# Deep Learning (IST, 2024-25)

# Homework 1

Diogo Ribeiro
ist1102484

Vasco Paisana
ist1102533

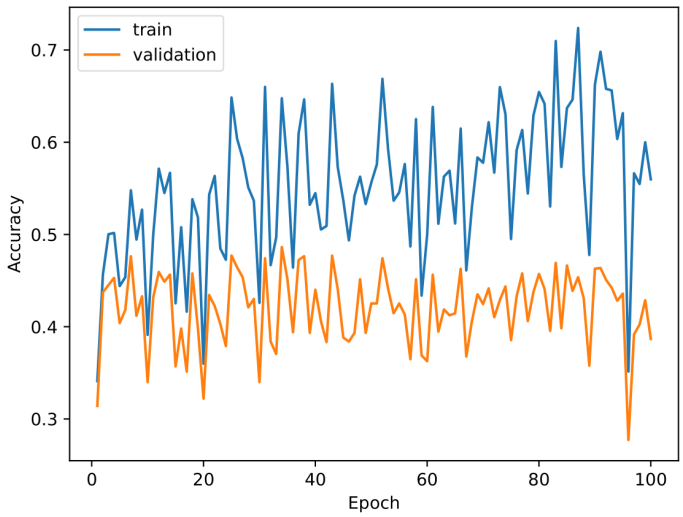
## Question 1

**1.**

**a)** After training the perceptron for 100 epochs, we obtained the following results:

- **train accuracy**: 0.5598
- **validation accuracy**: 0.3868
- **test accuracy**: 0.3743

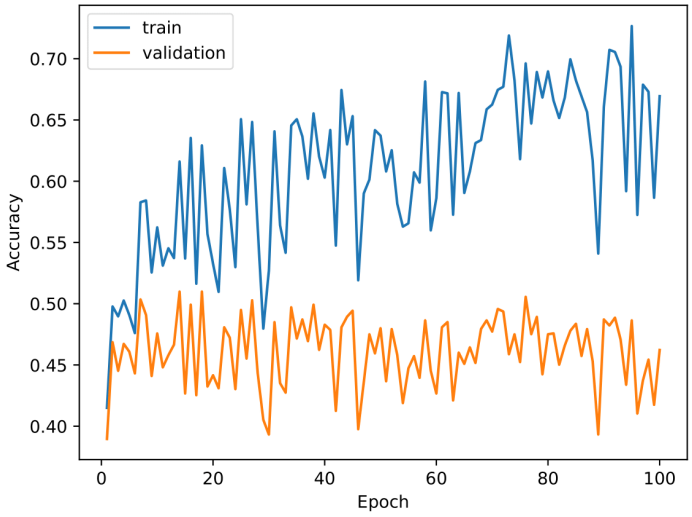
**Perceptron's train and validation accuracies as a function of the epoch number**



**2.**

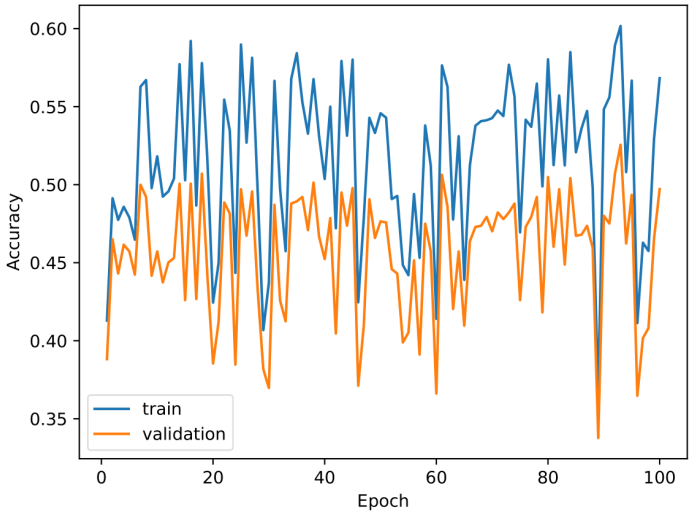
**a)** After training the logistic regression classifier (without $\ell_2$ regularization) for 100 epochs, we obtained a final **test accuracy** of 0.4597.

**Non-regularized logistic regression's train and validation accuracies over the epochs**




**b)** After training the logistic regression classifier (with $\ell_2$ regularization) for 100 epochs, we obtained a final **test accuracy** of 0.5053.

**Regularized logistic regression's train and validation accuracies over the epochs**



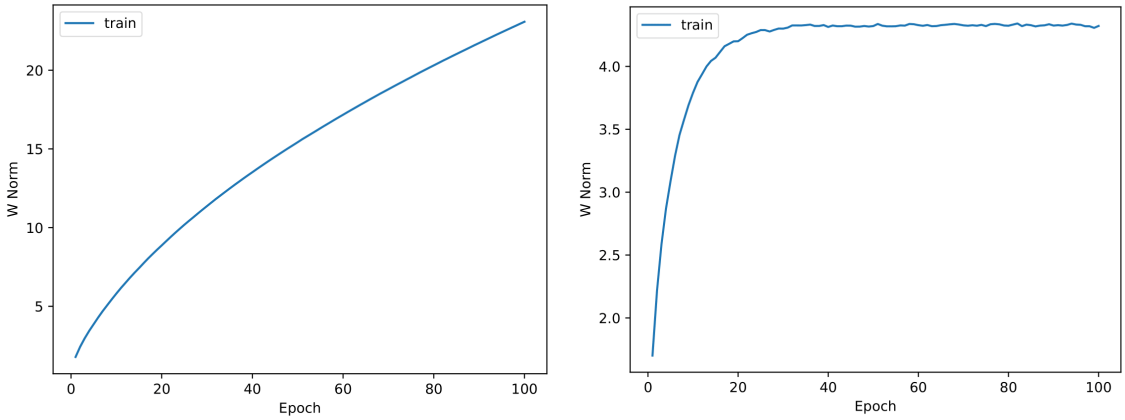

**Analysis:**

Comparing the training and validation accuracies across the stochastic gradient descent epochs, we can see that without regularization the training accuracies tend to be better than with it. Therefore, the differences between training and validation accuracies are greater and tend to increase with the number of epochs, which can make us infer a tendency for overfitting of the model.

So, our analysis enforces that regularization prevents overfitting as expected in theory, reducing and stabilizing the gap between training and testing accuracies. Furthermore, the final test accuracy is better including regularization.

**c)**

**$\ell_2$-norm of the weights of non-regularized (left) and regularized (right) versions of logistic regression as a function of the number of epochs**




**Analysis:**

According to the obtained values, it is visible that without regularization the weights tendency is to increase over the epochs and with regularization the weights become more stable and also smaller than non-regularized ones.

If the weight becomes too large, the regularization term adds a larger penalty, making the loss function increase and leading to the smaller weights seen in the graphics. Therefore, preventing overfitting and reducing the impact of noise in the training data.

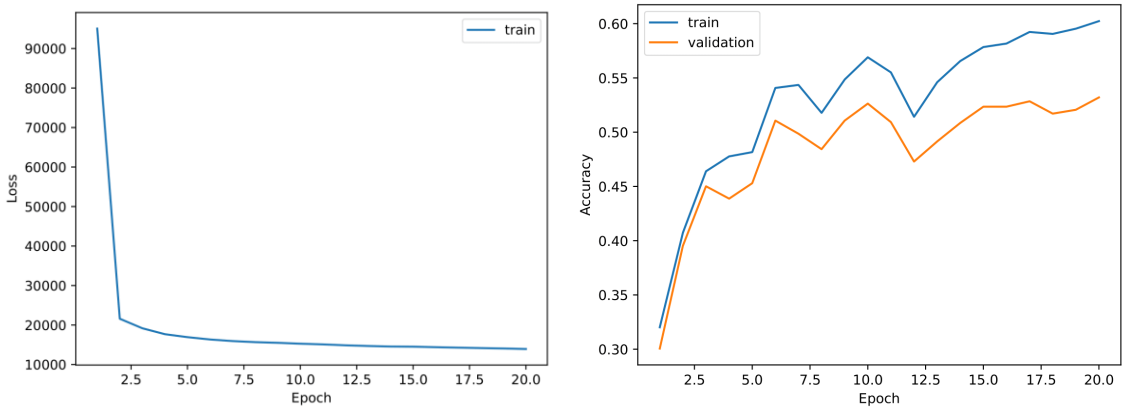
**d)** As $\ell_2$ regularization, using $\ell_1$ regularization the weights are smaller than non-regularized weights, thus preventing overfitting. The main difference between these regularizations is that $\ell_1$ encourages sparsity. Features with small weights have high probability to be set to zero during training.

As of $\ell_1$ norm is defined as the summation of absolute values, the gradient changes abruptly and favors a solution where many weights "spike" down to zero (at sparse points) reducing the number of non-zero weights, which leads some features to be completely ignored and to have fewer but more active features. That does not happen in $\ell_2$ regularization that spreads the penalty evenly across all weights, encouraging them to be small but not zero.

**3.**

**a)** After training the model (multi-layer perceptron with the parameters given in homework statement) for 20 epochs, we obtained a final **test accuracy** of 0.5417.

**Multi-layer perceptron's train loss (left) and train and validation accuracies (right) over the epochs**

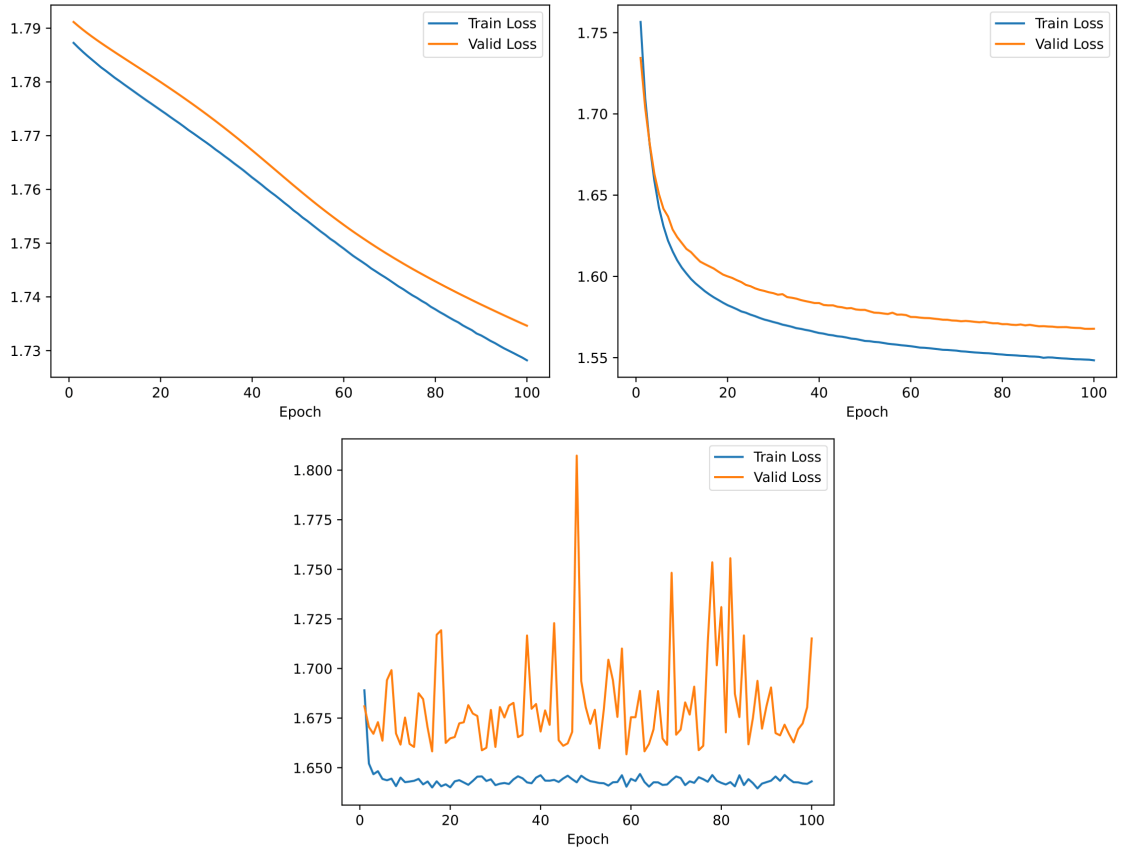



# Question 2

**1.** After training the linear model implemented for 100 epochs with 3 different values for learning rate (0.00001, 0.001, 0.1), we obtained the following results:

- **learning rate 0.00001**
  - **validation accuracy**: 0.2892
  - **test accuracy**: 0.3087
- **learning rate 0.001**
  - **validation accuracy**: 0.4751
  - **test accuracy**: 0.4827
- **learning rate 0.00001**
  - **validation accuracy**: 0.3091
  - **test accuracy**: 0.3093

**Training and validation losses for each configuration – 0.00001 (left); 0.001 (right); 0.1 (bottom)**




**Analysis:**

Best validation accuracy was obtained for learning rate 0.001 (final validation accuracy = 0.4751). As it is possible to observe by the validation accuracy results and by the validation loss curves, from learning rates tested the closest to being optimal is 0.001.

It is perceivable that with very low (like 0.00001) learning rates the curve is almost straight, and the weights are being updated steadily but very slowly, which does not make possible to verify any approximation to a convergence point in which the loss curve reaches a minimum.

That is not the case when looking at the curve associated with the learning rate that led to the best result. We can notice the loss decreases smoothly, almost flattening along the epochs which indicates convergence.

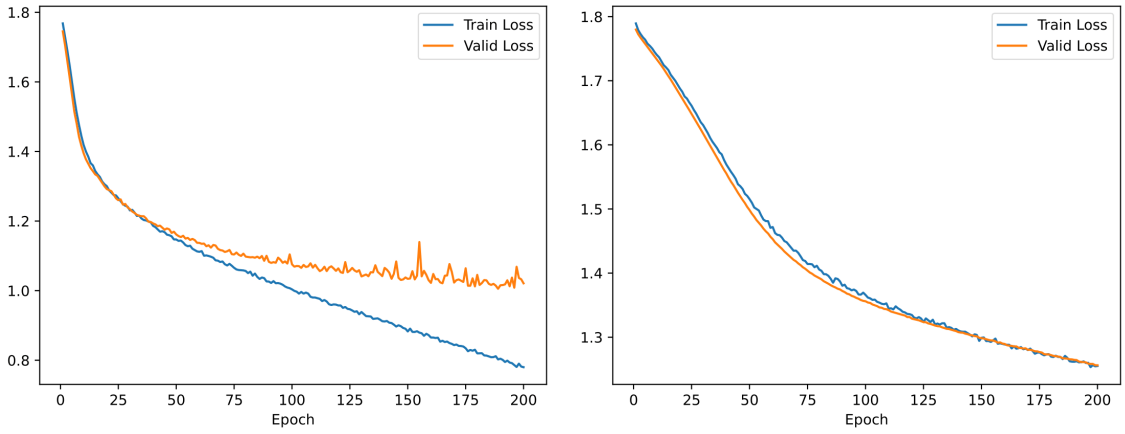
In the other hand, a very high learning rate (like 0.1) cause the loss to fluctuate and possible increase, which shows instability, overshooting the minimum and bouncing around instead of settling.

**2.**

**a)** After training the model for 200 epochs with batch size default (64) and batch size of 512, keeping the other parameters at their default value, we obtained:

- **batch size 64 (default)**
  - **validation accuracy**: 0.6061
  - **test accuracy**: 0.6073
- **batch size 512**
  - **validation accuracy**: 0.5028
  - **test accuracy**: 0.5190

**Train and validation losses for each model – 64 (left); 512 (right)**




**Analysis:**

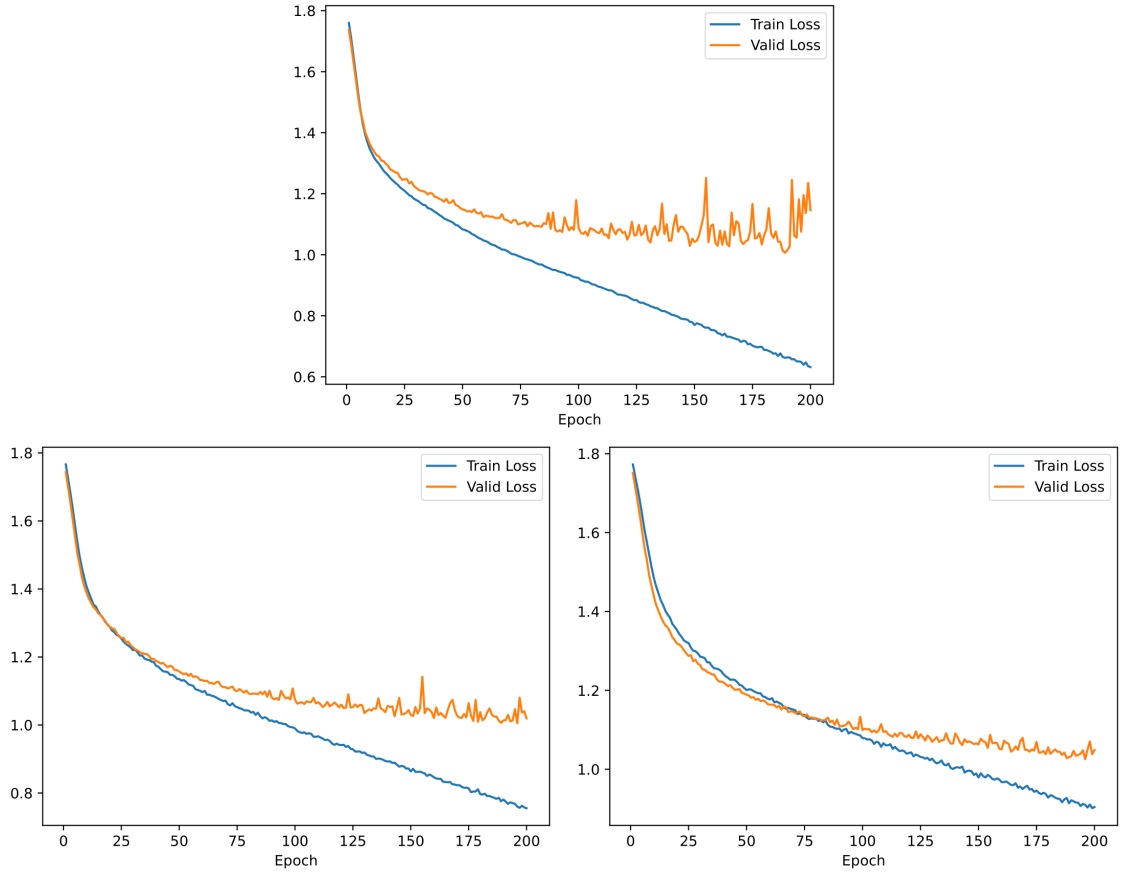
The best performance is observed with the default batch size (64). However, the execution time is shorter with the batch size set to 512. This happens because the weights need to be updated less frequently — specifically, dataset_size / batch_size times. Having a lower batch size implies a more computationally demanding classification task.

Despite this, a lower batch size allows the model to learn more effectively and capture specific patterns in the data, leading to better accuracy. Even though, it tends to overfitting (what can be seen in the loss plots, in which the validation loss flattens regardless of the dropping related with the training loss).

**b)** After training the model for 200 epochs with 3 different dropout values (0.01, 0.25, 0.5), keeping the other parameters at their default value, we obtained:

- **dropout 0.01**
  - **validation accuracy**: 0.5762
  - **test accuracy**: 0.5803
- **dropout 0.25**
  - **validation accuracy**: 0.6083
  - **test accuracy**: 0.6057
- **dropout 0.5**
  - **validation accuracy**: 0.5990
  - **test accuracy**: 0.5960

**Training and validation losses for each dropout value – 0.01 (top); 0.25 (left); 0.5 (right)**




**Analysis:**

Using a low dropout value (e.g. 0.01) only a small fraction of neurons is inactive during training, which translates in a minimal impact in terms of preventing overfitting (this is evident in the loss graph, where for this dropout rate it shows the most pronounced signs of overfitting).

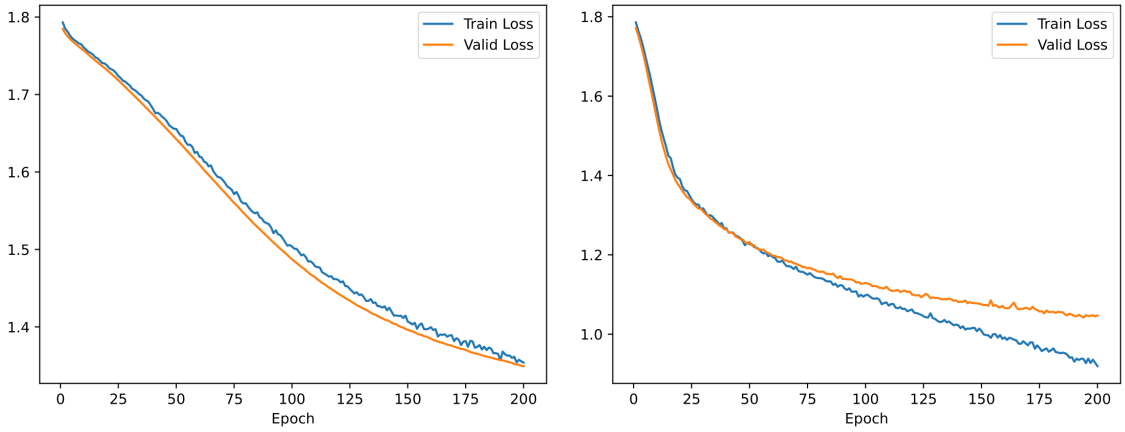
When dropout value is 0.5, half neurons are inactive during each training epoch, besides improving in generalization, it may start to excessively limit the network's capacity to learn patterns, particularly in some cases where the model struggles to converge effectively within each epoch, what is translated into a decreasing of performance. The drop in accuracy from 0.25 enforces this fact.

The best results are from training with dropout 0.25, which may indicate a better trade-off between overfitting and underfitting risks.

**c)** After training the model for 200 epochs with batch size of 1024 and setting momentum to 0.0 and 0.9, while keeping the other parameters at their default value, we obtained:

- **momentum 0.0**
  - **validation accuracy**: 0.4701
  - **test accuracy**: 0.4883
- **momentum 0.9**
  - **validation accuracy**: 0.5990
  - **test accuracy**: 0.6010

**Train and validation losses for each momentum – 0.0 (left); 0.9 (right)**




**Analysis:**

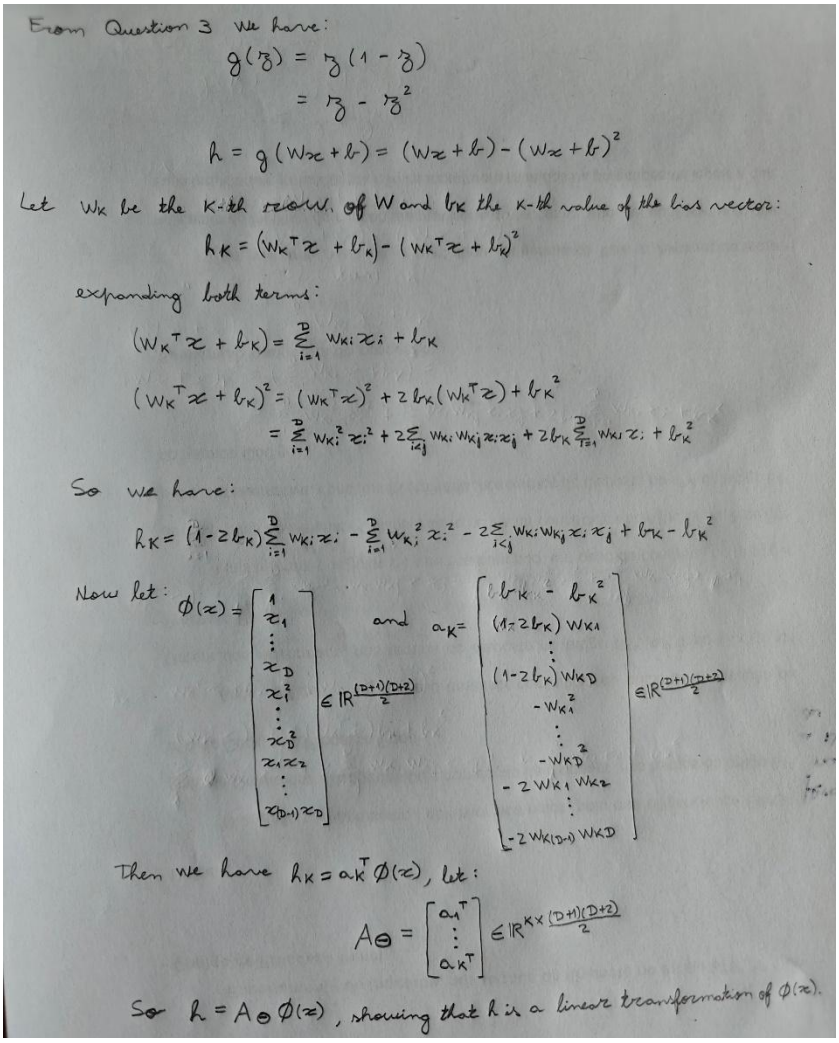
It is possible to verify that with a higher momentum a better performance is achieved.

From the plots it can be seen that the convergence is faster when a momentum is applied. Hence, less epochs are needed for assisting to smaller loss values.

Moreover, the account for the weight of past gradients in the updates helps the algorithm to avoid being stuck in local minimum or saddle points and to get closer to the global minimum, attaining higher accuracy values.

# Question 3

**1.**



From Question 3 we have:

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

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

Let $W_K$ be the K-th row of $W$ and $b_K$ the K-th value of the bias vector:

$$h_K = (W_K^T x + b_K) - (W_K^T x + b_K)^2$$

expanding both terms:

$$(W_K^T x + b_K) = \sum_{i=1}^{D} W_{Ki} x_i + b_K$$

$$(W_K^T x + b_K)^2 = (W_K^T x)^2 + 2 b_K (W_K^T x) + b_K^2$$

$$= \sum_{i=1}^{D} W_{Ki}^2 x_i^2 + 2 \sum_{i<j} W_{Ki} W_{Kj} x_i x_j + 2 b_K \sum_{i=1}^{D} W_{Ki} x_i + b_K^2$$

So we have:

$$h_K = (1 - 2b_K) \sum_{i=1}^{D} W_{Ki} x_i - \sum_{i=1}^{D} W_{Ki}^2 x_i^2 - 2 \sum_{i<j} W_{Ki} W_{Kj} x_i x_j + b_K - b_K^2$$

Now let:

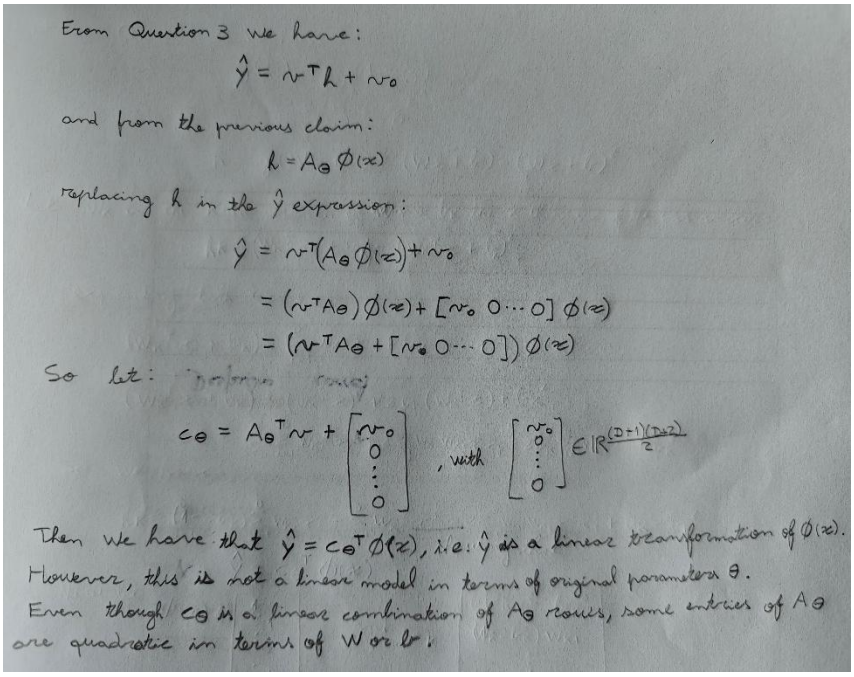
$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_D \\ x_1^2 \\ \vdots \\ x_D^2 \\ x_1 x_2 \\ \vdots \\ x_{(D-1)} x_D \end{bmatrix} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}} \quad \text{and} \quad a_K = \begin{bmatrix} b_K - b_K^2 \\ (1-2b_K) W_{K1} \\ \vdots \\ (1-2b_K) W_{KD} \\ -W_{K1}^2 \\ \vdots \\ -W_{KD}^2 \\ -2 W_{K1} W_{K2} \\ \vdots \\ -2 W_{K(D-1)} W_{KD} \end{bmatrix} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$$

Then we have $h_K = a_K^T \phi(x)$, let:

$$A_\Theta = \begin{bmatrix} a_1^T \\ \vdots \\ a_K^T \end{bmatrix} \in \mathbb{R}^{K \times \frac{(D+1)(D+2)}{2}}$$

So $h = A_\Theta \phi(x)$, showing that $h$ is a linear transformation of $\phi(x)$.

**2.**



From Question 3 we have:

$$\hat{y} = v^T h + v_0$$

and from the previous claim:

$$h = A_\Theta \phi(x)$$

replacing $h$ in the $\hat{y}$ expression:

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

$$= (v^T A_\Theta) \phi(x) + [v_0 \; 0 \cdots 0] \phi(x)$$

$$= (v^T A_\Theta + [v_0 \; 0 \cdots 0]) \phi(x)$$

So let:

$$c_\Theta = A_\Theta^T v + \begin{bmatrix} v_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \text{ with } \begin{bmatrix} v_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$$

Then we have that $\hat{y} = c_\Theta^T \phi(x)$, i.e. $\hat{y}$ is a linear transformation of $\phi(x)$.
However, this is not a linear model in terms of original parameters $\theta$.
Even though $c_\Theta$ is a linear combination of $A_\Theta$ rows, some entries of $A_\Theta$ are quadratic in terms of $W$ or $b$.

**3.**

From above we have:

$$\hat{y} = v^T h + v_0 = \sum_{i=1}^{K} v_i h_i + v_0 = \sum_{i=1}^{K} v_i w_i^T x + \sum_{i=1}^{K} v_i b_i - \sum_{i=1}^{K} v_i \langle\langle w_i w_i^T, x x^T \rangle\rangle$$

$$- 2 \sum_{i=1}^{K} b_i v_i w_i^T x - \sum_{i=1}^{K} v_i b_i^2 + v_0$$

$$= v_0 + \underbrace{\sum_{i=1}^{K} v_i b_i (1 - b_i)}_{\text{Constant terms}} + \underbrace{\sum_{i=1}^{K} v_i w_i^T (1 - 2b_i) x}_{\text{Linear Terms}} - \underbrace{\sum_{i=1}^{K} \langle\langle v_i w_i w_i^T, x x^T \rangle\rangle}_{\text{Quadratic Terms}} =$$

* $\langle\langle \rangle\rangle$ denotes Frobenius inner product; $w_i$ the $i$th row of $W$

$$= c_0 + c_1^T x + \langle\langle C', x x^T \rangle\rangle \quad ; \quad c_0 \in \mathbb{R}, \; c_1 \in \mathbb{R}^D, \; C' \in \mathbb{R}^{D \times D}$$

And:

$$c_\theta = A_\theta^T v + \begin{bmatrix} v_0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$$

Then, for a real vector $c \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$ we can decompose it into its constant, linear and quadratic terms and obtain $\theta$ from $c$.

Firstly we can extract $v_{0_\theta} \in \mathbb{R}$ as the element corresponding with the first entry of $c$, the constant term $c_0$. $b_\theta \in \mathbb{R}^K$ would be obtained from the linear component of $c$. by recovering the following $D$ elements and pad $b_c$ with 0 up to make up to dimension $K$. The quadratic components correspond to the vectorized lower triangular part of a matrix $C$ that can be recovered through orthogonal decomposition. For this reason we need to reshape the remaining $\frac{D(D+1)}{2}$ elements from $c$ into a symmetric matrix $C \in \mathbb{R}^{D \times D}$

After that, we can proceed to the orthogonal eigenvalue decomposition:

$$C = Q \Lambda Q^T, \quad Q \in \mathbb{R}^{D \times D}, \quad \Lambda \in \mathbb{R}^D$$

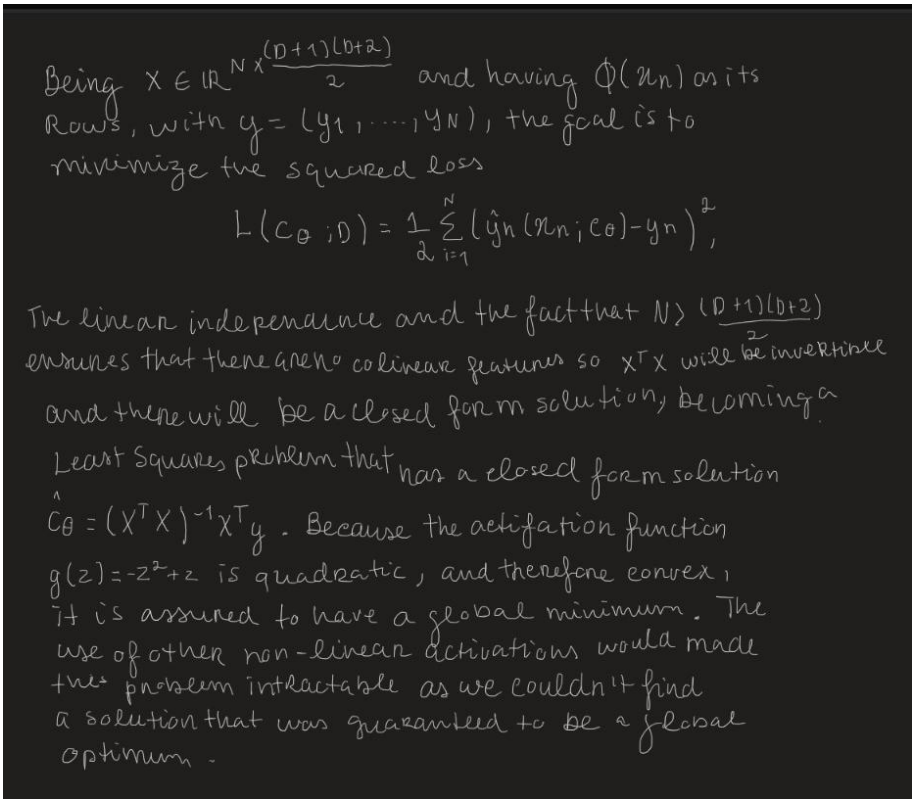
From this we can recover $W_\theta$ by mapping the eigenvectors (columns of $Q$) to the rows of $W$ up to dimension $D$ and generating $K - D$ orthogonal rows so that $W_c \in \mathbb{R}^{K \times D}$; and exctracting the eigenvalues from the diagonal of $\Lambda$, building $v_c$, and padding the remaining $K - D$ values with 0 so that $v_c \in \mathbb{R}^K$

As the orthogonal decomposition can only be applied to non-singular matrix, and it is guaranteed that any matrix can be made arbitrary close to a non-singular matrix (being this arbitrary value $\epsilon$) the matrix $C$ could have been obtained from $C'$,

$$C = C' + \epsilon I$$

Hence, for a real vector $c \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$ and $\epsilon > 0$ there is a choice of the original parameters $\theta$ such $\| c_\theta - c \| < \epsilon$, and the model can be parametrized up to $\epsilon$ precision with $c$ instead of $\theta$.

**4.**




## Contribution of each member & bibliography

Questions 1 & 2 were made by both group elements using VSCode Live Share extension (https://marketplace.visualstudio.com/items?itemName=ms-vsliveshare.vsliveshare).

Question 3 was split: Diogo made 1 and 2 and Vasco made 3 and 4.

To help solve the exercises, we looked for information in theorical and practical classes, and from the following pages:

- https://www.deeplearningbook.org/ (linear algebra section)
- https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0
- https://www.youtube.com/watch?v=dSqKGNT0qaw&ab_channel=Underfitted

ChatGPT was used in Question 1.2d to better understand the meaning of sparsity in $\ell_1$ regularization.

ChatGPT was also used for Question 3.1 to help in the expansion of terms i.e. quadratic terms; and Question 3.3 to understand how to recover W and v from the quadratic components after orthogonalization matrix C, while matching the original dimensions from Θ.