Deep Learning - DEI

Group 30: Alice Mota ist1102500 | Francisco Leitão ist103898

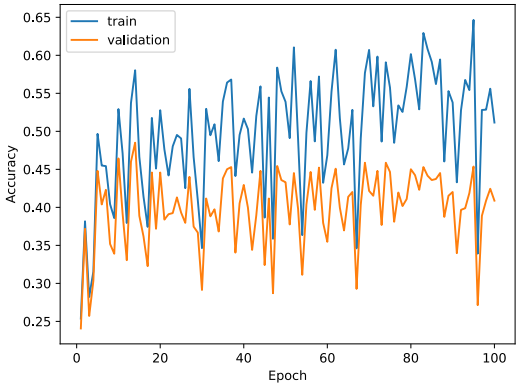
Contibution:

Question 1 was solved by Alice Mota, and Question 2 by Francisco Leitão. In regards to Question 3, both members contributed, with Alice having done 1. And 4., and Francisco 2. And 3.

Homework 1:

Question 1

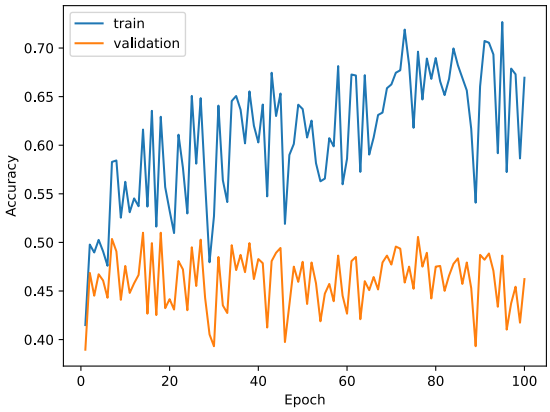
1. The performances, with accuracy, as the chosen metric were the following: 0.5116 on the training set, 0.4088 on the validation set and 0.4173 on the test set.
   The Perceptron model shows an upward trend in training accuracy. Both training and validation accuracy fluctuate significantly. The widening accuracy gap in later epochs highlights the model's difficulty in generalizing and its limitations with complex, non-linear patterns, emphasizing the need for more advanced algorithms for this task.



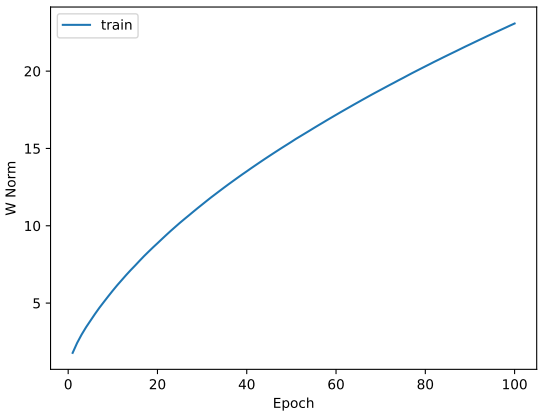
Plot 1 – Perceptron: train and validation accuracies in function of the epoch number

2.

a) The performances, with accuracy as the chosen metric, were the following: 0.6694 on the training set, 0.4623 on the validation set and 0.4597 on the test set. The training weight norm was 23.0806.

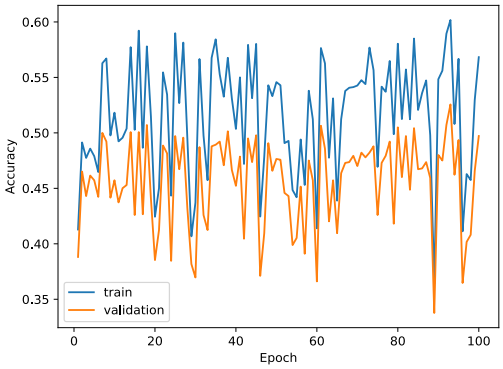


Plot 2 – Logistic Regression: train and validation accuracies in function of the epoch number

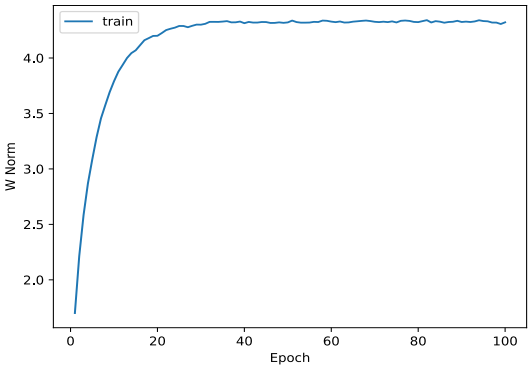


Plot 3 – Logistic Regression: Weigth norms in function of the epoch number

b) The performance with accuracy as the chosen metric were the following: 0.5683 on the training set, 0.4972 on the validation set and 0.5053 on the test set. The training weight norm was 4.3217.
Without regularization (Plot 2), the training accuracy has a more significant increase, surpassing validation accuracy, which initially improves, but starts to be further behind the training accuracy. With regularization (Plot 4), training accuracy grows more slowly and stays closer to validation accuracy, indicating better generalization, though both accuracies are slightly lower than in the non-regularized case. Regularization, particularly ℓ2, mitigates overfitting by penalizing large weights, achieving a balance between underfitting and overfitting for more robust performance on unseen data.



Plot 4 – Logistic Regression with L2 regularization: train and validation accuracies in function of the epoch number



Plot 5 – Logistic Regression with L2 regularization: weight norms in function of the epoch number

c) As we can see, In the non-regularized model (Plot 3)**, the** Weight norms are higher when compared to the regularized model (Plot 5), and steadily increase with epochs, reflecting unconstrained adjustments to minimize training loss. This focus on reducing training error often leads to overfitting, as larger weights make the model overly complex and less generalizable. In the L2-Regularized Models, the Weight norms remain smaller and stable due to the penalty term discouraging large weights. Regularization enforces simplicity, reducing overfitting and improving generalization.
In conclusion, L2-regularization balances model complexity and generalization by constraining weight growth, as evident from the stabilized norms and smoother trends in regularized models. This ensures better performance on unseen data when compared with non-regularized model.

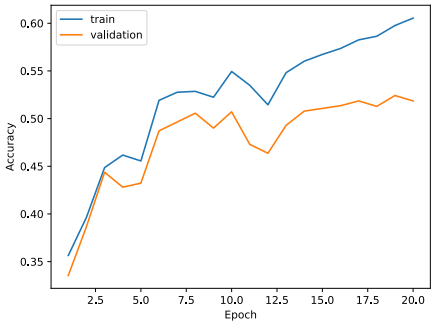
d) Using L1 regularization (L1-norm) instead of L2 regularization (L2-norm) would result in the following differences:
- **Sparsity:** L1 drives many weights to exactly zero, creating a sparse weight vector and performing feature selection. L2, on the other hand, produces small but rarely zero weights, retaining all features.
- **Weight Magnitudes:** L1 results in a few large weights for relevant features, making the model simpler and more interpretable. L2 reduces all weights uniformly, spreading their influence across all features.
- **Model Characteristics:** L1 simplifies the model by selecting key features, while L2 is beneficial when all features contribute meaningfully.

Concluding, L1 regularization produces sparse weights with many zeros, acting as a feature selector. L2 regularization results in smaller, non-zero weights across all features, preventing

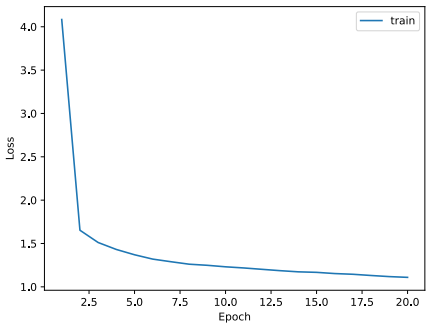
overfitting. This difference reflects L1's ability to act as a feature selector and L2's role in uniformly shrinking weights to prevent overfitting.

3.

a) The performances, with accuracy as the chosen metric were the following: 0.6054 on the training set, 0.5185 on the validation set and 0.5470 on the test set. The training loss was 1.1093.
At the start of training, the model showed an initial train accuracy of 16.15% and a validation accuracy of 16.45% (Plot 6). These values reflected random predictions due to the random initialization of parameters. Over the course of training, the model consistently improved its performance. The training loss (Plot 7) showed a consistent downward trend, indicative of effective learning.



Plot 4 – MLP: train and validation accuracies in function of the epoch



Plot 7 - MLP: train loss as a function of the epoch number

Question 2:

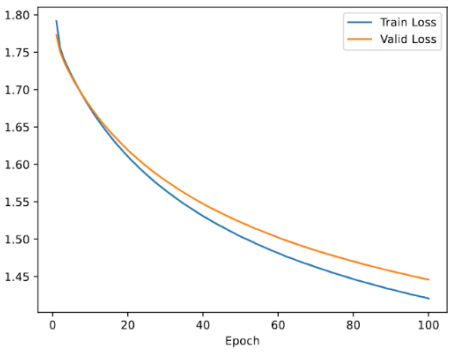
1. The following table shows the final validation and test accuracies for each learning rate.

Learning Rate	Validation Accuracy	Test Accuracy
0.00001	46.94%	46.23%
0.001	52.64%	52.47%
0.1	36.89%	38.43%

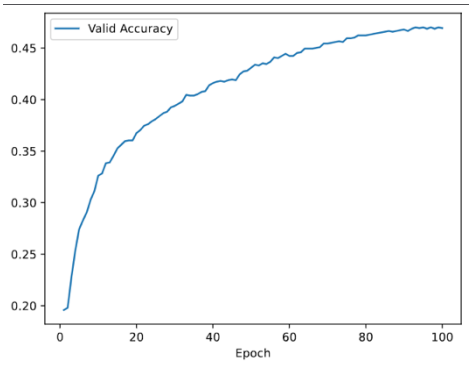
Table 1 - Final validation and test accuracies for each learning rate

The learning rate that achieved the highest validation accuracy was 0.001, yielding a validation accuracy of 52.64% and a test accuracy of 52.47%.
The plots below show the validation accuracy, validation and training loss as a function of the epoch number for each learning rate.

a) $\eta = 0.00001$

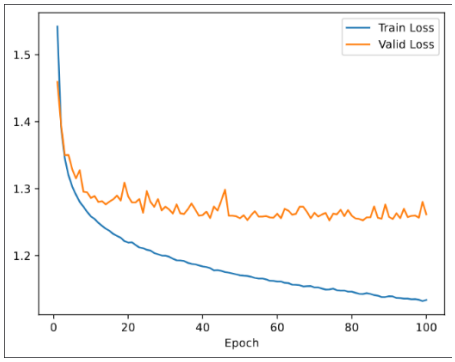


Plot 8 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number

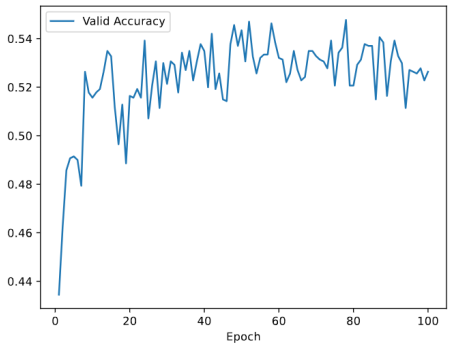


Plot 9 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

b) $\eta = 0.001$

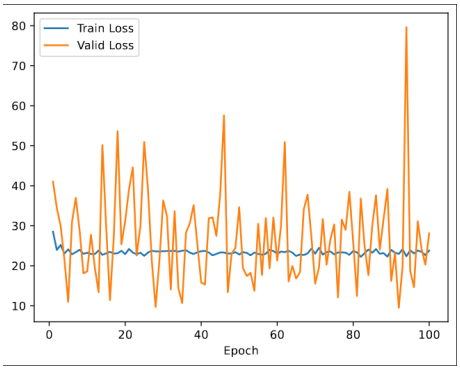


Plot 10 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number

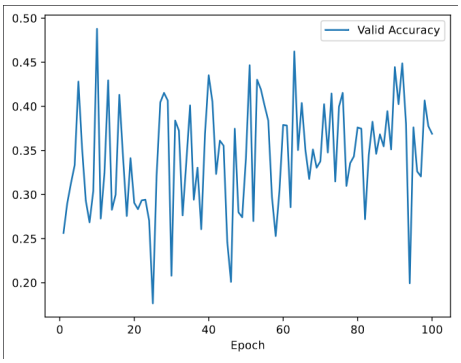


Plot 11 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

c) $\eta = 0.1$



Plot 12 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number



Plot 13 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

- **Smallest Learning Rate (0.00001):** For the smallest learning rate of 0.00001, the training and validation losses decrease steadily but very slowly over 100 epochs. The small updates to the model parameters result in slow convergence. While this approach avoids overshooting or divergence, it would require significantly more epochs to achieve competitive performance compared to other models.
- **Highest Learning Rate (0.1):** In contrast, the model with the highest learning rate of 0.1 exhibits highly unstable behavior. The validation loss fluctuates significantly, reaching values over 10 times higher than those observed with the smallest learning rate, while the training loss stabilizes early at around 25. The large step sizes caused by the high learning rate prevent the model from settling near the optimal parameter values, leading to instability and poor generalization. Consequently, this model achieves the lowest validation and test accuracies of the three.
- **Optimal Learning Rate (0.001):** The learning rate of 0.001 strikes a balance between the two extremes. Both the training and validation losses decrease steadily and at an effective pace. The validation loss stabilizes at around 1.3, slightly higher than the training loss, without the instability seen in the highest learning rate or the slow convergence of the smallest learning rate.

Comparison and Conclusion: Among the three models, the learning rate of 0.001 demonstrates the best performance. It provides sufficient updates to the model parameters without diverging or fluctuating excessively. The steady decrease in validation loss indicates effective learning and convergence within 100 epochs, resulting in the highest validation and test accuracies.

2.

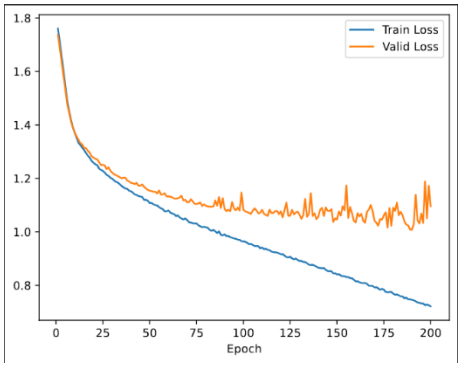
a) The following table shows the final validation and test accuracies and time of execution for each batch size.

Batch Size	Validation Accuracy	Test Accuracy	Time of Execution
64 (default)	58.62%	58.33%	2min, 26s
512	50.85%	53.47%	1m, 39s

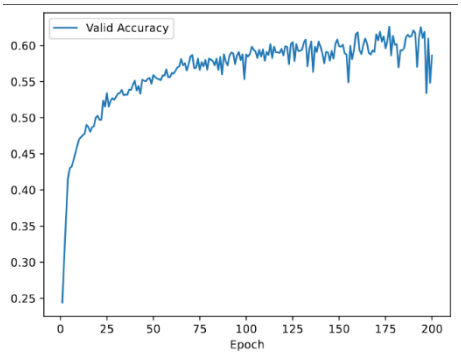
Table 2 - Final validation and test accuracies and time of execution for batch size

The plots below show the validation accuracy, validation and training loss as a function of the epoch number for each batch size.

i) batch size = 64 (default)

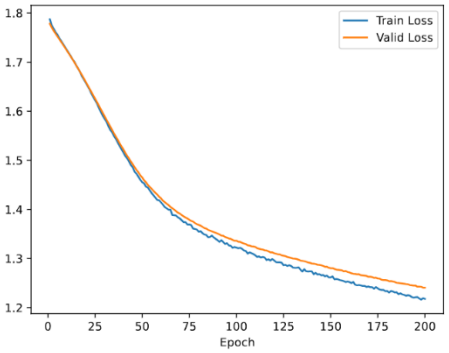


Plot 14 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number

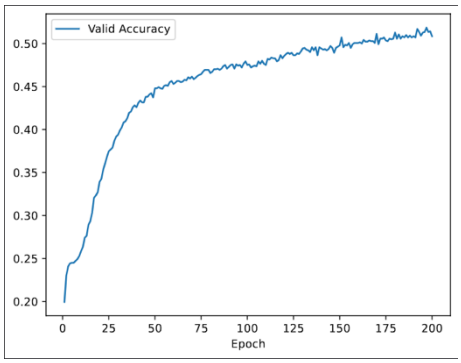


Plot 14 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

ii) batch size = 512



Plot 16 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number



Plot 17 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

Comparison of Batch Sizes: Analyzing the training and validation loss plots, we observe that the model with a batch size of 64 shows faster initial decreases in losses, stabilizing earlier compared to the model with a batch size of 512. This difference is even more pronounced in the validation accuracy plots. For instance, by epoch 25, the model with batch size 64 achieves a validation accuracy of nearly 50%, while the model with batch size 512 reaches only about 30%.

The numerical results further highlight this difference, with the model using batch size 64 showing approximately 15% improvement in validation accuracy and 10% improvement in test accuracy over the model with batch size 512.

These results can be attributed to the nature of smaller batch sizes. A smaller batch size provides more frequent updates to the model's weights, resulting in better gradient estimation and improved learning. However, this comes at the cost of increased training time. Smaller batch sizes require more iterations per epoch, as reflected in the results: the model with batch size 512 completes training 36 seconds faster than the one with batch size 64, representing a 31% reduction in training time.

Trade-offs in Batch Size: In conclusion, choosing a batch size involves a trade-off. Larger batch sizes offer faster training but may compromise generalization and accuracy, while smaller batch sizes improve performance and accuracy at the cost of longer training times. For this experiment, the smaller batch size of 64 delivers better results in terms of accuracy, though at a higher computational cost.

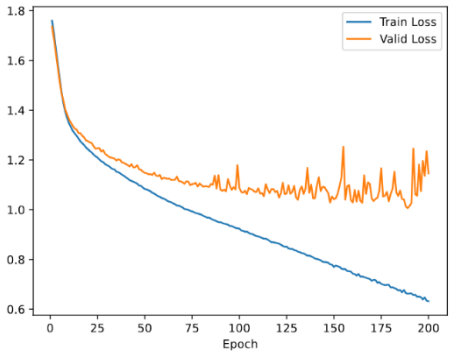
j) The following table shows the final validation and test accuracies and time of execution for each dropout.

Dropout	Validation Accuracy	Test Accuracy
0.01	57.62%	58.03%
0.25	60.83%	60.57%
0.5	59.90%	59.60%

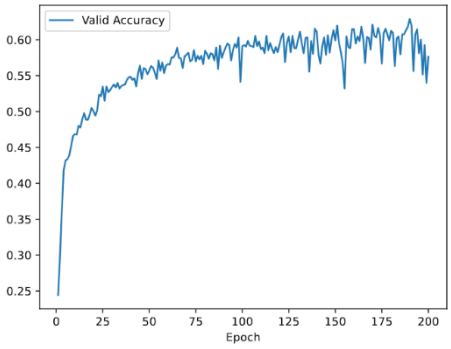
Table 3 - Final validation and test accuracies and time of execution for each dropout

The plots below show the validation accuracy, validation and training loss as a function of the epoch number for each dropout.

i) Droupout = 0.01

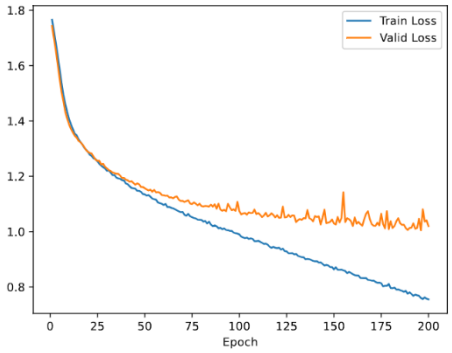


Plot 18 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number

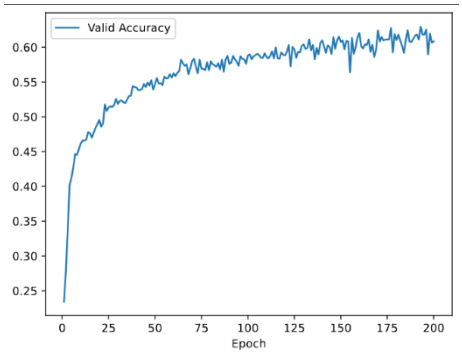


Plot 19 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

ii) Droupout = 0.25

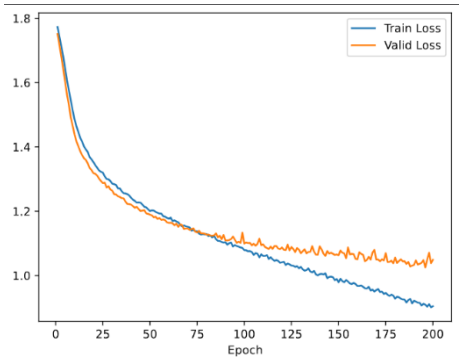


Plot 20 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number

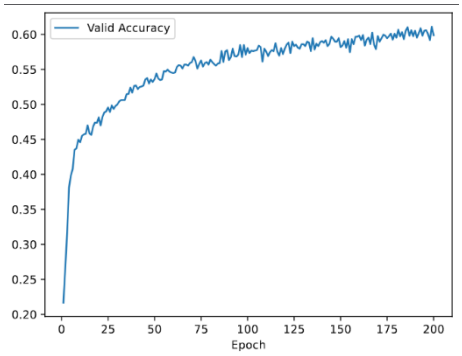


Plot 21 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

iii) Droupout = 0.5



Plot 22 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number



Plot 23 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

- **Smallest Dropout (0.01):** The model with the smallest dropout rate of 0.01 exhibits the lowest values of test and validation accuracy among the three, suggesting that minimal dropout leads to higher overfitting. This happens because with minimal regularization, the model closely fits the training data, overly relying on specific neurons, which limits its ability to generalize effectively.

- **Moderate Dropout (0.25):** The model with a dropout rate of 0.25 has better accuracy than the previous model, as a more substancial regularization prevents overfitting. We have a higher training loss which is balanced with a lower validation loss after stabilizing.

- **Largest Dropout (0.5):** The model with 0.5 dropout rate doesn't improve relating to the previous one on both accuracies, indicating potential over-regularization, but still performs better than the first model with the lowest dropout. Again, the training loss increases in relation to the previous model, as it struggles to learn due to the high rate of dropout, but validation stabilizes at a similar value to the previous model.

Trade-offs in Dropout Rates: The trend we can see from the plots if we compare them in sequence, from smallest to largest dropout, is that the training loss keeps getting higher while the validation loss keeps settling on a lower value with more stability (without so many ups and downs).

The dropout rate of 0.25 provides the best tradeoff between reducing overfitting and maintaining model capacity, as seen in both accuracy metrics and the loss behavior.

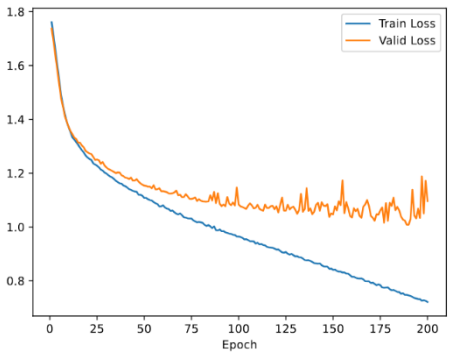
k) The following table shows the final validation and test accuracies and time of execution for each momentum.

Momentum	Validation Accuracy	Test Accuracy
0.0	58.62%	58.33%
0.9	61.75%	61.30%

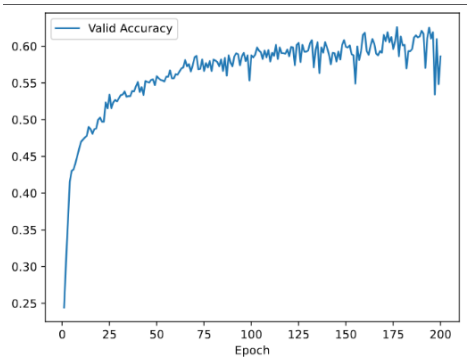
Table 4 - Final Validation and test accuracies and time of execution for each momentum

The plots below show the validation accuracy, validation and training loss as a function of the epoch number for each momentum.

i) Momentum = 0.0

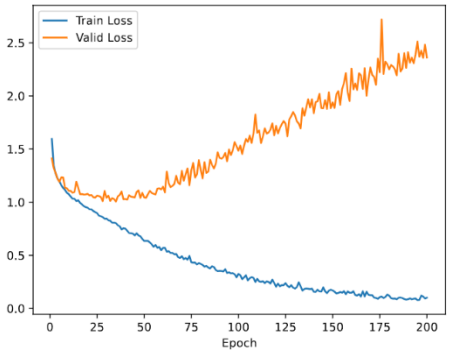


Plot 24 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number

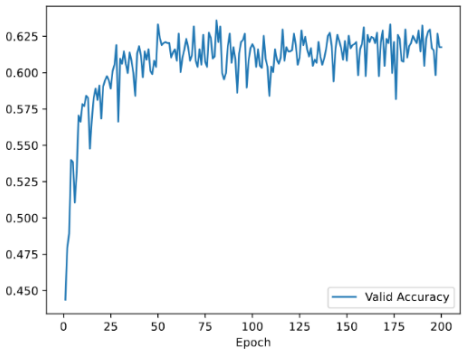


Plot 25 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

ii) Momentum = 0.9



Plot 26 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number



Plot 27 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number

Momentum-0.0 Model: The model with momentum set to 0.0 relies solely on the gradients from the current batch for updates. This leads to slower convergence and steady, but suboptimal, weight adjustments, making it more difficult for the model to escape local minima or saddle points.

As a result, this model achieves lower validation and test accuracy compared to the model with momentum.

Momentum-0.9 Model: In contrast, the model with momentum set to 0.9 benefits from incorporating information from previous updates. This accelerates convergence and allows the optimizer to take more directed steps toward the optimal solution. Consequently, this model achieves higher validation and test accuracy, demonstrating its superior ability to optimize and generalize compared to the no-momentum model.

However, an interesting observation emerges from the validation loss plot of the momentum-0.9 model. After around epoch 50, the validation loss begins to increase, even as the validation accuracy fluctuates between 60-65%. This divergence between training and validation performance indicates potential overfitting, where the model starts to capture noise and less relevant patterns in the training data.

Conclusion: Despite signs of overfitting in the momentum-0.9 model, it remains more effective at generalizing to unseen data than the momentum-0.0 model, as evidenced by its superior test accuracy. This underscores the advantage of using momentum in optimization to accelerate convergence and improve performance.

Question 3

The answers to question 3 of this homework are in the following pages. The start of each exercise is marked with the question number at the beginning of the page.

1. $\phi : \mathbb{R}^D \to \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$

Let w_i be the ith column of W:

$$\begin{aligned} h_i &= (w_i x + b)(1 - (W_i x + b)) \\ &= (W_i x + b)(1 - W_i x - b) \\ &= W_i x - (W_i x)^2 - b W_i x + b - b W_i x - b^2 \\ &= W_i x - (W_i x)^2 - 2b W_i x + b - b^2 \\ &= W_i x (1 - 2b) - (W_i x)^2 + b(1-b) \end{aligned}$$

$\dim(W_i x) = D \qquad \dim(b(1-b)) = 1$

Now let's calculate $\dim((W_i x)^2)$:

$$(W_i x)^2 = \sum_i (W_{ki}^2 x_i^2) + \sum_{i \neq j} W_{ki} W_{kj} x_i x_j, \quad i,j \leq D$$

$$= \sum_i (W_{ki}^2 x_i^2) + 2 \sum_{i<j} W_{ki} W_{kj} x_i x_j, \quad i < D$$

$\dim\left(\sum_i (W_{ki}^2 x_i^2)\right) = D$

last term: $W_{k(D-1)} W_{kD} x_{D-1} x_D$

$\dim\left(2 \sum_{i<j} W_{ki} W_{kj} x_i x_j\right) = \frac{D(D-1)}{2}$

$$\dim(\phi(x)) = D + 1 + D + \frac{D(D-1)}{2} = \frac{2D + 2D + 2 + D^2 - D}{2}$$

$$= \frac{D^2 + 3D + 2}{2} = \frac{(D+1)(D+2)}{2}.$$

Mapping of $\phi(x)$:

- Given $x \in \mathbb{R}^D$, $\phi(x)$ is defined as:

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_D \\ x_1^2 \\ \vdots \\ x_D^2 \\ x_1 x_2 \\ x_1 x_3 \\ \vdots \\ x_{D-1} x_D \end{bmatrix}$$

→ $\phi_1(x)$

→ D elements

→ D elements

→ $\frac{D(D-1)}{2}$ elements

Like described previously, the terms are composed of:

- a constant term : $\phi_1(x) = 1$
- linear terms : The next D elements correspond to the original components of x : $x_1, \dots x_D$
- quadratic terms: which include:
  - $x_i^2, \forall i = 1, \dots, D \rightarrow$ dim D
  - $x_i x_j, \forall i<j, i \leq D \wedge j \leq D$, which are $\frac{D(D-1)}{2}$ elements

Example:

For $D = 2$: if $x = [x_1, x_2]$, then:

$\phi = [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_2^2 \quad x_1 x_2]$, so $\dim(\phi(x)) = 6$

$$\phi(x) = \frac{(2+1)(2+2)}{2} = 6$$

$=$

Now, for the $A\theta$ matrix, which is the weight matrix that maps the transformed features to the hidden representation $h \in \mathbb{R}^k$.

$$A\theta \in \mathbb{R}^{K \times \frac{(D+1)(D+2)}{2}}$$

The matrix $A\theta$ has three main blocks, corresponding to the three groups of features in $\phi(x)$

- Constant terms: $b - b^2$, which corresponds to the first column of $A\theta$:

$$A\theta[:,1] = b_i - b_i^2$$

- Linear Terms: $Wx - 2b(Wx)$, which corresponds to the next D columns of $A\theta$, given by:

$$A\theta[:, 2:D+1] = W_i - 2bW_j$$

- Quadratic terms: $-(Wx)^2$, which corresponds to the next $\frac{D(D+1)}{2}$ columns of $A\theta$, are given by:

$$A\theta[:, D+2:] = -2W_i W_j, \ \forall i \leq j$$

Therefore, the general form of $A\theta$ is given by:

$$a_i = \begin{bmatrix} \text{constant} & \text{linear} & \text{quadratic} \\ \text{terms} & \text{terms} & \text{terms} \end{bmatrix}$$

$$= \begin{bmatrix} b_1 - b_1^2 & w_{11} - 2b_1 w_{11} & w_{12} - 2b_1 w_{12} \cdots & -w_{11}^2 & -w_{12}^2 \cdots & -2w_{11}w_{12} \cdots \\ b_2 - b_2^2 & w_{21} - 2b_2 w_{21} & w_{22} - 2b_2 w_{22} \cdots & -w_{21}^2 & -w_{22}^2 \cdots & -2w_{21}w_{22} \cdots \\ \vdots & \vdots & \vdots \ddots & \vdots & \vdots \ddots & \vdots \ddots \\ b_k - b_k^2 & w_{k1} - 2b_k w_{k1} & w_{k2} - 2b_k w_{k2} \cdots & -w_{k1}^2 & -w_{k2}^2 \cdots & -2w_{k1}w_{k2} \cdots \end{bmatrix}$$

So:

$$A\Theta = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1[(D+1)(D+2)/2]} \\ \vdots & a_{22} & \cdots & a_{2[(D+1)(D+2)/2]} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{k[(D+1)(D+2)/2]} \end{bmatrix}$$

Example:

For $D=2$ and $k=2$, in which D is the number of input features, and k is the number of hidden units, we have:

- $\phi(x) = [1, x_1, x_2, x_1^2, x_2^2, x_1x_2]^T$, $\phi(x) \in \mathbb{R}^6$
- $A\Theta \in \mathbb{R}^{2\times 6} \Leftrightarrow A\Theta \in \mathbb{R}^{12}$

The explicit form of $A\Theta$ is therefore:

$$A\Theta = \begin{bmatrix} b_1 - b_1^2 & w_{11} - 2b_1w_{11} & w_{12} - 2b_1w_{12} & -w_{11}^2 & -w_{12}^2 & -2w_{11}w_{12} \\ b_2 - b_2^2 & w_{21} - 2b_2w_{21} & w_{22} - 2b_2w_{22} & -w_{21}^2 & -w_{22}^2 & -2w_{21}w_{22} \end{bmatrix}$$

4.

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}, \quad N > \frac{(D+1)(D+2)}{2}$$

$X \in \mathbb{R}^{N \times \frac{(D+1)(D+2)}{2}}$, whose rows are the feature vectors $\{\phi(x_n)\}_{n=1}^{N}$

↳ rank $\frac{(D+1)(D+2)}{2}$ ⇒ number of linearly independent rows (or columns) in it

Define $y = (y_1, \ldots, y_N)$

Minimize:

$$L(\theta; \mathcal{D}) = \frac{1}{2} \sum_{n=1}^{N} (\hat{y}(x_n; \theta) - y_n)^2$$

↓ model parameters to be optimized

$\hat{y}(x_n; \theta)$: model's prediction

Since $\text{rank}(X) = \frac{(D+1)(D+2)}{2}$, and $X \in \mathbb{R}^{N \times \frac{(D+1)(D+2)}{2}}$,

we know X has full column rank, which means that the columns of X are linearly independent.

Rewriting Loss:

$$L(\theta; \mathcal{D}) = \frac{1}{2} \|X\theta - y\|_F^2$$

To minimize the loss, we take the gradient of $L(\theta; \mathcal{D})$ with respect to θ and set it to 0.

$$\nabla_\theta L = X^T(X\theta - y) = 0.$$

simplifying, we get:

$$X^T X \theta = X^T y$$ Normal equations for least-squares problem

Solving for c_θ:

- if X^TX is invertible, the normal equations can be solved explicitly:

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

- since X has full column rank, like seen previously, X^TX is invertible
- since $N > \frac{(D+1)(D+2)}{2}$, there is enough data to constrain the solution — overparametrization

Since these conditions are met, the solution c_θ is well-defined and unique.

Therefore, the closed-form solution for the optimization problem is:

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

assumption that the activation functions are quadratic

This problem is special because, for typical feedforward neural networks, the loss function is non-convex due to nonlinear activation functions, which often have multiple local minimums, making global minimization intractable.

On the other hand, the linear dependence on c_θ makes the loss quadratic, resulting in a convex optimization problem. This, along with the full rank of X and the overparametrization, ensures that a global minimum exists and can be computed directly in close form.