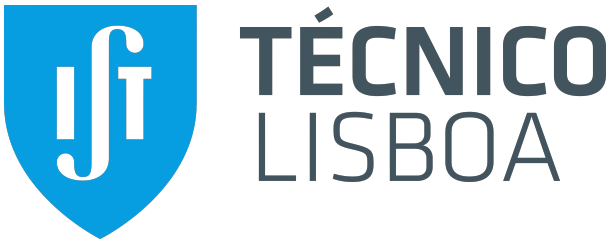

MEIC - Deep Learning (Dei)
2024/2025

Homework 1

Work by Group 34:
Diogo Miguel Castro Paixão - ist113214
João Guilherme Rodrigues Teixeira - ist113227

16/12/2024

Contents

1 Question 1

1.1 Question 1.1

We trained the perceptron algorithm (Figure 1) for 100 epochs, achieving a final training accuracy of 55.98%, a validation accuracy of 38.68%, and a test accuracy of 37.43%, with a total training time of 33 seconds.

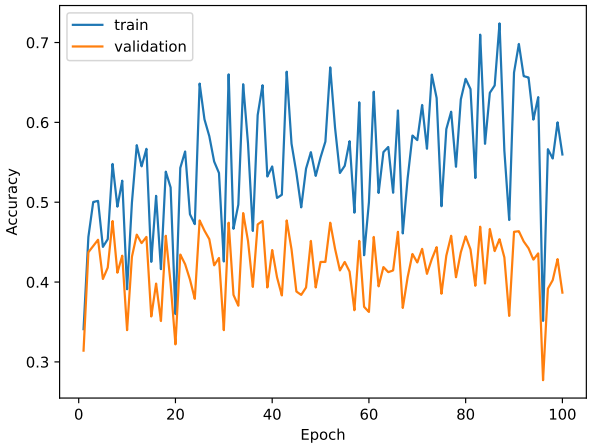

Figure 1: Plot of the train and validation accuracies as a function of the epoch number for the perceptron

1.2 Question 1.2.a

We trained the non-regularized version logistic regression classifier (Figure 2) using stochastic gradient descent for 100 epochs with a learning rate of 0.001, achieving a final training accuracy of 66.94%, a validation accuracy of 42.23%, and a test accuracy of 45.97%, with a total training time of 1 minute and 21 seconds.

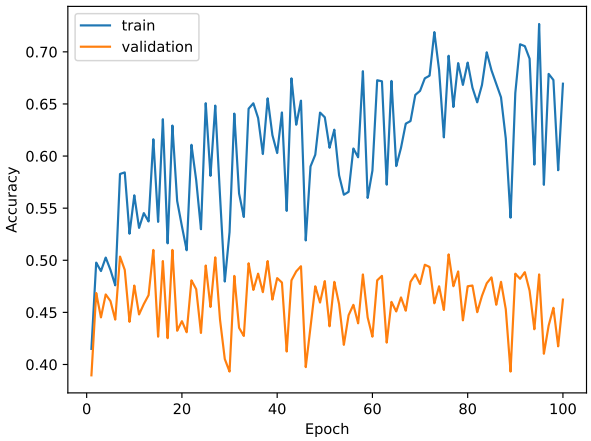

Figure 2: Plot of the train and validation accuracies as a function of the epoch number for the non-regularized LR

1.3 Question 1.2.b

We trained the $l2$-regularized version logistic regression classifier (Figure 3) using stochastic gradient descent for 100 epochs with a learning rate of 0.001 and $l2$-penalty of 0.01, achieving a final training accuracy of 56.83%, a validation accuracy of 49.72%, and a test accuracy of 50.53%, with a total training time of 1 minute and 37 seconds.

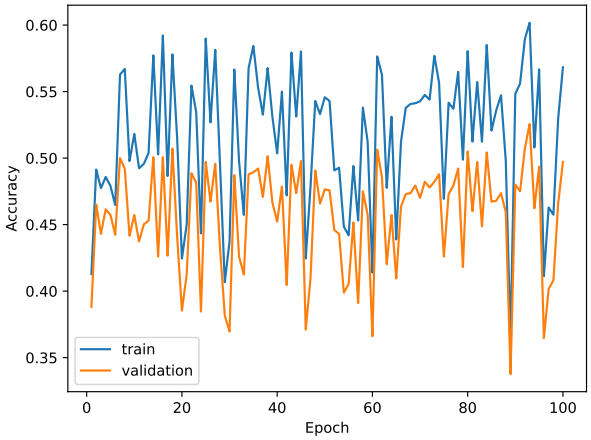

Figure 3: Plot of the train and validation accuracies as a function of the epoch number for the l2-regularized LR

We can see that the introduction of $l2$-regularization resulted in a notable difference in the train and validation accuracies compared to the non-regularized model.

Without regularization, the model achieved a higher training accuracy but suffered from a significantly lower validation accuracy, indicating overfitting.

With $l2$-regularization, the training accuracy decreased, while the validation accuracy improved, demonstrating better generalization to unseen data.

1.4 Question 1.2.c

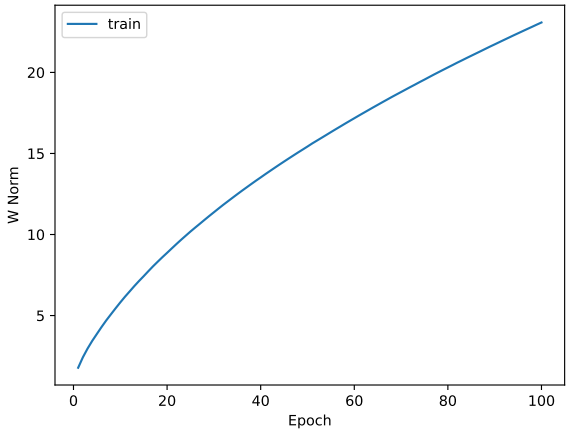

Figure 4: Plot of the weight norms as a function of the epoch number for the non-regularized LR

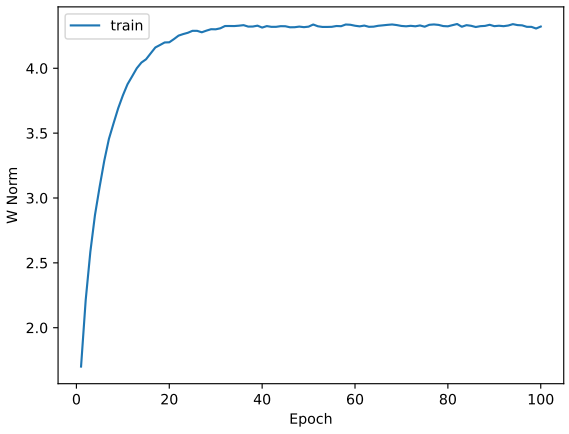

Figure 5: Plot of the weight norms as a function of the epoch number for the l2-regularized LR

We see that the non-regularized model (Figure 4) exhibits a much larger weight norm compared to the regularized model (Figure 5), where the weight values remain smaller and more stable over time.

1.5 Question 1.2.d

If $l1$-regularization were used instead of $l2$-regularization, the impact on the weights would be different.

Unlike $l2$-regularization, which penalizes the square of the weights and tends to shrink all weights uniformly, $l1$-regularization applies a penalty proportional to the absolute value of the weights. As a result, $l1$-regularization encourages sparsity in the model by driving many of the weights to exactly zero.

This sparsity means that only a subset of the features would have non-zero weights, which can make the model more interpretable and reduce its complexity. In contrast, with $l2$-regularization, the weights are typically small but rarely zero, as the penalty continuously shrinks them without completely eliminating them.

1.6 Question 1.3

We trained the multi-layer perceptron (Figures 6 and 7) with a single hidden layer using a gradient backpropagation algorithm, 100 hidden units, a relu activation function for the hidden layers and a multinomial logistic loss for the output layer for 20 epochs with a learning rate of 0.001, achieving a final training accuracy of 59.90%, a validation accuracy of 52.92%, and a test accuracy of 54.23%, with a total training time of 21 minutes and 37 seconds.

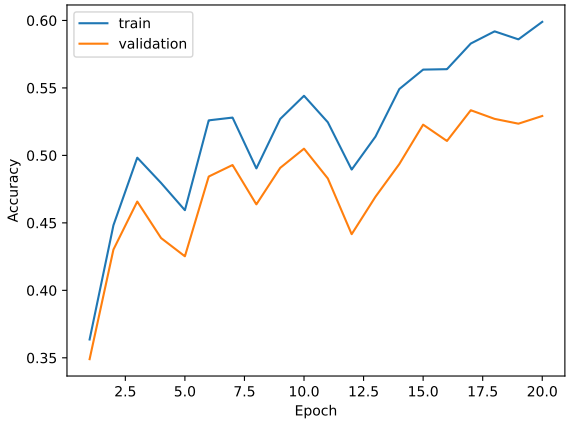

Figure 6: Plot of the train and validation accuracies as a function of the epoch number for the MLP

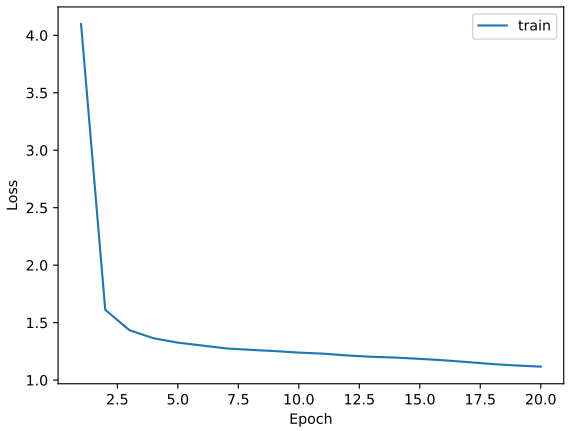

Figure 7: Plot of the loss as a function of the epoch number for the MLP

2 Question 2

2.1 Question 2.1

Using the deep learning framework PyTorch , we trained three logistic regression classifier using stochastic gradient descent for 100 epochs with a batch size of 32 and $l2$-

regularization parameter set to 0.01, with the only difference between them being the learning rates.

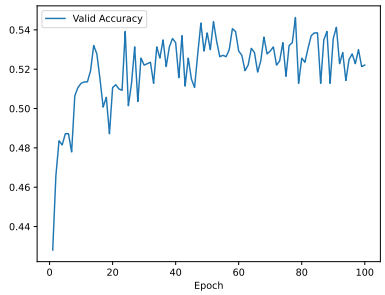

Figure 8: Plot of the validation accuracies as a function of the epoch number for the LR with a learning rate of 0.00001

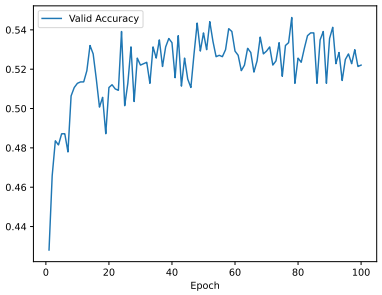

Figure 9: Plot of the validation accuracies as a function of the epoch number for the LR with a learning rate of 0.001

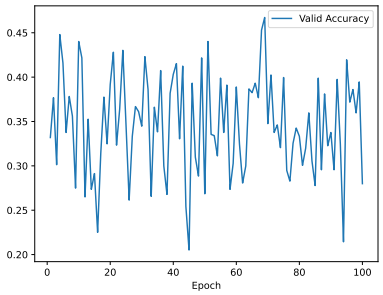

Figure 10: Plot of the validation accuracies as a function of the epoch number for the LR with a learning rate of 0.1

The feed-forward neural network with a learning rate of 0.001 (Figure 9) achieved the highest final validation accuracy of 52.21% and a test accuracy of 52.83%, with a training time of 1 minute and 57 seconds. For the one with a learning rate of 0.00001 (Figure 8) the final validation accuracy dropped to 44.66% and a test accuracy to 45.83%, taking 2 minute and 1 second to train. Finally, the one with a learning rate of 0.1 (Figure 10) had the worst performance, with a final validation accuracy of 27.99% and a test accuracy of 27.93%, taking 1 minute and 55 seconds to train.

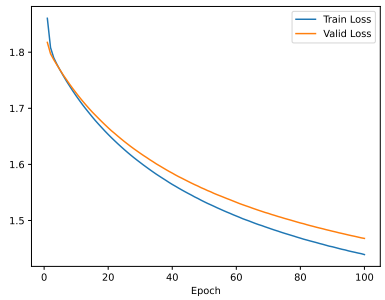

Figure 11: Plot of the train and validation loss as a function of the epoch number for the LR with a learning rate of 0.00001

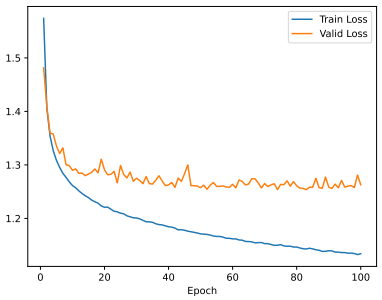

Figure 12: Plot of the train and validation loss as a function of the epoch number for the LR with a learning rate of 0.001

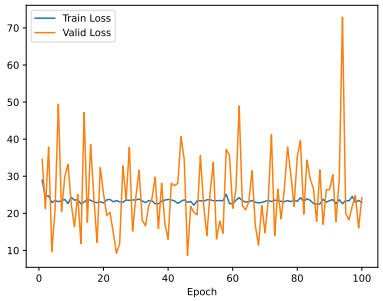

Figure 13: Plot of the train and validation loss as a function of the epoch number for the LR with a learning rate of 0.1

With a learning rate of 0.00001 (Figure 11), the training and validation losses decrease gradually and remain closely aligned throughout the training process. However, the final validation and test accuracy indicate slow convergence, as the learning rate is too small for significant progress within the given number of epochs.

In contrast, with a learning rate of 0.001 (Figure 12), the losses decrease more quickly, and the gap between the training and validation losses remains relatively small.

Finally, with a learning rate of 0.1 (Figure 13), the training and validation losses exhibit highly erratic behavior, with the validation loss fluctuating significantly throughout

the epochs. This instability suggests that the learning rate is too large, preventing the model from converging effectively.

2.2 Question 2.2.a

Using the deep learning framework PyTorch , we trained two feed-forward neural networks for 200 epochs with all the default hyperparameters and the only difference between them being the batch size parameter.

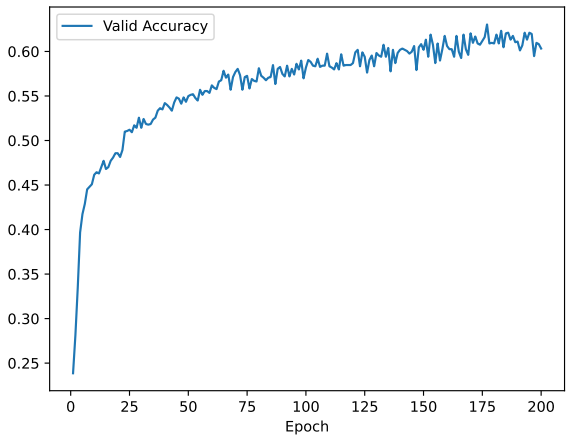

Figure 14: Plot of the validation accuracies as a function of the epoch number for the MLP with a default batch size of 64

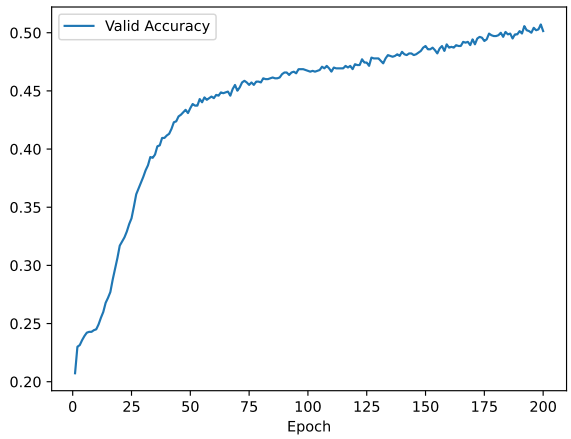

Figure 15: Plot of the validation accuracies as a function of the epoch number for the MLP with a batch size of 512

The feed-forward neural network with the default batch size of 64 (Figure 14) achieved the highest validation accuracy of 60.33%, with a test accuracy of 60.13% and a training time of 10 minutes and 32 seconds. In contrast, a larger batch size of 512 (Figure 15) resulted in a significantly lower final validation accuracy of 50.14%, a test accuracy of 51.90% and a training time of 4 minutes and 31 seconds.

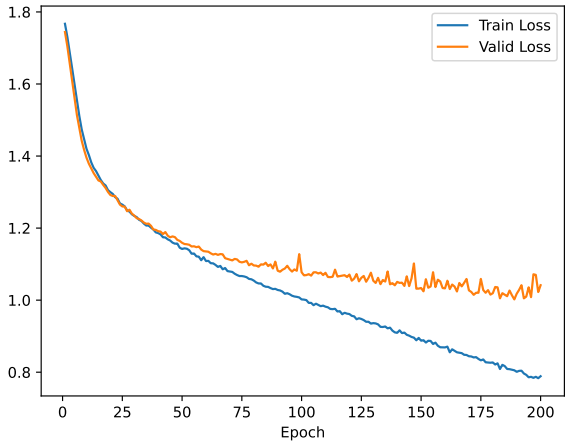

Figure 16: Plot of the train and validation loss as a function of the epoch number for the MLP with a default batch size of 64

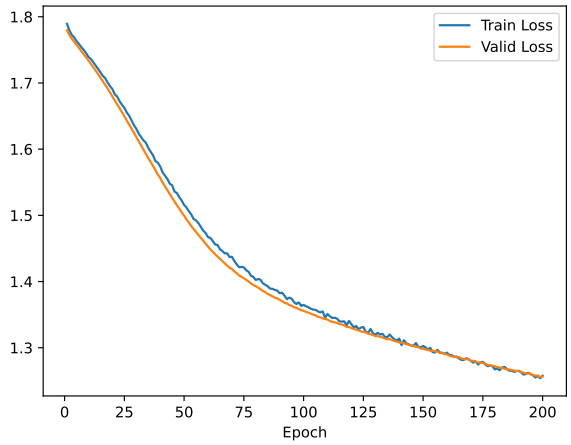

Figure 17: Plot of the train and validation loss as a function of the epoch number for the MLP with a batch size of 512

From the loss plots (Figures 16 and 17), it can be observed that with a batch size of 64, the validation loss decreases consistently but stabilizes at a higher level compared to

the training loss, indicating slight overfitting. For a batch size of 512, both the training and validation loss curves converge more closely, but the overall validation performance remains lower, likely due to insufficient updates caused by the smaller number of gradient steps per epoch associated with the larger batch size.

2.3 Question 2.2.b

Using the deep learning framework PyTorch , we trained three feed-forward neural networks for 200 epochs with all the default hyperparameters and the only difference between them being the dropout parameter.

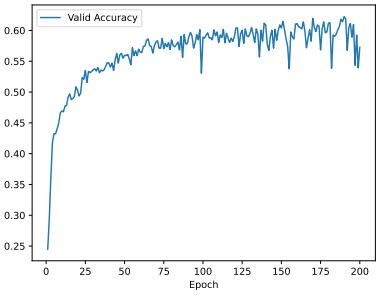

Figure 18: Plot of the validation accuracies as a function of the epoch number for the MLP with a dropout of 0.01

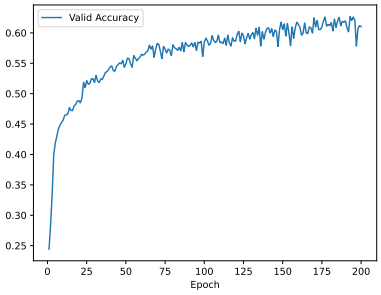

Figure 19: Plot of the validation accuracies as a function of the epoch number for the MLP with a dropout of 0.25

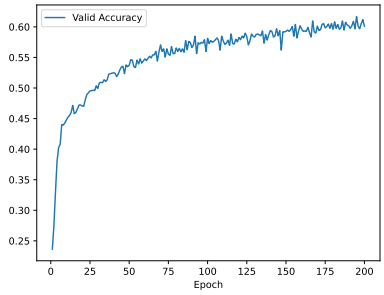

Figure 20: Plot of the validation accuracies as a function of the epoch number for the MLP with a dropout of 0.5

The feed-forward neural network with a dropout of 0.25 (Figure 19) achieved the highest final validation accuracy of 61.04% and a test accuracy of 60.37%, with a training time of 18 minutes and 50 seconds. For the one with a dropout of 0.5 (Figure 20), the final validation accuracy dropped slightly to 60.11%, with a test accuracy of 58.77%, taking 13 minutes and 12 seconds to train. Finally the one with a dropout of 0.01 (Figure 18), achieved a final validation accuracy of 57.26%, and a test accuracy of 56.87%, taking 10 minutes and 30 seconds to train.

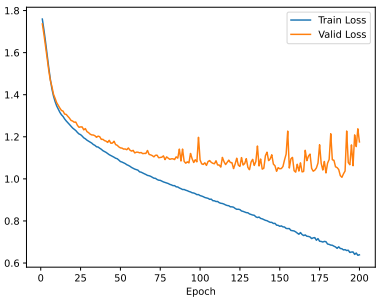

Figure 21: Plot of the train and validation loss as a function of the epoch number for the MLP with a dropout of 0.01

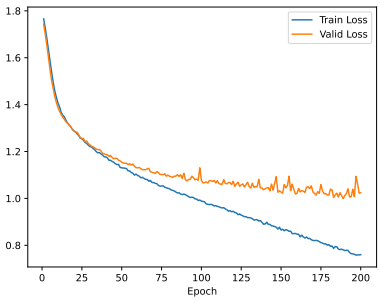

Figure 22: Plot of the train and validation loss as a function of the epoch number for the MLP with a dropout of 0.25

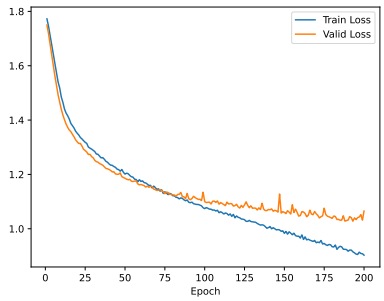

Figure 23: Plot of the train and validation loss as a function of the epoch number for the MLP with a dropout of 0.5

From the loss plots, it can be observed that a dropout of 0.01 (Figure 21) shows significant overfitting, as the training loss decreases continuously while the validation

loss stabilizes at a higher value. For a dropout of 0.25 (Figure 22), the training and validation losses remain closer, indicating the best balance between regularization and performance. A dropout of 0.5 (Figure 23) shows a higher training loss and a more stable validation loss, but the performance suffers slightly due to underfitting caused by the aggressive regularization.

2.4 Question 2.2.c

Using the deep learning framework PyTorch , we trained two feed-forward neural networks for 200 epochs with a batch size of 1024 and all the other hyperparameters at the default value with the only difference between them being the momentum parameter.

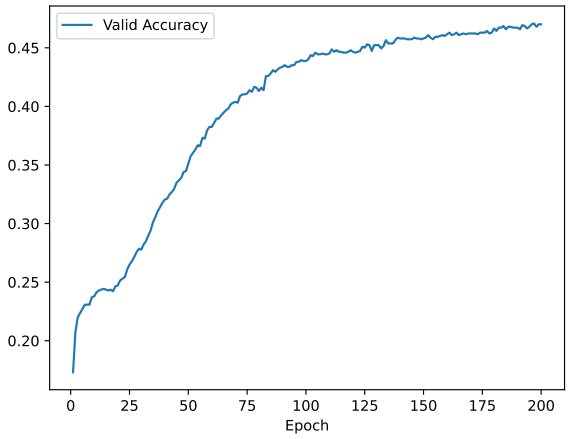

Figure 24: Plot of the validation accuracies as a function of the epoch number for the MLP with a default momentum of 0

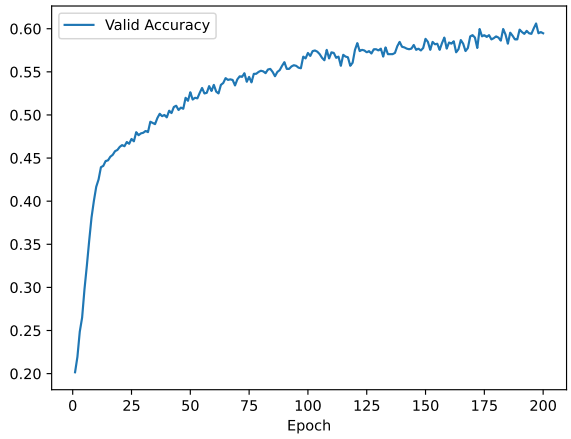

Figure 25: Plot of the validation accuracies as a function of the epoch number for the MLP with a momentum of 0.9

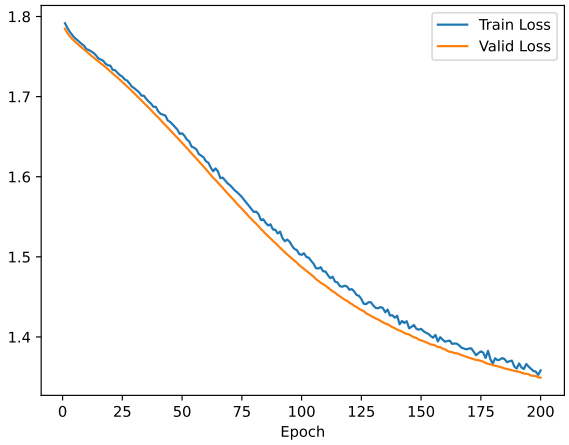

Figure 26: Plot of the train and validation loss as a function of the epoch number for the MLP with a default momentum of 0

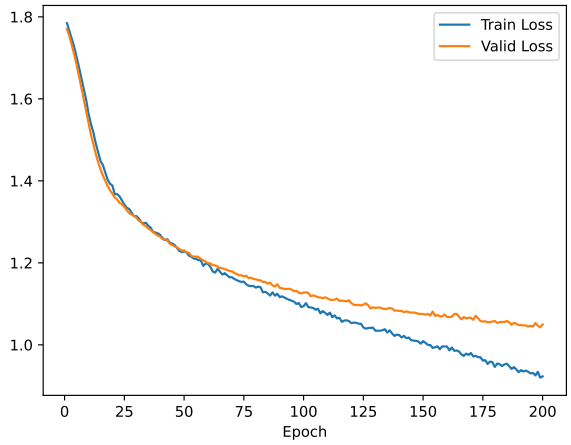

Figure 27: Plot of the train and validation loss as a function of the epoch number for the MLP with a momentum of 0.9

The feed-forward neural network with a momentum of 0.9 (Figure 25) achieved the highest final validation accuracy of 59.47% and a test accuracy of 59.87%, with a training time of 2 minutes and 47 seconds. For the model with a momentum of 0.0 (Figure 24),

the final validation accuracy dropped to 47.01%, with a test accuracy of 48.87%, while maintaining the same training time of 2 minutes and 47 seconds.

From the loss plots, it can be observed that the model with 0.9 (Figure 27) momentum converges faster and achieves a smoother and lower training loss, with the validation loss stabilizing at a lower value, indicating better generalization. In contrast, with 0.0 momentum (Figure 26), the training loss decreases more slowly, and the validation loss remains higher, suggesting slower convergence and poorer generalization.

3 Question 3

3.1 Question 3.1

We have that $g(z) = z(1-z)$ and $h = g(Wx+b)$, where x is a vector of input variables and $\theta = (W, b, v, v_0)$. The objective is to express h as $h = A_\theta \phi(x)$, where $\phi(x)$ is independent of θ and A_θ contains the model parameters.

We start by expanding $g(z)$ and replacing it in h:

$$g(z) = z(1-z) = z - z^2$$

$$h = g(Wx+b) = (Wx+b) - (Wx+b)^2$$

Expanding $(Wx+b)^2$:

$$(Wx+b)^2 = WxWx^T + 2bWx + b^2$$

Thus:

$$h = (Wx+b) - (WxWx^T + 2bWx + b^2)$$

At this point h has:
a constant term: b^2
linear terms: Wx, b, Wx
quadratic terms: $WxWx^T$

To isolate these terms, we can rewrite h as $h = A_\theta \phi(x)$, where:
$\phi(x)$ contains all the possible values of constant, linear and quadratic terms of x;
A_θ is the matrix that contains the coefficients (weights);

Mapping of ϕ:

$$\phi(x) = [1, x_1, ..., x_D, x_1^2, x_1x_2, ..., x_1x_D, ..., x_D^2]^T$$

where:

- 1 is a constant term such that $\phi_1(x) = 1$;
- $x_1, ..., x_D$ are the linear ones;
- $x_1^2, x_1x_2, ..., x_1x_D, ..., x_D^2$ are the quadratic ones;

The total number of features in $\phi(x)$ is $\frac{(D+1)(D+2)}{2}$ and this accounts for all constants, linear and quadratic terms.

Expression for A_θ:

$$A_\theta = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kM} \end{bmatrix} \quad \text{where } M = \frac{(D+1)(D+2)}{2}$$

and a_{ij} is defined as follows:

- $a_{ij} = b_i - b_1^2$ for constant terms ($\phi_1(x) = 1$);
- $a_{ij} = W_{ik} - 2b_i W_{ik}$ for linear terms ($\phi_j(x) = x_k$ for $k = 1, ..., D$)
- $a_{ij} = W_{ik} W_{il}$ for quadratic terms ($\phi_j(x) = x_k x_l$ for $1 \leq k \leq l \leq D$

So, A_θ is the matrix of coefficients that combines features in the vector of transformed features $\phi(x)$ to produce h.

With this we conclude that we can write h as: $h = A_\theta \phi(x)$ for a certain feature transformation $\phi : \mathbb{R}^D \to \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$ independent of θ where $\phi_1(x) = 1$ (i.e., the first feature is a constant feature), and $A_\theta \in \mathbb{R}^{k \frac{(D+1)(D+2)}{2}}$.

3.2 Question 3.2

We want to prove that $\hat{y} = (x; c_\theta) = c_\theta^T \phi(x)$ for some $c_\theta \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$. As we already proved in the previous exercise, $h = A_\theta \phi(x)$, so:

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

We don't need to include v_0 because that constant is already defined in $\phi(x)$.

Let $c_\theta^T = v^T A_\theta$ where A_θ contains the contributions of weights and b to the constant, linear and quadratic terms in $\phi(x)$:

$$\hat{y} = c_\theta^T \phi(x)$$

thus:

$$c_\theta = \begin{bmatrix} c_{constant} \\ c_{linear} \\ c_{quadratic} \end{bmatrix}$$

Where:

- $c_{constant} = \sum_{j=1}^{k} v_j (b_j - b_j^2)$ for constant terms
- $c_{linear,i} = \sum_{j=1}^{k} v_j W_{ji}$ for linear terms
- $c_{quadratic,i,j=\sum_{j=1}^{k} v_j W j i W j k}$ for quadratic terms

This is not a linear model in terms of the original parameters $\theta = (W, b, v, v_0)$, because:

- A_θ, which contributes to c_θ, is derived from non-linear terms from $W, W_{ij} W ik$;
- There is only linearity with respect to $\phi(x)$, not θ;

3.3 Question 3.3

We want to show that it is possible to approximate any vector c using the parameters $\theta = (W, b, v, v_0)$. To achieve this, we use the result from Question 3.2 ($c_\theta = v^\top A_\theta$), where A_θ depends on weights and biases.

$$\|c_\theta - c\| < \varepsilon \quad \Leftrightarrow \quad \|v^\top A_\theta - c\| < \varepsilon$$

We want to solve the following:

$$v^\top A_\theta = c \quad \Leftrightarrow \quad v^\top = c^\top A_\theta^{-1}.$$

This v will satisfy the condition $v^\top A_\theta = c$, if A_θ is non-singular, thus invertible.
If A_θ is found to be singular or nearly singular, we apply a small perturbation:

$$A'_\theta = A_\theta + \varepsilon I,$$

where $\varepsilon > 0$ is a small value. This guarantees A'_θ is non-singular while keeping the perturbation minimal.
We also use orthogonal decomposition to simplify computations:

$$A_\theta = QR,$$

where Q is an orthogonal matrix ($Q^\top Q = I$) and R is an upper triangular matrix. This decomposition ensures numerical stability when computing A_θ^{-1} or $A_\theta'^{-1}$.
Now, to construct θ from c, we follow these instructions:

1. Initialize W and b to construct A_θ.
2. Using the formulas derived in Question 3.1, compute A_θ based on W and b:

- **Constant term:**
$$a_{i,1} = b_i - b_i^2, \quad \text{for } i = 1, \ldots, K.$$

- **Linear terms:** For each x_k, where $k = 1, \ldots, D$:

$$a_{i,1+k} = w_{ik} - 2b_i w_{ik}, \quad \text{for } i = 1, \ldots, K.$$

- **Quadratic terms:** For each $x_k x_l$, where $1 \leq k \leq l \leq D$:

$$a_{i,\text{quad}(k,l)} = w_{ik} w_{il}, \quad \text{for } i = 1, \ldots, K.$$

4. Given c, compute:
$$v^\top = c^\top A_\theta'^{-1}.$$
5. Compute:
$$c_\theta = v^\top A'_\theta,$$

and verify that:

$$\|c_\theta - c\| < \varepsilon.$$

Since A_θ (or A'_θ) is non-singular, the expression $c_\theta = c$ is valid, ensuring the approximation $\|c_\theta - c\| < \varepsilon$.

3.4 Question 3.4

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

Let $X \in \mathbb{R}^{NM}$, where $M = \frac{(D+1)(D+2)}{2}$, be the feature matrix, whose rows are $\phi(x_n)^T$
$c_\theta \in \mathbb{R}^{\mathbb{M}}$: Vector of model parameters
We concluded on Question 3.2 that $\hat{y}(x_n; c_\theta) = c_\theta^T \phi(x)$ so:

$$L(c_\theta; D) = \frac{1}{2} \sum_{n=1}^{n} (c_\theta^T \phi(x_n) - y_n)^2$$

Expressing the summation in matrix form, let's write the difference between the predicted and true outputs as: $Xc_\theta - y$, where Xc_θ is the vector of predicted outputs for all training examples and y is the vector of true outputs.

$$\begin{aligned} L(c_\theta; D) &= \frac{1}{2}\sum_{n=1}^{n}(\hat{y}(x_n; c_\theta) - y_n)^2 = \frac{1}{2}||Xc_\theta - y||_2^2 \\ &= \frac{1}{2}((Xc_\theta - y)^T(Xc_\theta - y)) \\ &= \frac{1}{2}(c_\theta^T X^T X c_\theta - 2y^T X c_\theta + y^T y). \end{aligned} \tag{1}$$

This rewrite is valid because the model is linear in c_θ because $\hat{y}(x_n; c_\theta) = c_\theta^T \phi(x)$ making it possible to compact all the predicted values as Xc_θ

Now we need to compute the gradient.

$$\nabla c_\theta L(c_\theta; D) = \frac{\partial}{\partial c_\theta}(\frac{1}{2}(c_\theta^T X^T X c_\theta - 2y^T X c_\theta + y^T y))$$

We will differenciate term by term:

1. First Term: $\frac{\partial}{\partial c_\theta}(\frac{1}{2}c_\theta^T X^T X c_\theta)$, using $\nabla_x(\frac{1}{2}x^T Ax) = Ax$, the derivative is: $X^T X c_\theta$;
2. Second Term: $\frac{\partial}{\partial c_\theta}(-y^T X c_\theta)$, using $\nabla_x(b^T x) = b$, the derivative is $-X^T y$;
3. Third Term: $\frac{\partial}{\partial c_\theta}(\frac{1}{2}y^T y)$, since this term does not depend on c_θ it's derivative is 0;

Thus:

$$\nabla c_\theta L(c_\theta; D) = X^T X c_\theta - X^T y$$

Now we need to minimize the loss, so we set the gradient to zero:

$$X^T X c_\theta - X^T y = 0^T X c_\theta = X^T y$$

Since $N > M$ and X is derived from a feature transformation $\phi(x)$ that includes all linear and quadratic terms of the original features, it is reasonable to assume that X has full column rank.

Since X has full column rank, then $X^T X$ is positive definite and invertible, so the closed form solution is $\hat{c_\theta} = (X^T X)^{-1} X^T y$

Usually global minimization is intractable for feedforward neural networks because non-linear activations make the optimization problem non-convex, requiring iterative methods to approximate a (possibly local) minimum.

This problem is special because the loss function $L(c_\theta; D) = \frac{1}{2}||Xc_\theta - y||_2^2$ is quadratic in c_θ, meaning that it's second derivative is constant and the loss function is also convex because $X^T X$ is positive definite. Convex functions have a unique global minimum, which we compute directly using the normal equation.

4 Contributions

In terms of contributions to the completion of the homework, we decided to divide the work fairly, with Diogo Paixão completing the first question, João Guilherme working on the second question, and both students collaborating to complete the third question together since it was the hardest in our opinion.