Homework 1 - Deep Learning (Dei)

Rafael Sargento, Diogo Miranda

December 2024

Question 1

Question 1.1a

1a - The implementation of the Perceptron model is in the `hw1-q1.py` file. The model was trained for 100 epochs, and the results are as follows:

- **Training Time:** 0 minutes and 42 seconds
- **Final Test Accuracy:** 0.3743

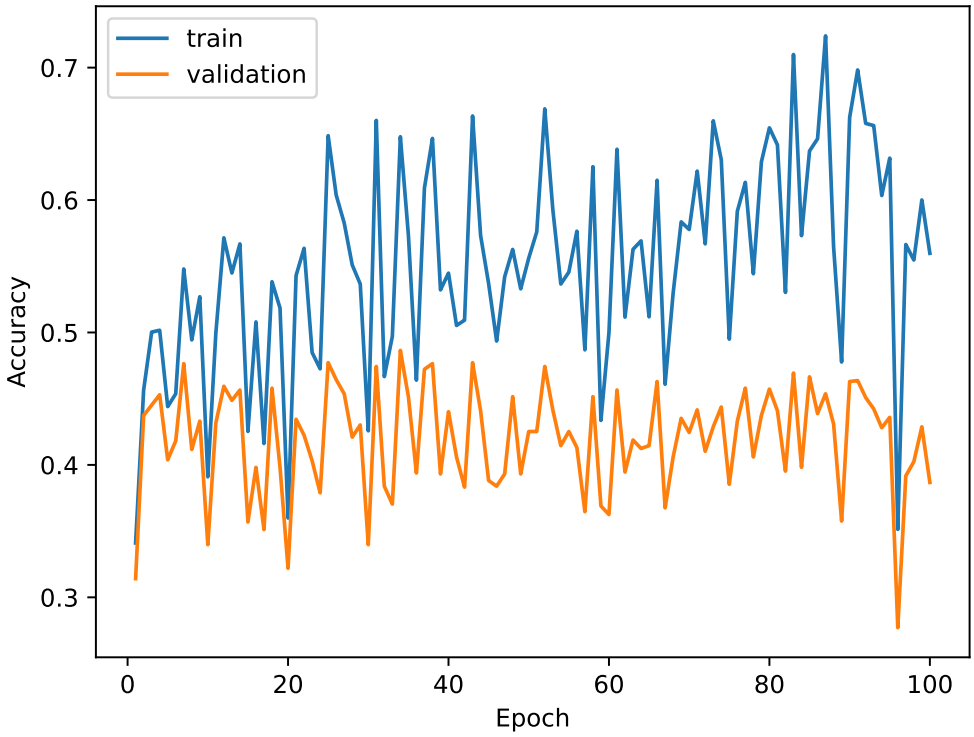


Figure 1: Perceptron Model Training and Validation Accuracies

Question 1.2a

2a - The implementation of the Logistic Regression without ℓ_2 penalty model is in the `hw1-q1.py` file. The model was trained for 100 epochs, and the results are as follows:

- **Training Time:** 2 minutes and 29 seconds
- **Final Test Accuracy:** 0.47833

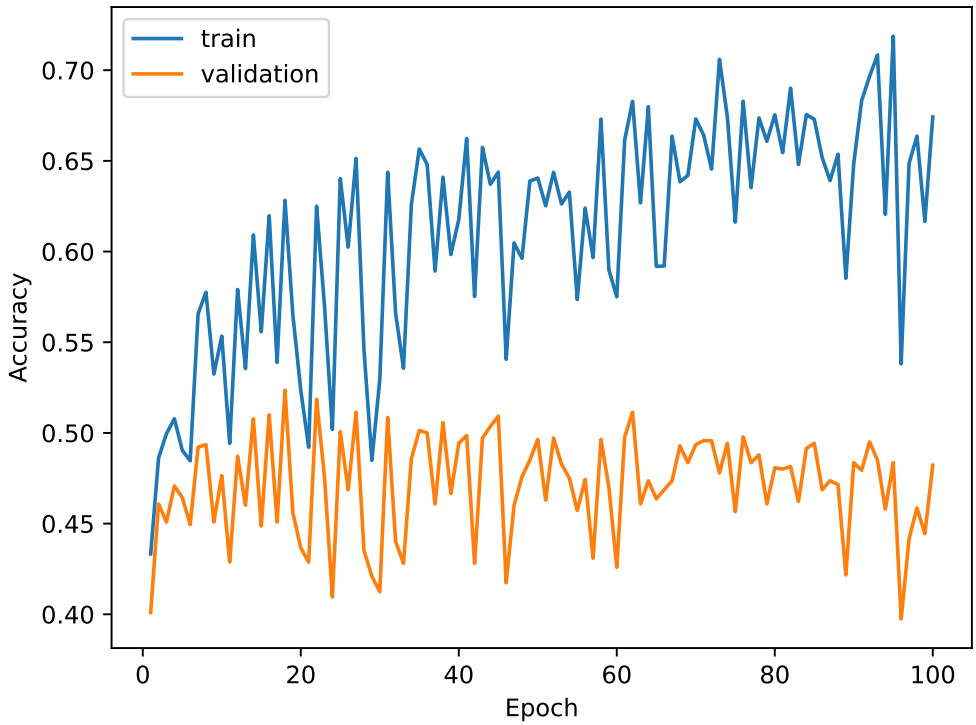


Figure 2: Logistic Regression without ℓ_2 Penalty - Training and Validation Accuracies

Question 1.2b

2b - The implementation of the Logistic Regression with ℓ_2 penalty model is in the `hw1-q1.py` file. The model was trained for 100 epochs, and the results are as follows:

- **Training Time:** 2 minutes and 30 seconds
- **Final Test Accuracy:** 0.5113
- **Validation Accuracy:** 0.5000

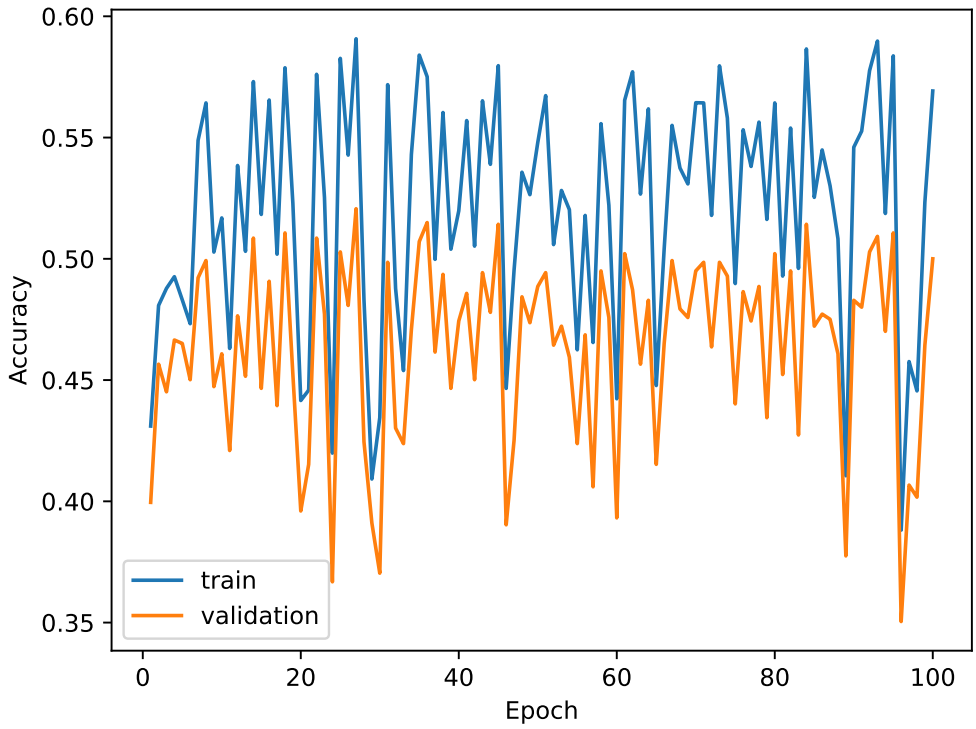
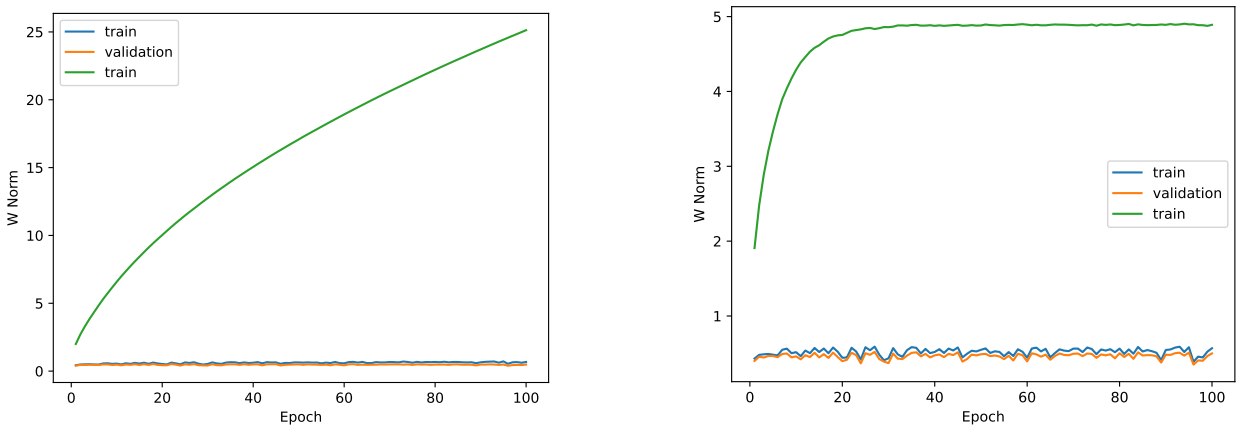


Figure 3: Logistic Regression with ℓ_2 Penalty - Training and Validation Accuracies

Comment: Without regularization, the model achieves higher training accuracy but lower validation accuracy, indicating overfitting. With ℓ_2 regularization, training accuracy decreases, but validation and test accuracies improve slightly, suggesting better generalization.

Question 1.2c

2c - The plots for the ℓ_2-norm of the weights of both the non-regularized and regularized versions of the logistic regression classifiers as a function of the number of epochs are as follows:



(a) ℓ_2-norm of W without ℓ_2-penalty

(b) ℓ_2-norm of W with ℓ_2-penalty

Figure 4: Comparison of Weight Norms for Logistic Regression Models

Comment: In model a) (without ℓ_2 penalty), the weight norm grows significantly larger, reaching around 25, reflecting a high tendency to overfit. In contrast, b) (with ℓ_2 penalty) constrains the weight norm to below 5, leading to better regularization and more stable generalization.

Question 1.2d

2d - Using ℓ_1 regularization, the optimization introduces a penalty that encourages sparsity in the weight vector. This means many weights would become exactly zero, effectively excluding less relevant features from the model. This property is useful for feature selection in high-dimensional datasets. On the other hand, ℓ_2 regularization does not set weights to zero but instead penalizes their magnitude, distributing the reductions more evenly. This results in a model where all features contribute, but their impact is scaled down, leading to smoother solutions without feature elimination.

Question 1.3a

3a - The implementation of the MLP model is in the `hw1-q1.py` file. The model was trained for 20 epochs, and the results are as follows:

- **Training Time:** 10 minutes and 6 seconds
- **Final Test Accuracy:** 0.52733

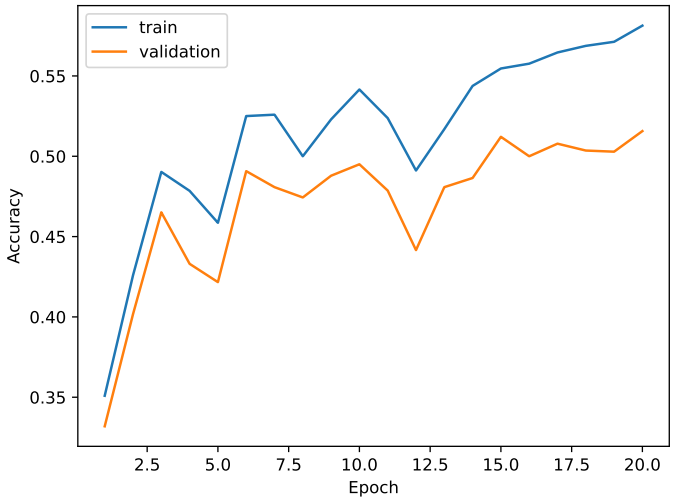


Figure 5: MLP Model - Training and Validation Accuracies

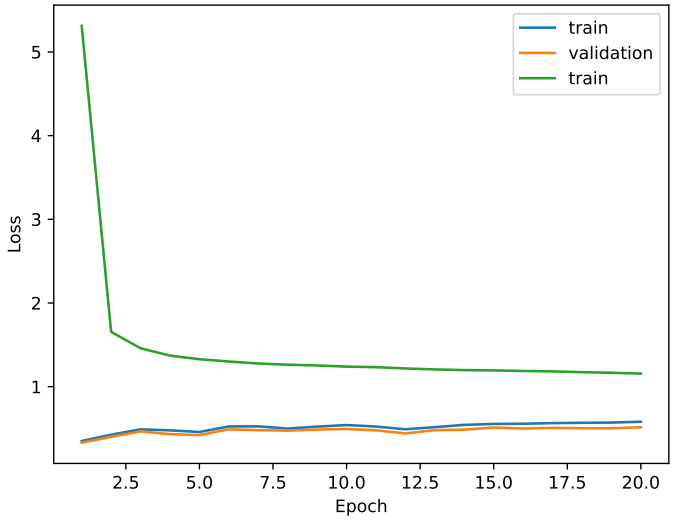


Figure 6: MLP Model - Training Loss

Question 2

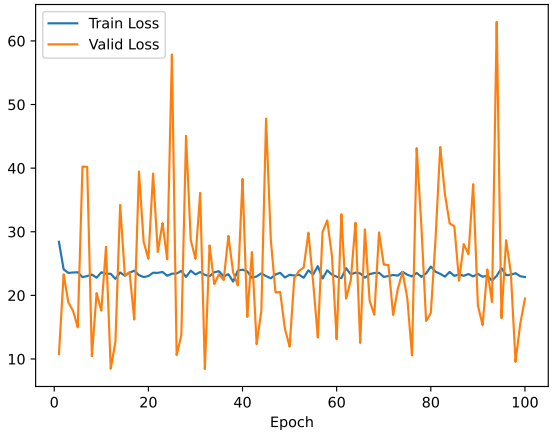
Question 2.1

The implementation to produce the requested plots is in the `hw1-q2.py` file.

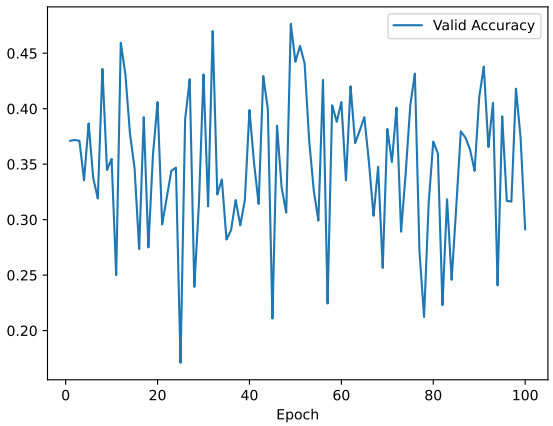
0.1 Learning Rate

The model, with a learning rate of 0.1, appears to "learn" too quickly, leading to significant fluctuations in both training and validation losses. This instability likely results from the model adjusting too rapidly to the data, which may cause it to overfit specific samples or miss broader patterns needed for generalization. The graph for the validation accuracy also doesn't seem to converge to any specific value.

- **Training Time:** 10 seconds
- **Final Test Accuracy:** 0.2757
- **Validation Accuracy:** 0.2913



(a) Training Loss (Learning Rate = 0.1)



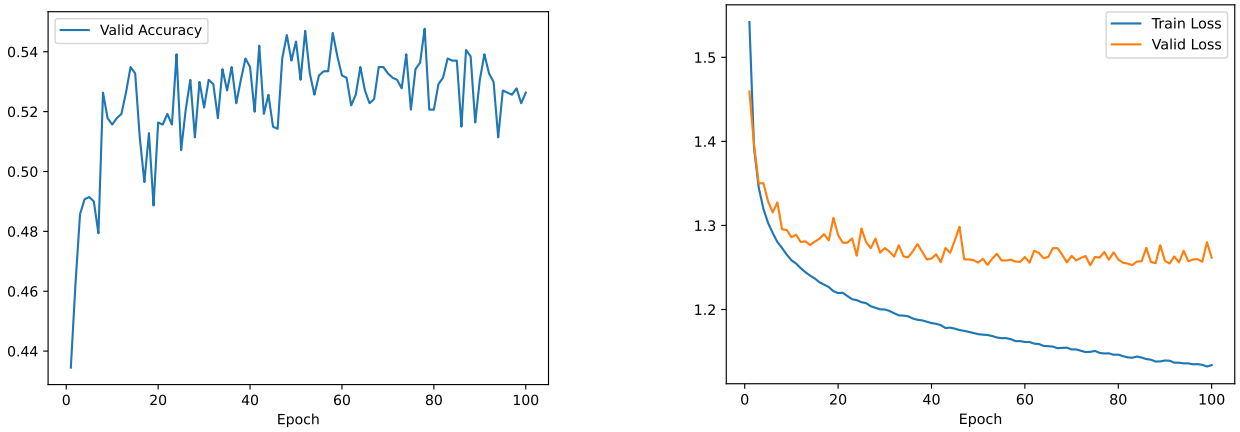
(b) Validation Accuracy (Learning Rate = 0.1)

Figure 7: Logistic Regression Training Loss and Validation Accuracy for 0.1 Learning Rate

0.001 Learning Rate (Highest Validation Accuracy)

The model, with a learning rate of 0.001, shows a smoother but gradual decrease in loss over 100 epochs. This suggests that the model is learning at a steadier pace, reducing the risk of overfitting early in the training process. However, the relatively slow rate of improvement might also indicate that the learning rate could be too conservative.

- **Training Time:** 11 seconds
- **Final Test Accuracy:** 0.5247
- **Validation Accuracy:** 0.5264



(a) Validation Accuracy (Learning Rate = 0.001)

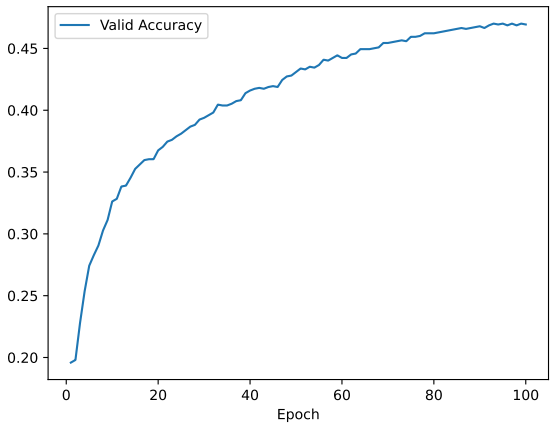
(b) Training Loss (Learning Rate = 0.001)

Figure 8: Logistic Regression Training Loss and Validation Accuracy for 0.001 Learning Rate

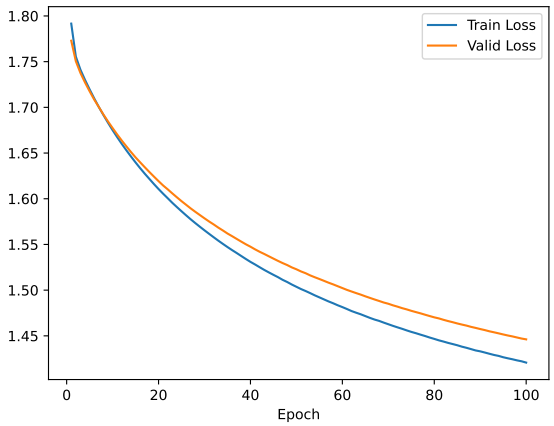
0.00001 Learning Rate

The model, with a learning rate of 1e-05, shows a very gradual decrease both in training and validation losses over the 100 epochs. Both losses fail to converge at a number, which suggests that the learning rate might be too low, resulting in this slow convergence and potentially underfitting the data. A possible solution to this would be train using additional epochs.

- **Training Time:** 10 seconds
- **Final Test Accuracy:** 0.4623
- **Validation Accuracy:** 0.4694



(a) Validation Accuracy (Learning Rate = 1e-05)



(b) Training Loss (Learning Rate = 1e-05)

Figure 9: Logistic Regression Training Loss and Validation Accuracy for 1e-05 Learning Rate

Conclusion:

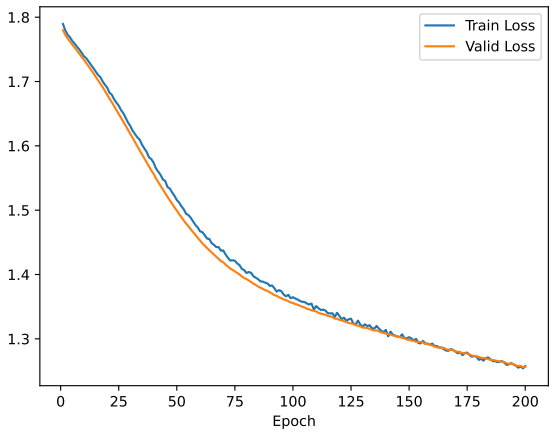
Learning rate that achieved highest validation accuracy: **0.001**

Comment:

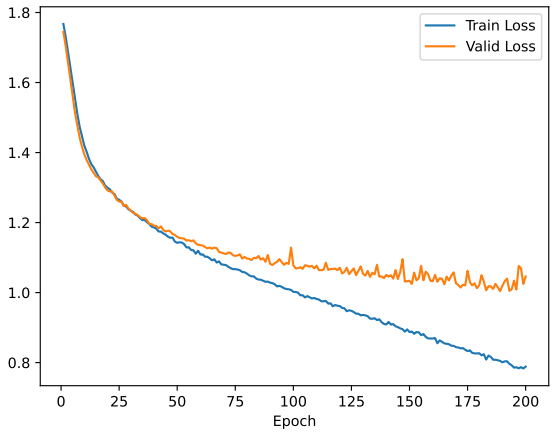
Choosing an optimal learning rate is crucial as it directly affects how well and how quickly a model can converge to a low error state while avoiding the pitfalls of overfitting or underfitting. The moderate learning rate of 0.001 seems to provide the best results, allowing sufficient learning without the instability seen with higher rates or the slow convergence seen with lower rates.

Question 2.2a

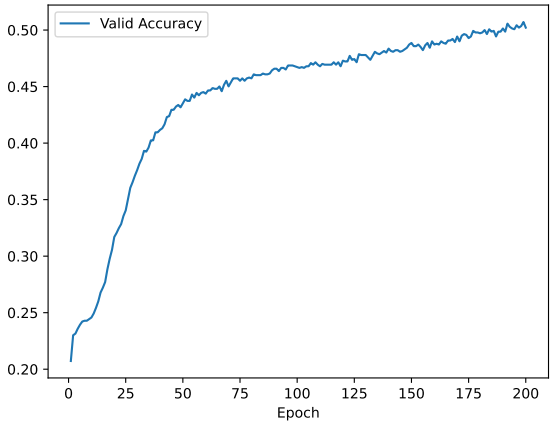
The implementation to produce the requested plots is in the `hw1-q2.py` file.
The plots requested are as follows:



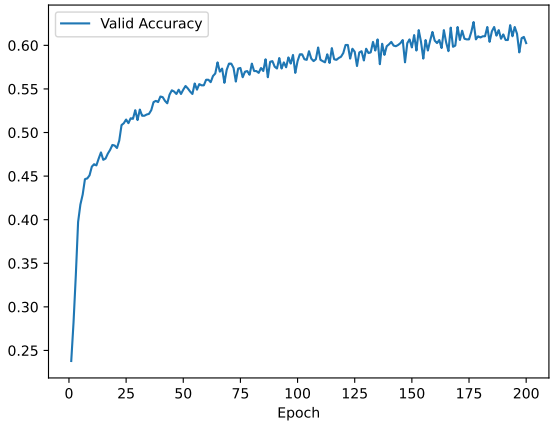
(a) Training Loss (Batch Size 512)



(b) Training Loss (Batch Size 64)



(c) Validation Accuracy (Batch Size 512)



(d) Validation Accuracy (Batch Size 64)

Figure 10: MLP Model Performance with Different Batch Sizes

Comment:

The choice of batch size can significantly affect the training dynamics of a model, smaller batch sizes, such as 64 often provide more noise in the gradient estimation, which can help escape local minima and explore the loss landscape more effectively, potentially leading to better generalization as seen here. On the other hand, larger batch sizes like 512 provide more stable but less diverse gradient estimates per update, which might slow the learning process or trap the model in suboptimal minima, as might be indicated by the higher validation losses and lower accuracy. Also, smaller batches are typically processed faster per batch but require more updates (resulting in higher training time), thus potentially enhancing the model's ability to generalize from the training data. Larger batches process more data at once, which can be computationally efficient but may reduce the frequency and variability of updates (lesser training time), impacting the model's ability to learn significant patterns in the data.

Default Hyperparameters:

- **Training Time:** 1 minutes and 36 seconds
- **Final Test Accuracy:** 0.6003
- **Validation Accuracy:** 0.6026

Batch 512:

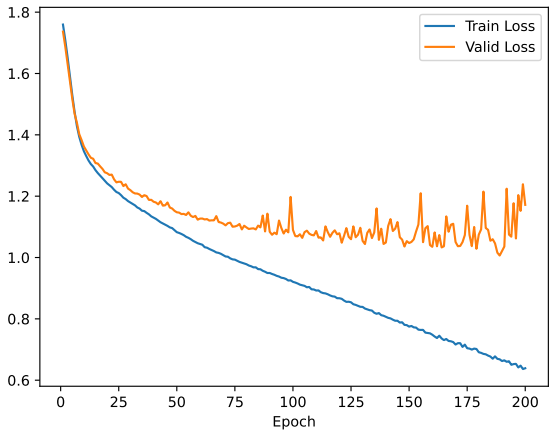
- **Training Time:** 53 seconds
- **Final Test Accuracy:** 0.5187
- **Validation Accuracy:** 0.5021

Question 2.2b

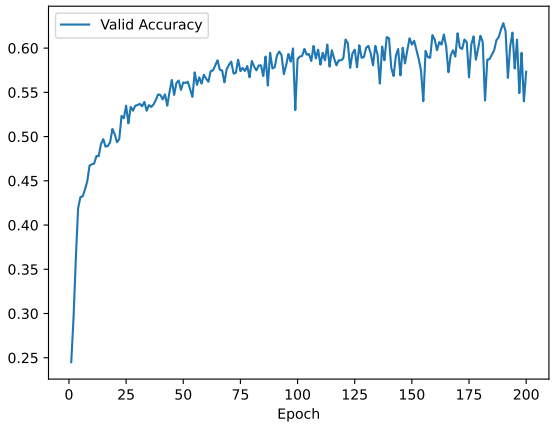
The implementation to produce the requested plots is in the `hw1-q2.py` file.
The plots requested are as follows:

0.01 Dropout:

- **Training Time:** 1 minutes and 35 seconds
- **Final Test Accuracy:** 0.5717
- **Validation Accuracy:** 0.5734



(a) Training Loss



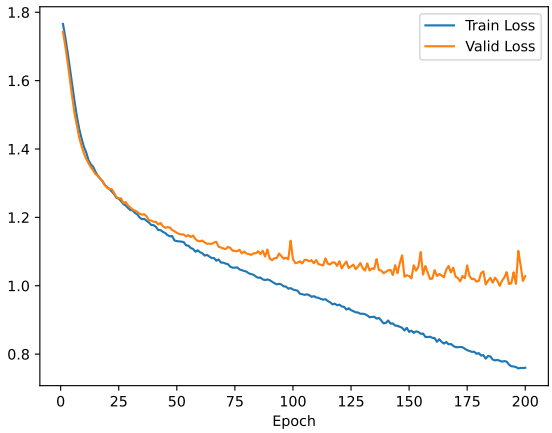
(b) Validation Accuracy

Figure 11: Dropout Rate 0.01 - Minimal Regularization

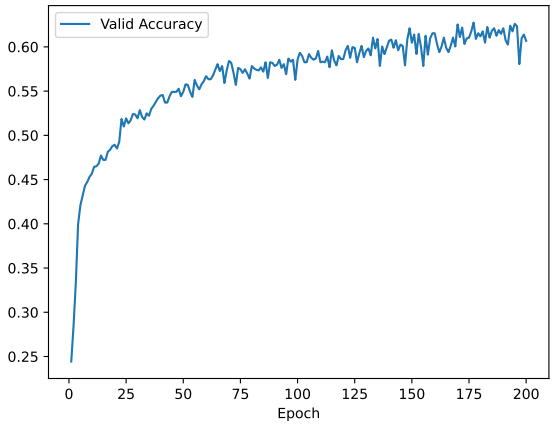
The graph demonstrates a smoother decline in loss values, reaching lower levels for both training and validation losses. This indicates that a lower dropout rate allows the model to utilize more of its capacity during training. However, towards the end of the training epochs, an increase in validation loss is observed. This uptick suggests that the model may be beginning to overfit, as it gets better at predicting the training data at the expense of its generalization to new, unseen data. This graph indicates a steady increase in accuracy, peaking at around 0.6. The low dropout rate likely supports better network learning and generalization, resulting in this high validation accuracy.

0.25 Dropout:

- **Training Time:** 1 minutes and 33 seconds
- **Final Test Accuracy:** 0.6050
- **Validation Accuracy:** 0.6068



(a) Training Loss



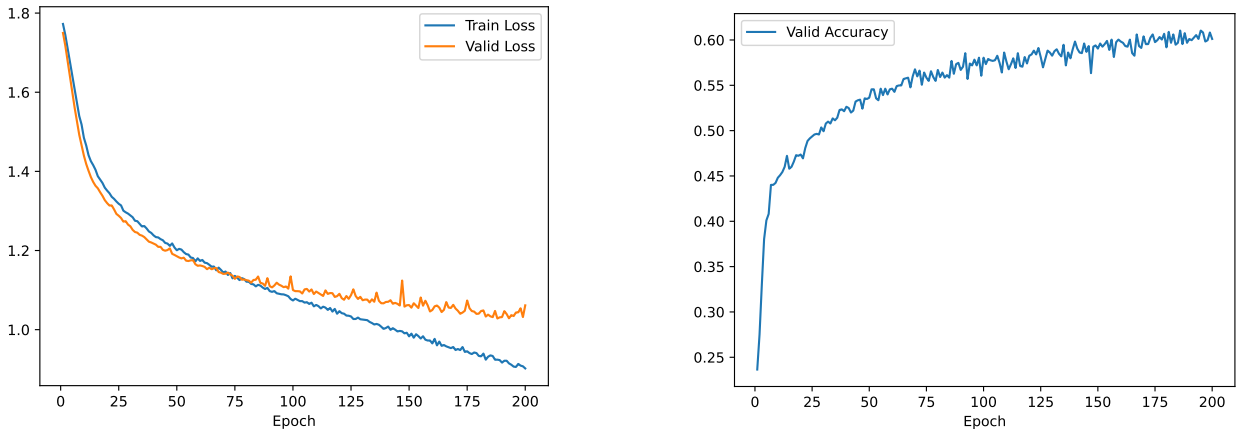
(b) Validation Accuracy

Figure 12: Dropout Rate 0.25 - Moderate Regularization

The graph shows a steady decrease in both training and validation losses, indicating effective learning and generalization with a moderate dropout rate. This rate allows for a balance between allowing the model to learn patterns and preventing overfitting by randomly disabling an higher portion of neurons during training. Although the validation loss does not reach as low as with the lower dropout rate, it maintains a consistent decline without significant spikes, in contrary to what seen previously, suggesting that the model retains good generalization capabilities across the training epochs. This moderate level of dropout helps in avoiding overfitting while still leveraging a substantial amount of the network's capacity for learning.While accuracy still improves, the trend is similar to lower dropout rates, peaking around 0.6. This shows that while the model can still generalize, the higher dropout rate might be limiting the depth of patterns the network can learn.

0.5 Dropout:

- **Training Time:** 1 minutes and 40 seconds
- **Final Test Accuracy:** 0.5883
- **Validation Accuracy:** 0.6011



(a) Training Loss

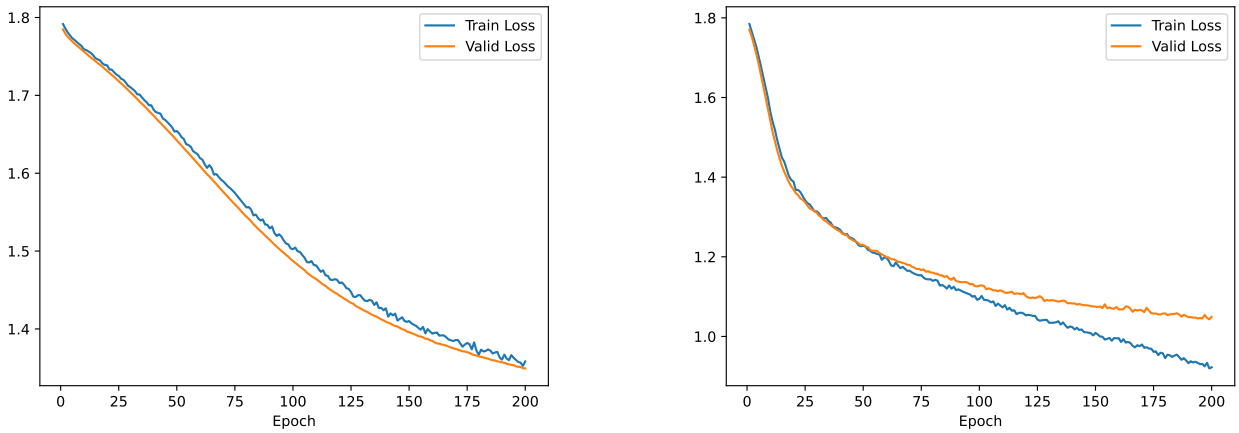
(b) Validation Accuracy

Figure 13: Dropout Rate 0.5 - Strong Regularization

The graph for the model trained with a dropout rate of 0.5 illustrates a slower decline in both training and validation losses, which indicates that the higher dropout rate might be too restrictive, possibly hindering the model's ability to effectively learn more complex patterns. Despite this, the validation loss shows relative stability, suggesting that the model maintains generalization without significant signs of overfitting. However, the relatively higher and stable validation loss throughout training indicates that while the model is robust against overfitting, it may not be learning as efficiently or deeply as possible due to the high rate of neuron dropout during training.While accuracy still improves, the trend is similar to lower dropout rates, peaking around 0.6. This shows that while the model can still generalize, the higher dropout rate might be limiting the depth of patterns the network can learn.

Question 2.2c

The implementation to produce the requested plots is in the `hw1-q2.py` file.
The plots requested are as follows:



(a) Training Loss (Momentum = 0.0)

(b) Training Loss (Momentum = 0.9)

Figure 14: MLP Loss for Different Momentum Values

Comment:

The results from training two models with different momentum values (0.0 and 0.9) clearly illustrate the impact of using momentum in the optimization process. In the model with 0.0 momentum, the loss graph shows a gradual but consistent decrease, however, the losses remain relatively higher throughout training compared to the model with momentum, suggesting a slower convergence rate. The validation accuracy shows improvement over the epochs but plateaus around the 0.45 mark, indicating that the model struggles to improve further without the assistance of momentum.

On the other hand, the model with 0.9 momentum demonstrates faster and more significant decreases in both training and validation losses, suggesting that the high momentum enables the model to converge more quickly and effectively, potentially overcoming challenges such as navigating out of local minimas. The validation accuracy for this model shows a clear upward trend, reaching higher levels up to 0.60, surpassing the performance of the model without momentum.

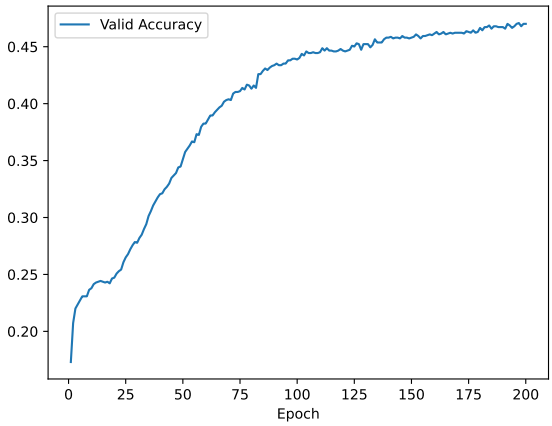
The use of momentum, particularly a high value like 0.9, appears to have a substantial positive effect on the model's learning dynamics and its ability to generalize. This is evident from both the faster reduction in loss values and the higher final validation accuracy.

No Momentum (0.0):

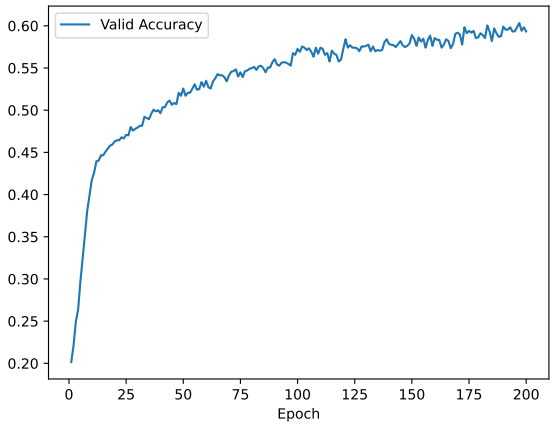
- **Training Time:** 1 minutes and 36 seconds
- **Final Test Accuracy:** 0.4887
- **Validation Accuracy:** 0.4701

Momentum of 0.9:

- **Training Time:** 50 seconds
- **Final Test Accuracy:** 0.5980
- **Validation Accuracy:** 0.5933



(a) Validation Accuracy (Momentum = 0.0)



(b) Validation Accuracy (Momentum = 0.9)

Figure 15: MLP Validation Accuracy for Different Momentum Values

Question 3

All the solutions for the pen and paper exercises are in the PDF file included in the zip submitted.

(2) $h = A_\theta \phi(x) = [A_1\ A_2\ A_3] \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix} =$

$= A_1\phi_1 + A_2\phi_2 + A_3\phi_3$

$\frac{(D+1)(D+2)}{2}$

$= \frac{D^2+2D+D+2}{2}$

$= \frac{D^2+3D+2}{2}$

hi is a value from h
wi is a row W

$h_i = w_i x + b_i - (w_i x + b_i)^2$

$= w_i x - (w_i x)^2 - 2 w_i x b_i + b_i - b_i^2$

Example for $D=3$ $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$

$\phi_3^T = [x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2]$

$A_1 = b_i - b_i^2 \quad \therefore \phi_1^T = 1$

$A_2 = w_i - 2w_ib_i \quad \therefore \phi_2^T = x = [x_1, \dots x_D]$

$A_3 = -w_{ia}w_{ib} \quad \therefore \phi_3^T = x_a x_b$ for every combination of $1 \le a \le D$, $1 \le b \le D$

D weighted sum generated by the product of $A_3\ \phi_3$

$$\left(\begin{matrix} W: \\ (a\ b\ c) \end{matrix} \cdot \begin{matrix} x \\ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \end{matrix} \right)^2 = \left(a^2x^2 + b^2y^2 + 2\cdot(abxy) + c^2z^2 + 2\cdot(acxz) + 2\cdot(bcyz) \right)$$

(2) $\hat{y} = v^T h + v_0 \qquad h = A_\theta \phi(x)$

$\hat{y} = v^T A_\theta \phi(x) + v_0$

$c_\theta^T \phi(x) = v^T A_\theta \phi(x) + v_0$

$c_\theta^T = v^T A_\theta + v_0$

$c_\theta = (v^T A_\theta + v_0)^T$

$c_\theta = A_\theta^T v + v_0$

R: It's not a linear model with respect to the original parameters because A_θ has non linear relationships with the parameters and c_θ is defined with A_θ as shown above.

$1+D+\frac{D(D+1)}{2} = \frac{(D+1)(D+2)}{2}$

③
$$\begin{cases} c_1 = (b-b^2)v + v_0 \\ c_2 = (w-2wb)v \Rightarrow c_2 = w(1-2b)v \\ c_3 = w^T \text{diag}(v) w \end{cases}$$

$c = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$ → const, → linear, → $\frac{D(D+1)}{2}$

④ solve the system

$$A_3 = -w_a w_b \left[a_1^2, a_2^2, a_3^2, 2a_1a_3, 2a_1a_2, 2a_2a_3\right]$$

↑ how of A_3 assuming $w = (a_1 \; a_2 \; a_3)$

$c_3 = A_3^T v$

$c_3 = \text{vec}(Z)$

$Z = Q \,\text{diag}(\lambda) Q^T$

$w = Q\sqrt{\text{diag}(\lambda)}$

$Z = w^T \text{diag}(v) w$

Z

$H = ww^T$ → gives symetrical matrix $\begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix}$

$H' = w \,\text{diag}(v) w^T$

$\text{vech}(H') = c_3$

④
$$\begin{cases} v_0 = c_1 - (b-b^2)v \\ w = \frac{c_2}{(1-2b)v} \\ b = \frac{1 \pm \sqrt{1 - 4\frac{c_1 - v_0}{v}}}{2} \end{cases}$$

(4)

- X^TX is guaranteed to be invertible because we know that for any real matrix $\text{rank}(X^TX) = \text{rank}(X)$ and we know that X^TX is a matrix $M \times M$, $M = \frac{(D+1)(D+2)}{2}$ we also know that $\text{rank}(X^TX) = M$ so the matrix X^TX has full rank and so by the invertible matrix theorem, it is invertible.

- Although c_θ depends non linearly on the model parameters, the prediction $\hat{y} = c_\theta^T \phi(x)$ is linear for c_θ for a fixed feature map. So we can use the global minimizer:

$$c_\theta = (X^TX)^{-1}X^TY$$

- In typical neural networks when we use activation functions like ReLU, sigmoid, or tanh the loss function becomes non-convex, making global minimization intractable.

- In this problem the loss function our activation function ensures that the network behaves linearly in the feature space $\phi(x)$. The loss function remains convex with respect to c_θ and that guarantees a unique global minimum

In our group for Homework 1, Rafael Sargento worked on solving everything related to Question 1, while Diogo Miranda focused on Question 2. For Question 3, Diogo Miranda initially solved 3.1, Rafael Sargento handled 3.2 and 3.4, and both of us collaborated to solve 3.3. It is worth noting, however, that all answers to Question 3 were discussed as a group. This means that, although one person finalized each part, the entire process of arriving at the solutions was done collaboratively.