm 2 SGD (Stochastic Gradient Descent)

```
1: $W^{(k+1)} = W^{(k)} + \eta_k e_y \phi(x)^T - \sum_{y'} P_W(y' \mid x) e_{y'} \phi(x)^T$
```

Algorithm 3 Softmax Transformation in Logistic Regression

```
1: Input: Raw scores $\mathbf{z}$ for each class
2: Output: Probability distribution $\mathbf{p}$ over classes
3: function SOFTMAX($\mathbf{z}$)
4:     Initialize an array $\mathbf{p}$ with zeros
5:     Compute the maximum score: $z_{\max} \leftarrow \max(\mathbf{z})$
6:     Compute the exponentials of the shifted scores:
7:     for $i \leftarrow 1$ to length($\mathbf{z}$) do
8:         exp_score $\leftarrow \exp(z_i - z_{\max})$
9:         $\mathbf{p}[i] \leftarrow$ exp_score
10:    end for
11:    Normalize to obtain probabilities:
12:    sum_exp $\leftarrow \sum_i$ exp_score
13:    for $i \leftarrow 1$ to length($\mathbf{p}$) do
14:        $\mathbf{p}[i] \leftarrow \frac{\mathbf{p}[i]}{\text{sum\_exp}}$
15:    end for
16:    return p
17: end function
```

```python
class LogisticRegression(LinearModel):
    def update_weight(self, x_i, y_i, learning_rate=0.001):
        """
        x_i (n_features): a single training example
        y_i: the gold label for that example
        learning_rate (float): keep it at the default value for your
  plots
        """

        scores = np.dot(self.W, x_i)

        exp_scores = np.exp(scores)
        prob = exp_scores / np.sum(exp_scores)

        prob[y_i] -= 1
        self.W = self.W - learning_rate * np.outer(prob, x_i)
```

Listing 2: LogisticRegression class update_weight method

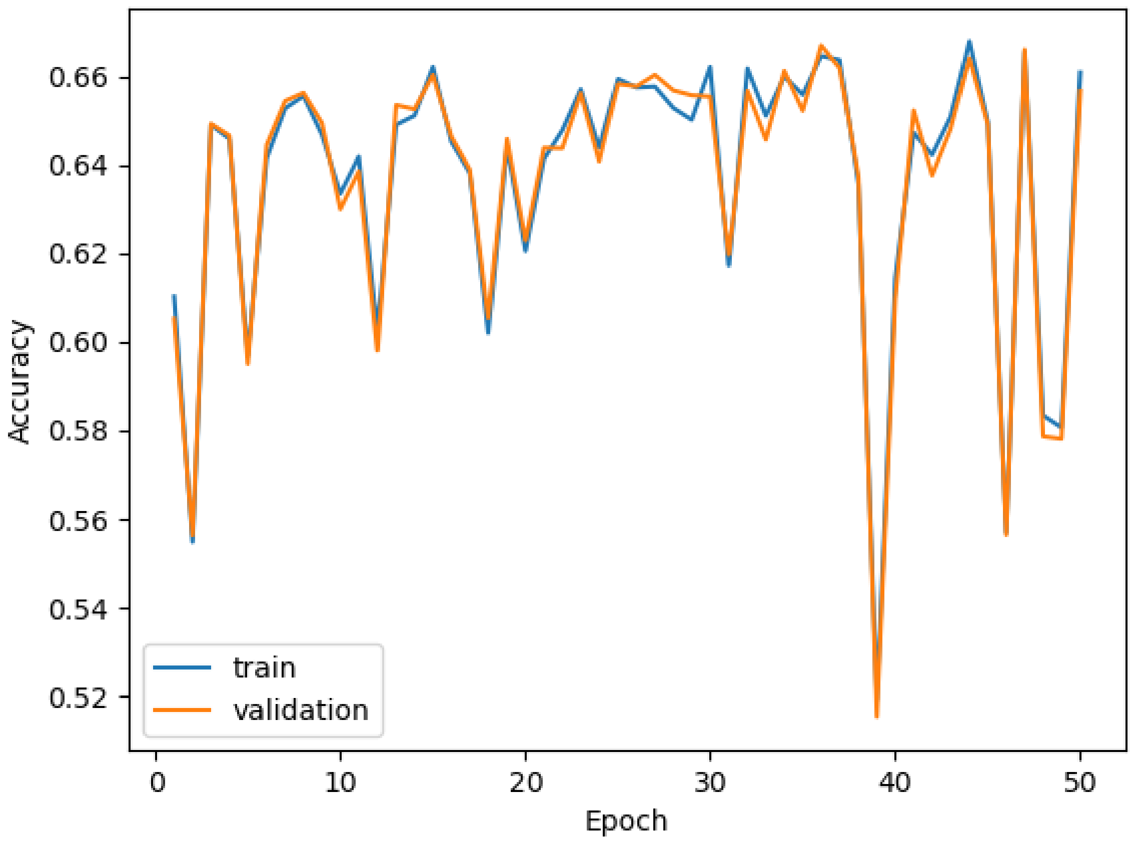


Figure 2: Training and validation accuracies for $\eta = 0.01$

The logistic regression model with Stochastic Gradient Descent (SGD) has been employed to tackle the challenges posed by the OCTMNIST dataset. Unlike the perceptron, which is inherently a binary classifier, logistic regression extends its capabilities to handle multiple classes using the SGD optimization technique.

The logistic regression model, leveraging SGD, exhibits improved performance compared to the perceptron. The training and validation accuracies are slightly higher, indicating that the model is better capturing the underlying patterns in the dataset. The final test accuracy, reaching 0.5784, demonstrates the model's ability to generalize well to previously unseen data.

Unlike the perceptron, logistic regression with SGD can handle non-linearly separable problems to a certain extent. This increased flexibility contributes to its enhanced performance on the OCTMNIST dataset. While logistic regression provides improved results, it's worth noting that the accuracy is still moderate. This suggests that more complex models, perhaps incorporating additional layers or features, could further enhance predictive performance.

In conclusion, the logistic regression model with SGD presents a step forward in addressing the challenges posed by the OCTMNIST dataset. While achieving a higher accuracy than the perceptron, there is room for exploration of more sophisticated models to push the predictive capabilities even further.

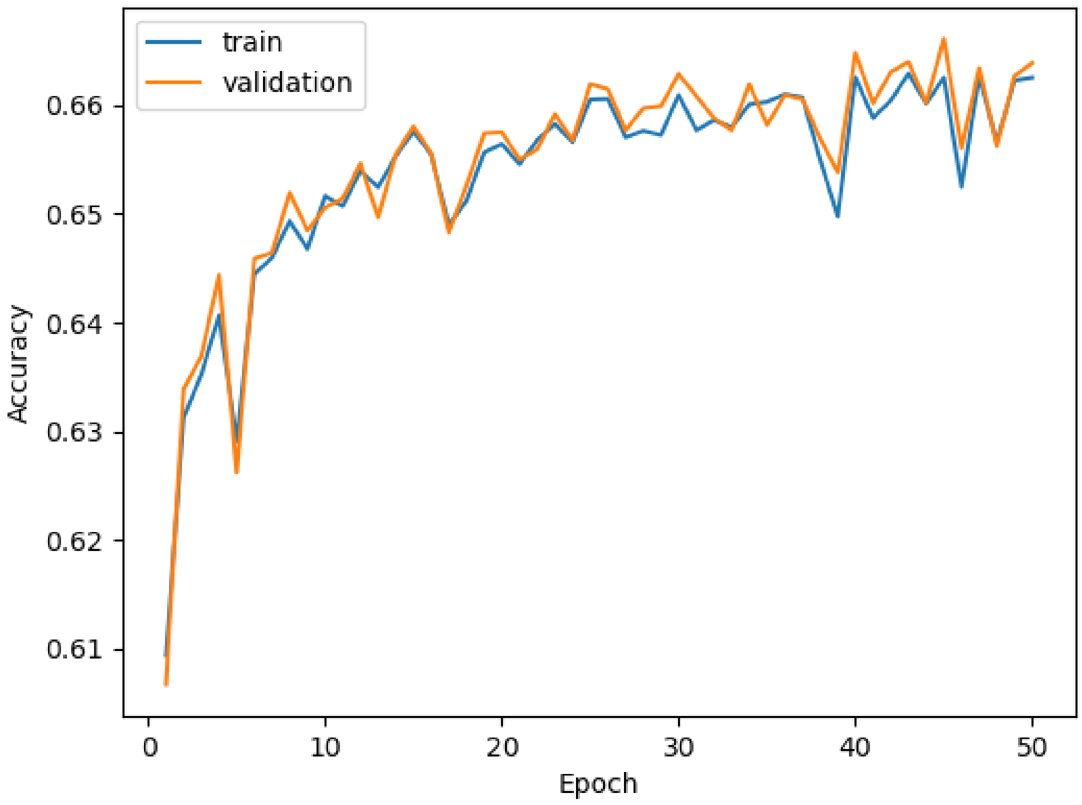


Figure 3: train vs val log lr=0.001

Exploring the impact of hyperparameter tuning on the logistic regression model with Stochastic Gradient Descent (SGD) has yielded insightful results. Specifically, a reduction in the learning rate from 0.01 to 0.001 has been employed, resulting in notable improvements in both stability and overall accuracy across training, validation, and test datasets.

The decision to decrease the learning rate was motivated by the desire to enhance the model's convergence and adaptability. A higher learning rate can lead to overshooting the optimal weights and may hinder the model's ability to converge to the global minimum of the loss function. By dialing down the learning rate to 0.001, we observed increased stability in the accuracy values throughout the training process.

The training accuracy, validation accuracy, and, most importantly, the final test accuracy all exhibited positive responses to this adjustment. The final test accuracy, reaching 0.5936, surpasses the previous result of 0.5784, indicating that the model with the reduced learning rate generalizes better to unseen data.

This improvement aligns with the expectation that a lower learning rate allows the model to make more cautious weight updates, preventing overshooting and ensuring a more careful exploration of the parameter space. While the training process might take longer with a smaller learning rate, the trade-off in terms of improved stability and accuracy makes it a worthwhile adjustment.

In summary, the decision to decrease the learning rate from 0.01 to 0.001 has led to enhanced stability and improved accuracy across all evaluation metrics, underscoring the importance of thoughtful hyperparameter tuning in achieving optimal model performance.

2 a)

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.”

Justification:

1. **Logistic Regression vs. Multi-layer Perceptron (MLP) Expressiveness:**

- **Logistic Regression:** It is a linear model that applies a logistic (softmax) function to the linear combination of input features. This makes it suitable for binary classification problems. However, it’s limited to learning linear decision boundaries.
- **MLP with ReLU Activations:** The presence of ReLU (Rectified Linear Unit) activations introduces non-linearity, allowing the network to learn complex, non-linear relationships within the data. MLPs with appropriate architecture (multiple hidden layers, non-linear activations) can capture more intricate patterns.

2. **Training Difficulty:**

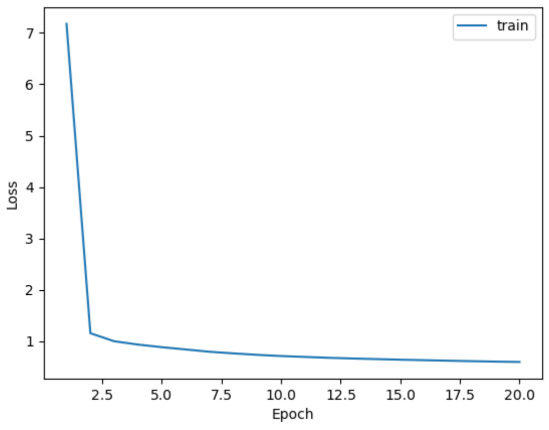
- **Logistic Regression:** It is indeed easier to train because it involves convex optimization. The optimization problem is convex, meaning it has a single, global minimum that makes it relatively straightforward to find the optimal parameters.
- **MLP with ReLU Activations:** Training an MLP, especially with multiple hidden layers and non-linear activations, involves a non-convex optimization problem. This can lead to the presence of multiple local minima, making convergence more challenging. However, techniques like stochastic gradient descent with backpropagation and weight initialization strategies have proven effective in training deep networks.

3. **Data Complexity:**

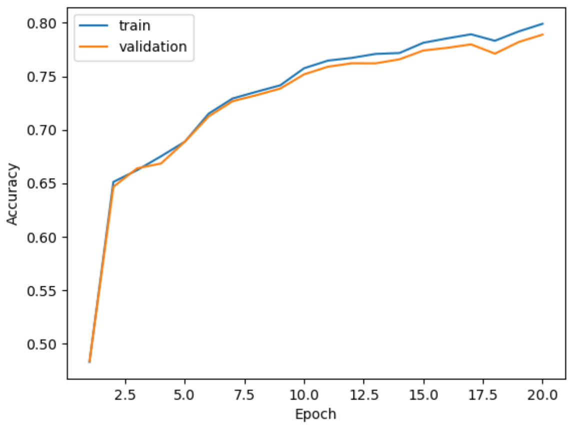
- **Logistic Regression:** Suitable for simpler tasks where the decision boundary is close to linear.
- **MLP with ReLU Activations:** More expressive for complex tasks with non-linear relationships in the data.

2 b)

To address the binary classification task on the OCTMNIST dataset, we employ a Multi-Layer Perceptron (MLP) with a carefully designed architecture. The model comprises a single hidden layer housing 200 hidden units, and the activation function for this hidden layer is the rectified linear unit (ReLU). The output layer employs the softmax activation function, facilitating the model’s transformation into a probabilistic classifier.



(a) MLP Loss



(b) MLP Train vs. Validation

Figure 4: MLP Training Plots

The superior performance of the MLP can be attributed to its capacity to learn hierarchical and non-linear features through the hidden layers. Unlike the linear decision boundaries of logistic regression and the limitations of perceptron, the MLP's ability to model complex relationships led to a substantial boost in accuracy.

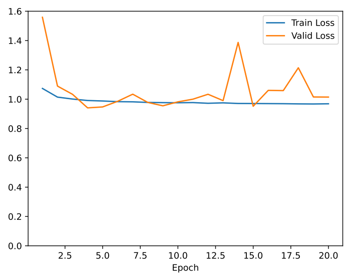
In summary, while logistic regression and perceptron models struggled with the non-linearity of the OCTMNIST dataset, the Multi-Layer Perceptron demonstrated its prowess in capturing intricate patterns, resulting in significantly improved accuracy. The choice of model is evidently crucial when tackling complex image classification tasks, emphasizing the importance of employing architectures capable of learning hierarchical representations. The MLP, with its non-linear activation functions and deeper architecture, demonstrated remarkable adaptability to the intricate patterns within the OCTMNIST dataset. The training dynamics, as illustrated by the loss curve, showcased a rapid reduction in loss during the initial epochs, stabilizing around 1. The model achieved a final test accuracy of 0.7543, signifying a substantial improvement over both logistic regression and perceptron models.

2 Question 2

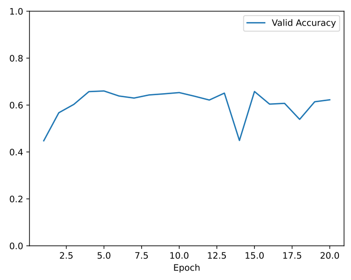
In this question we are going to use an autodiff toolkit, this means using pytorch.

1.

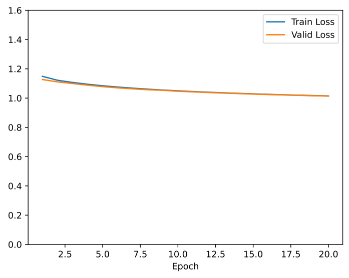
We implemented a simple linear model with logistic regression, with SGD(Stochastic Gradient Descent) as the training algorithm, with different η values: $\{0.001, 0.01, 0.1\}$



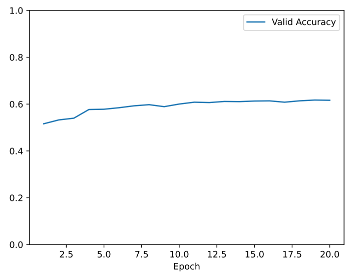
(a) Training and validation loss with $\eta = 0.1$



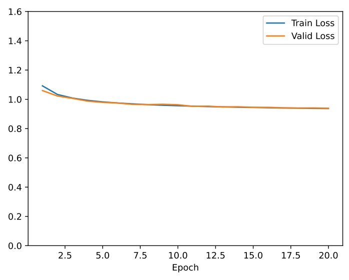
(b) Validation accuracy with $\eta = 0.1$



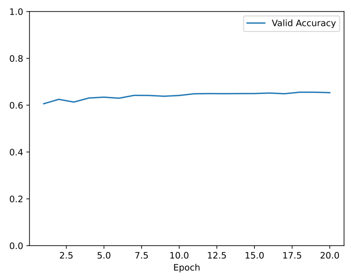
(c) Training and validation loss with $\eta = 0.001$



(d) Validation accuracy with $\eta = 0.001$



(e) Training and validation loss with $\eta = 0.01$



(f) Validation accuracy with $\eta = 0.01$

Figure 5: Plots for Question 2.1 with different learning rates

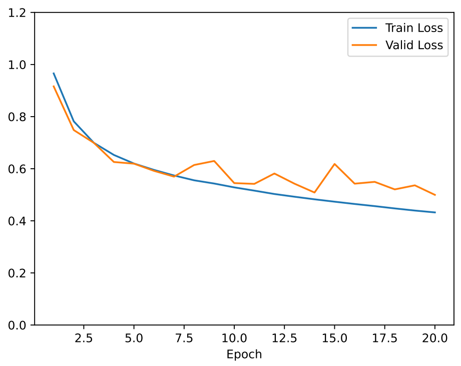
η	Test Accuracy
0.1	0.5577
0.001	0.6503
0.01	0.6200

Table 1: Test Accuracies for Different η Values

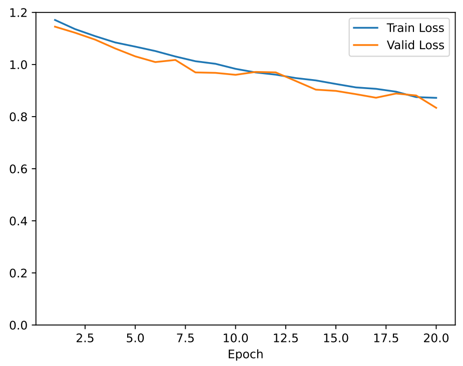
We observed a notable trend in our experiments, indicating that the choice of the learning rate (η) significantly impacts the model's performance. Surprisingly, as we decreased the learning rate, we witnessed an improvement in test accuracy. This phenomenon suggests that a more conservative updating strategy, achieved through lower learning rates, contributes to better generalization and, consequently, enhanced performance on unseen data. It emphasizes the importance of fine-tuning hyperparameters to achieve optimal results in machine learning models.

2 a)

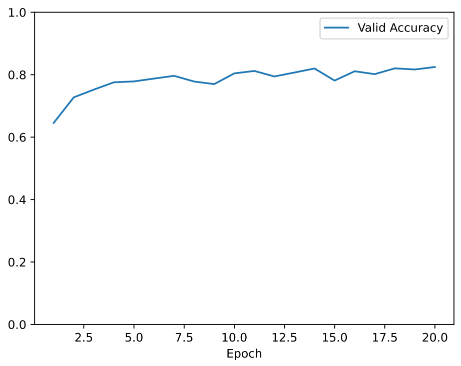
This exercise is all about tuning the batch size and seeing it's impact in the mlp (Multi layer perceptron), the batch sizes experimented with were 16 and 1024.



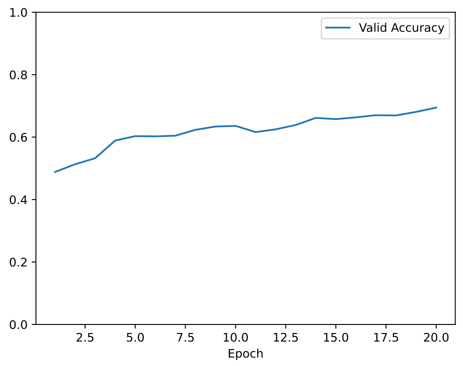
(a) Training and validation loss (batch 16)



(b) Training and validation loss (batch 1024)



(c) Validation accuracy (batch 16)



(d) Validation accuracy (batch 1024)

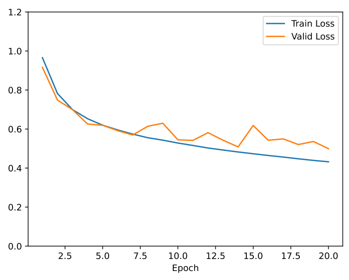
Figure 6: MLP Training Loss and Validation Accuracy with different batch sizes

Batch Size	Test Accuracy	Training Time (s)
16	0.7524	47.44
1024	0.7240	14.53

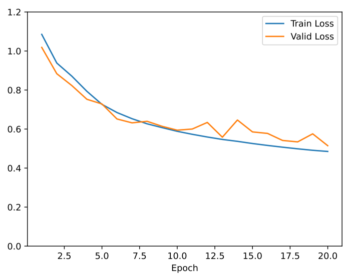
Table 2: Impact of Batch Size on Model Performance

As we can see from the plots, it is evident that the model with batch size 16 is better in terms of overall prediction, it has lower training loss between epochs and higher validation accuracy, leading to a better test accuracy, so we can conclude that it is a better predictive model. However, using batch size of 1024, the training time is reduced by a lot because of the distribution of the computational workload across a larger number of samples. In this case the batch size of 1024 clearly smooths out important features that should be considered and trained on.

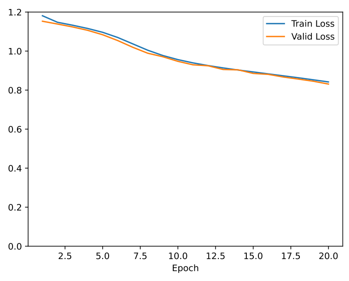
2 b)



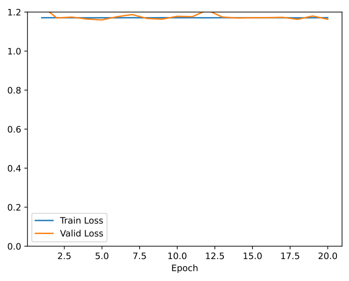
(a) Training and validation loss ($\eta = 0.1$)



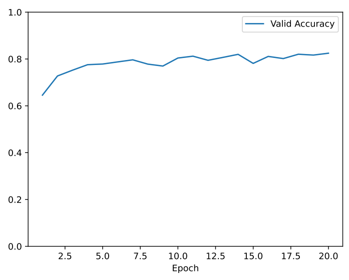
(b) Training and validation loss ($\eta = 0.01$)



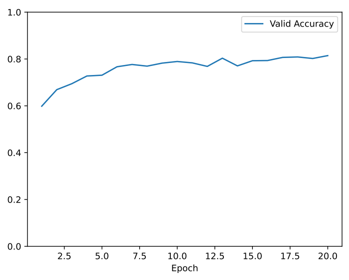
(c) Training and validation loss ($\eta = 0.001$)



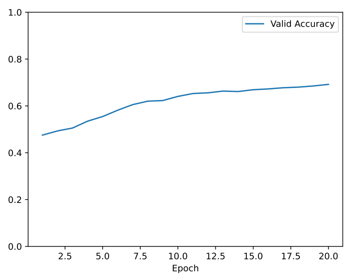
(d) Training and validation loss ($\eta = 1$)



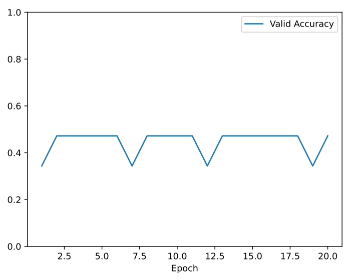
(e) Validation accuracy ($\eta = 0.1$)



(f) Validation accuracy ($\eta = 0.01$)



(g) Validation accuracy ($\eta = 0.001$)



(h) Validation accuracy ($\eta = 1$)

Figure 7: MLP Training Loss and Validation Accuracy with different learning rates

In our experiments, we investigated the impact of different learning rates (η) on the performance of the model. The test accuracies obtained with varying learning rates are summarized in Table 3.

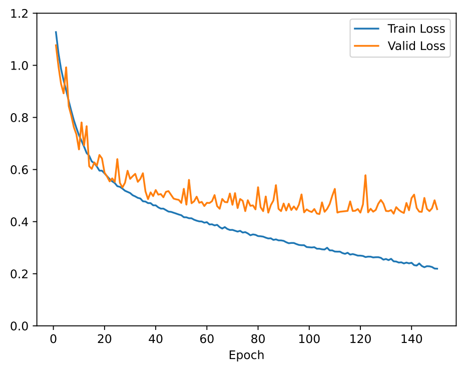
Learning Rate (η)	Test Accuracy
1	0.4726
0.1	0.7524
0.01	0.7561
0.001	0.7108

Table 3: Impact of Learning Rate on Model Performance

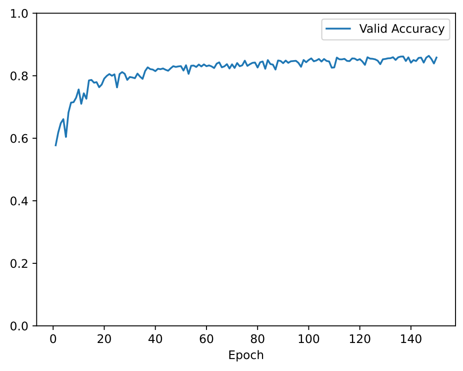
From the results, it is evident that the choice of learning rate significantly influences the model's ability to generalize to unseen data. A learning rate of 0.01 yielded the highest test accuracy of 0.7561, indicating a good balance between convergence speed and avoiding overshooting. However, a learning rate of 1 resulted in a lower accuracy, suggesting that the model diverged due to excessively large steps during training. Smaller learning rates, such as 0.1 and 0.001, also demonstrated reasonable performance, showcasing the sensitivity of the model to this hyperparameter.

In conclusion, careful tuning of the learning rate is crucial for achieving optimal performance, as excessively high or low values can impede convergence or hinder the model's ability to learn complex patterns within the data.

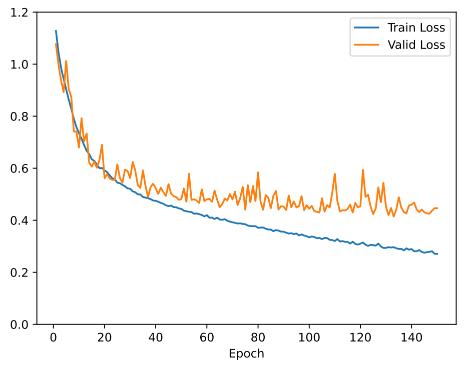
2 c)



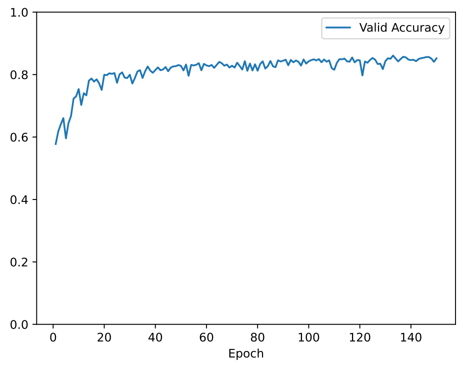
(a) Training Loss (No L2, No Dropout)



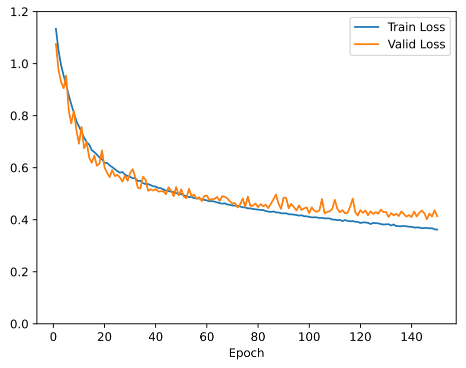
(b) Validation Accuracy (No L2, No Dropout)



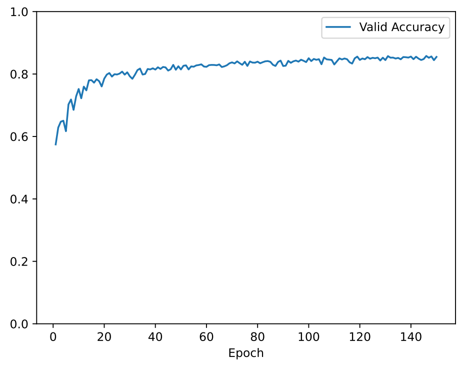
(c) Training Loss (L2 Regularization)



(d) Validation Accuracy (L2 Regularization)



(e) Training Loss (Dropout)



(f) Validation Accuracy (Dropout)

Figure 8: MLP Training Loss and Validation Accuracy with different regularization techniques

In our experimentation, we explored the impact of various regularization techniques on the model's performance. The test accuracies obtained with different regularization strategies are summarized in Table 4.

Regularization Technique	Test Accuracy
Regular	0.7505
L2 regularization	0.7618
Dropout	0.7864

Table 4: Impact of Regularization Techniques on Model Performance

From the results, it is evident that incorporating regularization techniques has a discernible effect on the model's generalization to unseen data. The introduction of L2 regularization, which penalizes large weights, resulted in a slightly higher test accuracy of 0.7618 compared to the non-regularized model (0.7505). This suggests that L2 regularization helped mitigate overfitting and improve the model's ability to generalize.

Surprisingly, dropout regularization surpassed both regular and L2 regularization techniques, achieving the highest test accuracy of 0.7864. This unexpected result emphasizes the effectiveness of dropout in preventing overfitting and highlights its role in creating a more robust model.

L2 regularization and dropout are distinct regularization techniques employed in neural network training to mitigate overfitting. L2 regularization, also known as weight decay, directly penalizes large weights by adding a term proportional to the sum of squared weights to the loss function. This continuous regularization method encourages smaller and more evenly distributed weights, thereby reducing model complexity. In contrast, dropout is a discrete regularization approach where random subsets of neurons are ignored during each training iteration, effectively introducing an ensemble effect by training different architectures. The randomness introduced by dropout helps prevent co-adaptation of neurons, fostering a more robust and generalized model. While L2 regularization encourages smaller weights throughout the training process, dropout dynamically disrupts the network's structure, providing a unique form of regularization through random dropout of neurons during each iteration.

In conclusion, our experiments underscore the importance of regularization in enhancing model performance. The choice of regularization technique may depend on the specific characteristics of the dataset and the architecture of the neural network. Regularization techniques play a vital role in preventing overfitting, improving generalization, and creating more robust models.

3 Question 3

1 a)

In order to prove that one single perceptron isn't generally able to compute the function mentioned in the statement, we'll follow the advice and prove with a counter-example by assigning specific values to the variables A, B and D.

Let's consider $D = 2$, $A = -1$ and $B = 1$. Thus,

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

Considering these values and the given function we have the following possible combinations:

x_1	x_2	$\sum_{i=1}^{2} x_i$	$f(x)$
-1	-1	-2	-1
-1	$+1$	0	$+1$
$+1$	-1	0	$+1$
$+1$	$+1$	2	-1

Table 5: $f(x)$ for various x_i values.

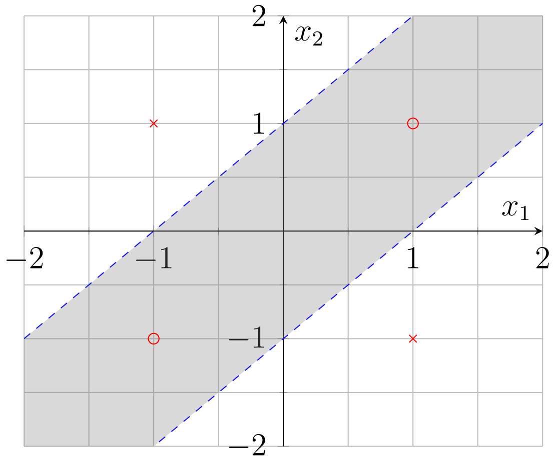


Figure 9: One possible data separation.

Let's recall the equation of a single perceptron:

$$f(x) = \text{sign}(w_1x_1 + w_2x_2 + b)$$

Which in turn may be simplified to the following:

$$f(x) = \text{sign}(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

By analyzing closely this equation we conclude that the decision boundary for a perceptron is a **straight line**. This happens because a perceptron is a linear combination of inputs with a threshold applied meaning that a single perceptron is **only able to solve linearly separable problems**.

Thus, in this specific case for any combination of weights and bias we won't be able to find a single perceptron that correctly classifies all 4 possible inputs based on the given function as this is a **non-linearly separable problem**. Visually this is coherent since by looking at Figure 9 it's easy to understand that we would have to draw at least two straight lines in order to separate the data.

Therefore, a single perceptron is not sufficient to generally represent this function.

1 b)

The multilayer perceptron (MLP) described in the statement has $x_{D \in \mathbb{N}}$ boolean inputs, only one hidden layer with two units and a output layer with only one unit which means that the network may be represented as it is in the image below.

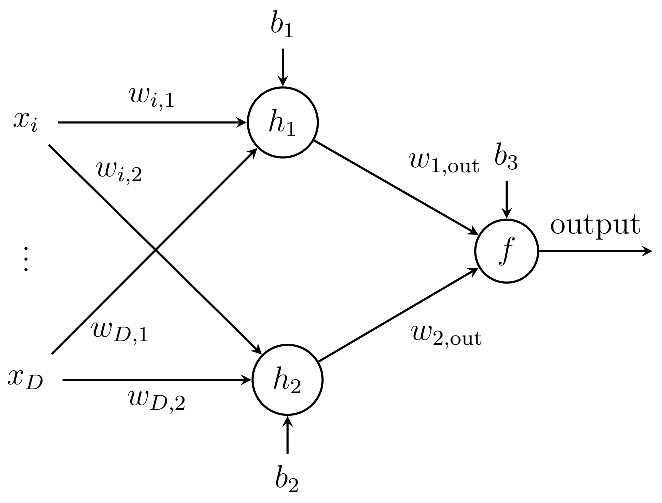


Figure 10: Reduced Height MLP with Ellipsis

Each of these units (perceptrons) is expected to have an hard threshold activation function $g(z)$, which corresponds to the following function:

$$g(z) = \text{sign}(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Now that we understand what's being asked in the statement let's begin implementing the function $f(x)$ through this specific MLP. In our problem we have 3 different regions:

A) Less than the value of A (class -1).

B) Between the value of A and B, inclusive (class 1).

C) Bigger than the value of B (class -1).

A priori...

From now on we will refer to k multiple times. For the records, $k \in \mathbb{N} \setminus \{0\}$.
The values with a '*' will be discussed afterwards in the section of robustness.

Hidden Unit 1 (h_1)**:**

With this perceptron we will classify the data which belongs to region A). In this unit we should fire output -1 if the sum of the inputs is less than A, in other words,

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

Considering $g(z)$, if we want the perceptron to output -1 we have to follow the equation:

$$w_{i,1} \cdot x_i + b_1 < 0 \quad \Leftrightarrow \quad w_{i,1} \cdot x_i < -b_1$$

So if we want the desired output when the sum of the inputs are less than A we must:

- Set all weights $w_{i,1}$ to k.
- Assign the bias b_1 the value -$k\dot{(}A+1)^*$.

Hidden Unit 2 (h_2):

With this perceptron we will classify the data which belongs to region C). In this unit we should fire output -1 if the sum of the inputs is bigger than B, in other words,

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

Considering $g(z)$, if we want the perceptron to output -1 we have to follow the equation:

$$w_{i,2} \cdot x_i + b_2 < 0 \quad \Leftrightarrow \quad b_2 < -w_{i,2} \cdot x_i$$

So if we want the desired output when the sum of the inputs are greater than B we must:

- Set all weights $w_{i,2}$ to -k.
- Assign the bias b_2 the value $k\dot{(}B+1)^*$.

Output unit (f):

With this perceptron we will classify the data which belongs to region B). In this unit we should fire output 1 if both outputs of the hidden units (z_1 and z_2) are 1 and output -1 otherwise. In other words,

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

Considering $g(z)$, if we want the perceptron to output 1 when $z_2 = z_1 = 1$ we have to follow the equation:

$$w_{1,out} \cdot z_1 + w_{2,out} \cdot z_2 + b_3 \geq 0 \quad \Leftrightarrow \quad w_{1,out} \cdot z_1 + w_{2,out} \cdot z_2 \geq -b_3$$

So if we want the desired output when the sum of the inputs belong to $[A, B]$ we must:

- Set the weights $w_{1,out}$ and $w_{2,out}$ to k.
- Assign the bias b_3 the value -k.

On a side note, this the implementation of the logic gate AND.

Note

The use of k is due to the fact that the perceptron behaves the same way for any linear combination of the inputs. This means that as long as the weights and biases grow/decrease in the same proportion (k) the output won't be affected. The simpler case is when $k = 1$.

Computed Function

$$f(x) = \text{sign}\left(k \cdot \text{sign}\left(k \cdot x_i - k(A+1)\right) + k \cdot \text{sign}\left(-k \cdot x_i + k(B+1)\right) - k\right)$$

Robustness to infinitesimal perturbations (*)

Above we provided a solution that correctly implements the function $f(x)$ and is robust to infinitesimal perturbations on the input. This means that the biases and weights are defined in such a way that a small change in the input doesn't cross the decision boundary. Since we are working with hard threshold activation functions this means that the decision boundary is when the input of the activation function is 0.

What this actually means is that when the input is exactly 0, an infinitesimal change in the input could flip the sign of the output. In order to prevent this we should ensure that the input is never exactly 0 when accounting the valid constrains of the problem.

As the input belongs to $\{-1, 1\}$ a change in the input results in a change of 2 in the output inasmuch as changing x_i from -1 to +1 for example will result in a change of 2 of the sum. Therefore, to accommodate these eventual infinitesimal changes, the bias of h_1 is set just below A and the bias of h_2 is set just above B. Consequently, defining $b_1 = -k(A+1)$ and $b_2 = k(B+1)$ implicits that in order to have an output change, we must have a change of 2 in one of the inputs which isn't an infinitesimal amount. Thus, this biases' values ensure indeed that infinitesimal perturbations in the input doesn't change the output, enhancing the robustness of the network.

1 c)

In this exercise the idea behind is the same as the previous one but here the hidden units will have a ReLU activation function which will change some details in the network as we will see.

$$\text{ReLU}(z) = \max(0, z)$$

Since we are working with integers the output of the units of the hidden layers are either 0 or some positive integer. This idea have to do with the definition of the ReLU activation function.

$$\text{ReLU}(z) = \begin{cases} z & \text{if } z > 0,\ z \in \mathbb{Z}, \\ 0 & \text{if } z \leq 0. \end{cases}$$

This change may seem minor but it actually

Hidden Unit 1 (h_1)**:**

With this perceptron we will classify the data which belongs to region A). In this unit we should fire now the output 0 if the sum of the inputs is less than A, in other words,

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

Considering the activation function ReLU(z), if we want the perceptron to output 0 we have follow the equation:

$$w_{i,1} \cdot x_i + b_1 \leq 0 \quad \Leftrightarrow \quad w_{i,1} \cdot x_i \leq -b_1$$

So if we want the desired output when the sum of the inputs are less than A we must:

- Set all weights $w_{i,1}$ to k.
- Assign the bias b_1 the value -$k\dot{(}A+1)$.

Hidden Unit 2 (h_2)**:**

With this perceptron we will classify the data which belongs to region C). In this unit we should fire output 0 if the sum of the inputs is bigger than B, in other words,

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

Considering ReLU(z), if we want the perceptron to output 0 we have to follow the equation:

$$w_{i,2} \cdot x_i + b_2 \leq 0 \quad \Leftrightarrow \quad b_2 \leq -w_{i,2} \cdot x_i$$

So if we want the desired output when the sum of the inputs are greater than B we must:

- Set all weights $w_{i,2}$ to -k.
- Assign the bias b_2 the value $k\dot{(}B+1)$.

Note

Although the activation function have changed in this exercise, the bias of the hidden units were kept the same since we still wish to prevent the output of these hidden units from being affected by small changes in the input. This also ensures that the output is only 0 when the defined constrains are validated.

Output unit (f):

With this perceptron we will classify the data which belongs to region B). In this unit we should fire output 1 if both outputs of the hidden units (z_1 and z_2) are bigger than 0 (1, 2, 3, ...) and output -1 otherwise. In other words,

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

Considering the hard threshold activation function (defined in the previous exercise), if we want the perceptron to output 1 when z_2 and $z_1 > 0$ we have to follow the equation:

$$w_{1,out} \cdot z_1 + w_{2,out} \cdot z_2 + b_3 \geq 0 \quad \Leftrightarrow \quad w_{1,out} \cdot z_1 + w_{2,out} \cdot z_2 \geq -b_3$$

So if we want the desired output when the sum of the inputs belong to $[A, B]$ we must:

- Set the weights $w_{1,out}$ and $w_{2,out}$ to k.
- Assign the bias b_3 the value -2k.

Computed Function

$$f(x) = \text{sign}\left(k \cdot \text{ReLU}\left(k \cdot x_i - k(A+1)\right) + k \cdot \text{ReLU}\left(-k \cdot x_i + k(B+1)\right) - 2k\right)$$

The value of the bias of the output layer b_3 is the only value that have changed compared to the previous exercise. As we have stated previously the output of the output unit should only be $+1$ when the outputs of both hidden layers are positive. As both of the outputs of the hidden layer (z_1 and z_2) contribute k to the sum, subtracting $-2k$ will ensure that the sum that inputs the output unit will be only equal or greater than 0 when both of the inputs are positive and consequently output $+1$.

4 Contribution of each member

In this homework we separated the 3 questions amongst the two members like this:

- Miguel Luís Rente Lourenço (100044) - Questions 1 and 2
- Tiago Mendes Alves Santos Costa (100094) - Question 3 and Question 1.2 b)

This was our initial separation, even though this wasn't linearly separable (both of us interfered in each one of the questions), the report was elaborated and revised by both of us. In conclusion, the amount of work done turned out to be a 50-50 split.