Instituto Superior Técnico

Master Degree in Applied Mathematics and Computation

Deep Learning

Homework 1

Catarina Franco (102678)

Mariana Henriques (103165)

December 2024

Content

Note: We both contributed equally to this project. One of us focused on Question 1 (Mariana Henriques) and the other handled Question 2 (Catarina Franco). For Question 3, we worked together to combine our ideas.

Question 1.

1.

a)

The final training accuracy was 0.5598, the final validation accuracy was 0.3868 and the final test accuracy was 0.3743.

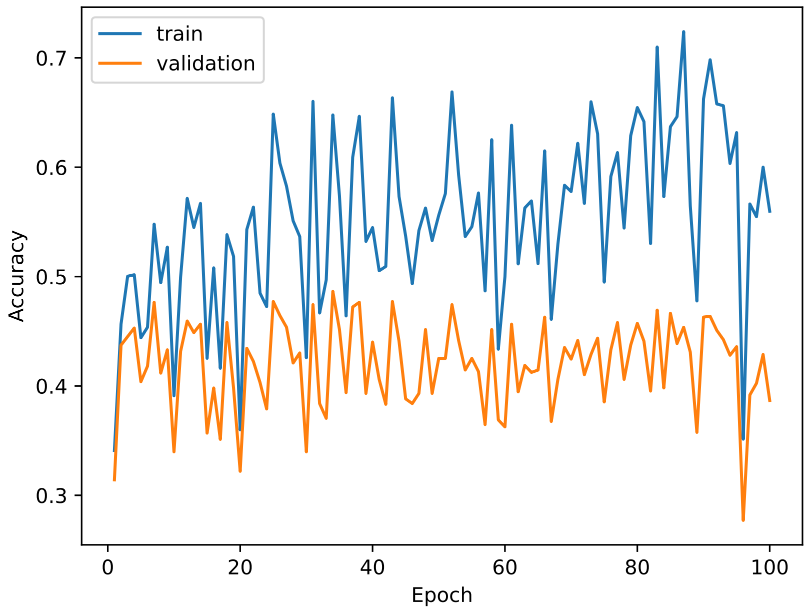

Figure 1. Train and validation accuracies of the percetron classifier.

2.

a)

For the non-regularized logistic regression classifier we have that the final training accuracy was 0.6694, the final validation accuracy was 0.4623 and the final test accuracy was 0.4597.

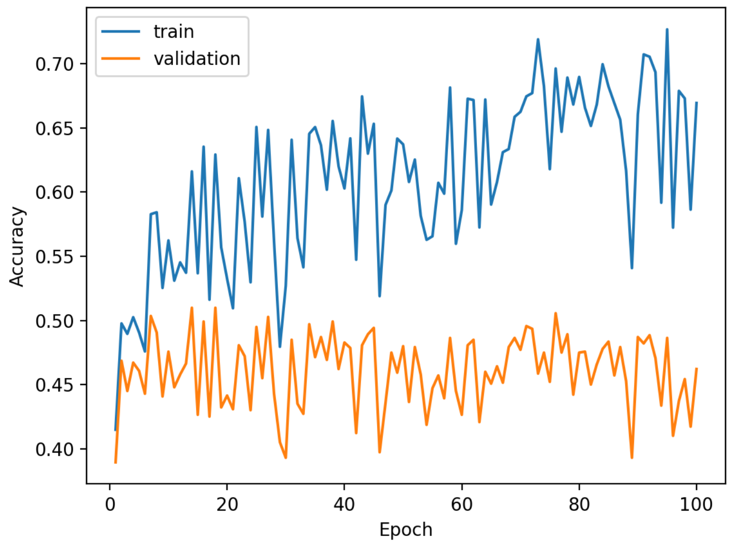

Figure 2. Train and validation accuracies of the non-regularized logistic regression classifier.

b)

For the regularized logistic regression classifier we have that the final training accuracy was 0.5683, the final validation accuracy was 0.4972 and the final test accuracy was 0.5053.

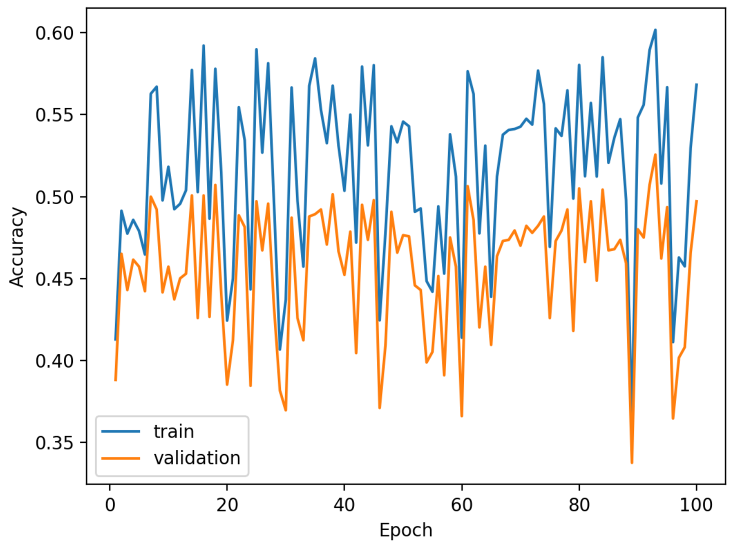

Figure 3. Train and validation accuracies of the regularized logistic regression classifier.

Without regularization (Figure 2), the training accuracy steadily increases and fluctuates around $60 - 70\%$, while the validation accuracy stagnates around $45 - 50\%$ and does not improve over epochs. This suggests that the model overfits to the training data. When ℓ_2 regularization is applied (Figure 3), the training accuracy decreases slightly (around $50 - 60\%$) due to the penalty on large weights, while the validation accuracy remain close to $45 - 50\%$.

This outcome suggests that despite introducing ℓ_2 regularization, the validation accuracy does not improve significantly. This may indicate that the regularization strength is insufficient or the model is not expressive enough. Moreover, as the validation accuracy is quite low compared to the training accuracy, this could mean that the dataset has complex patterns that a simple logistic regression model cannot capture.

c)

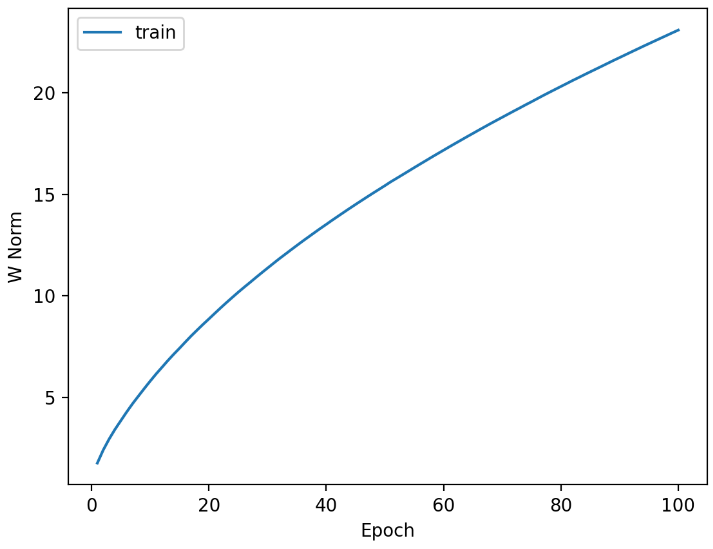

Figure 4. ℓ_2-norm of the weights of the non-regularized logistic regression classifier.

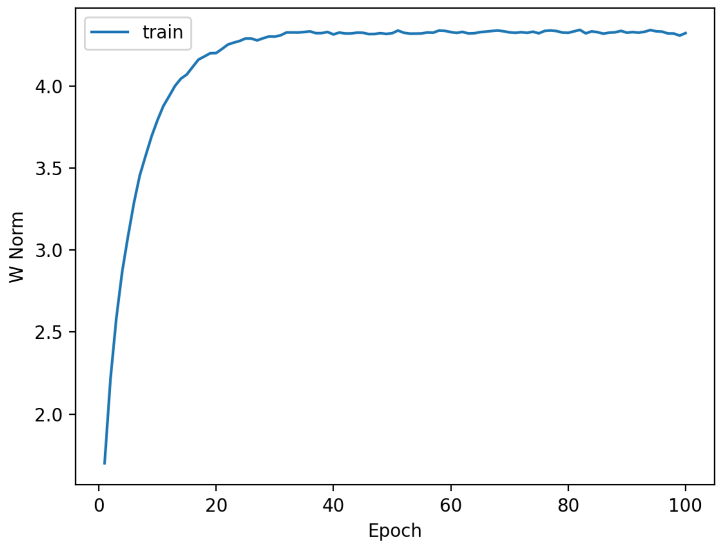

Figure 5. ℓ_2-norm of the weights of the regularized logistic regression classifier.

In the non-regularized model (Figure 4), the weight norm increases without bound, leading to large weights and potential overfitting. This indicates that the model does not have any constraint to prevent weight growth, and its complexity increases with training time.

In the regularized model (Figure 5), the weight norm grows initially but quickly stabilizes. The regularization term penalizes large weights, forcing the model to find a solution where the weights remain small and controlled. This behavior improves generalization by preventing the model from overfitting the training data.

d)

Adding the ℓ_2 penalty to the loss function, it shrinks the weights but does not eliminate them entirely. This means that it keeps all features in the model but with reduce magnitudes. This regularization is useful when there are many features with non-zero contributions and we want to minimize their impact rather than remove them completely.

Adding the ℓ_1 penalty to the loss function, it can skrink some weights to zero, effectively performing feature selection by removing irrelevant or less important features from the model. This helps simplify the model by identifying the most impactful features. This regularization is better suited for cases where feature selection is desired and we expected that only some features are relevant for the predictive model.

In practice, ℓ_1 regularization is preferred when feature selection is desired, while ℓ_2 regularization is better for controlling weight magnitudes without discarding features entirely.

3.

a)

The final test accuracy was 0.5423.

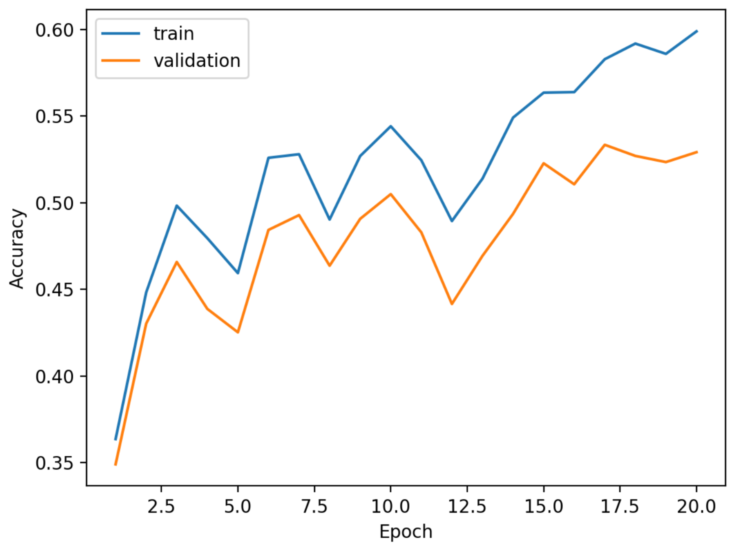

Figure 6. Train and validation accuracies of the multi-layer perceptron per epoch.

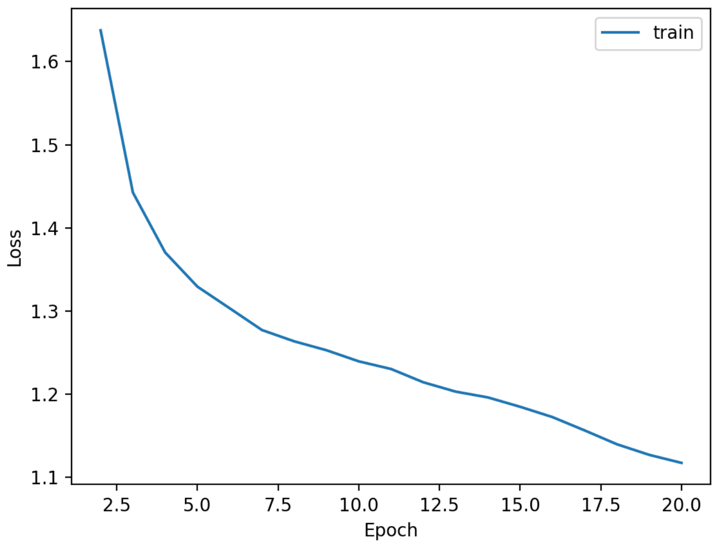

Figure 7. Train loss of the multi-layer perceptron per epoch.

Question 2.

1.

For a learning rate of 0.00001:
The final validation accuracy was 0.4694 and the final test accuracy was 0.4623.

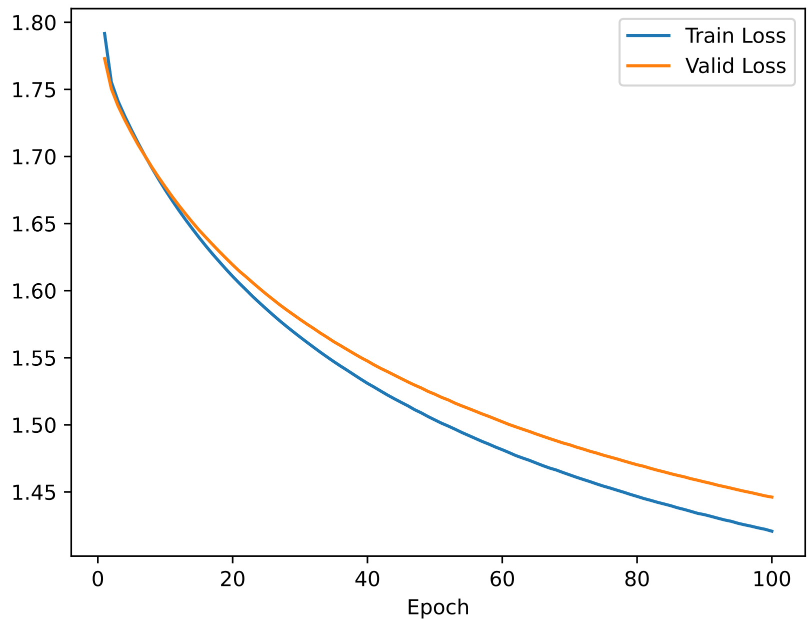

Figure 8. Train loss and validation loss curves for a learning rate of 0.00001.

For a learning rate of 0.001:
The final validation accuracy was 0.5264 and the final test accuracy was 0.5247.

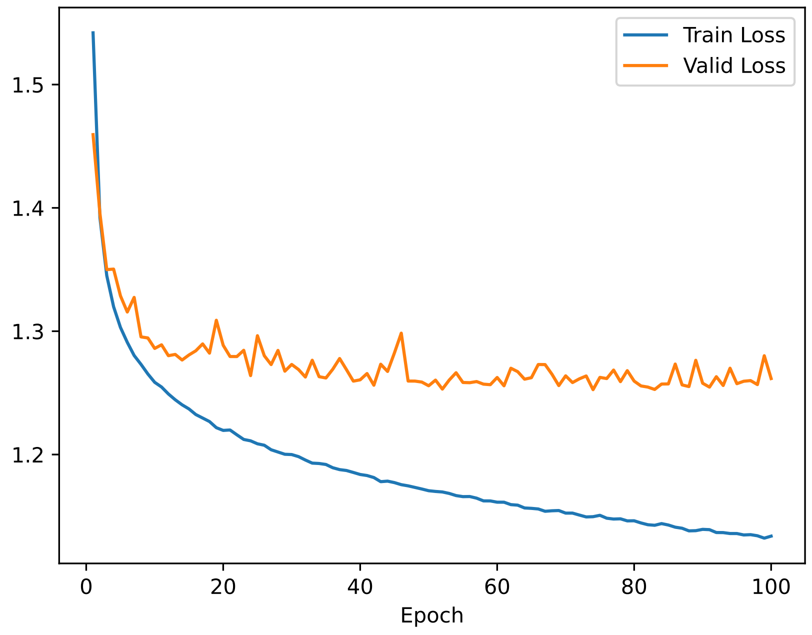

Figure 9. Train loss and validation loss curves for a learning rate of 0.001.

For a learning rate of 0.1:

The final validation accuracy was 0.2913 and the final test accuracy was 0.2757.

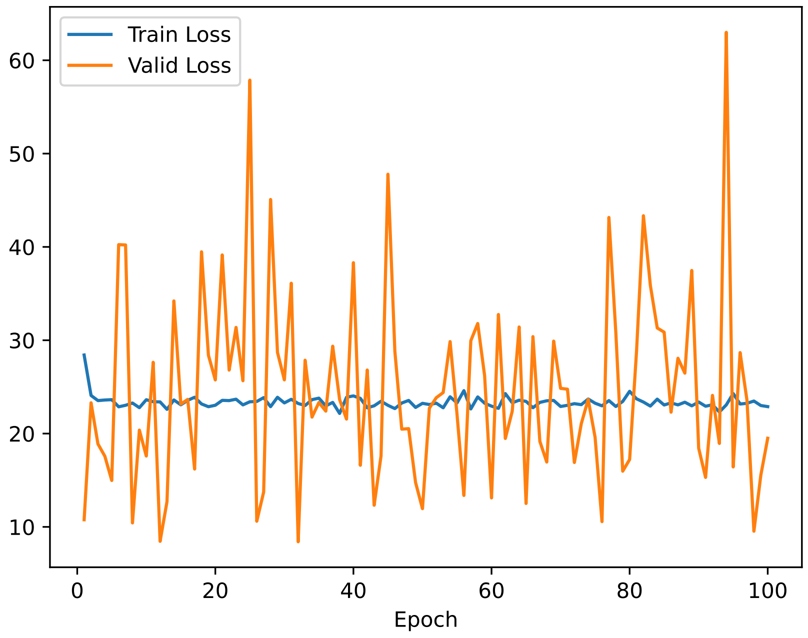

Figure 10. Train loss and validation loss curves for a learning rate of 0.1.

We can conclude that the learning rate that achieved the final highest validation accuracy was 0.001, with a final validation accuracy of 0.5264 and a final test accuracy of 0.5247.

For the learning rate of 0.00001, the validation loss decreases steadily and smoothly over the 100 epochs. However, the rate of decrease is very slow. This behavior occurs because a learning rate of 0.00001 is extremely small, causing the model to make very tiny updates to its weights during each training step. While this ensures that the loss decreases in a stable and consistent manner, the model learns inefficiently, and the overall progress is limited.

For the learning rate of 0.001, the validation loss decreases rapidly at the beginning and then stabilizes after some epochs, showing only minor fluctuations. This behavior reflects the fact that 0.001 is an optimal learning rate for this task, allowing the model to update its weights efficiently without overshooting the loss minimum. The rapid decrease in validation loss during the early stages of training indicates that the model is learning effectively and making significant improvements to its predictions. As training progresses, the validation loss curve stabilizes, suggesting that the model has converged to a local or global minimum.

In contrast, for the learning rate of 0.1, the validation loss curve behaves completely differently, it fluctuates wildly and shows no sign of stabilizing. This happens because the learning rate is far too high, causing the model to take huge steps when updating its weights. These large updates result in the loss oscillating instead of decreasing, preventing the model from converging to a stable solution. As a consequence, the model fails to learn effectively, and the validation loss remains unstable.

2.

a)

For batch size of 64:

The final validation accuracy was 0.5997 and the final test accuracy was 0.5993.
Also, the training took 1 minutes and 27 seconds to be executed.

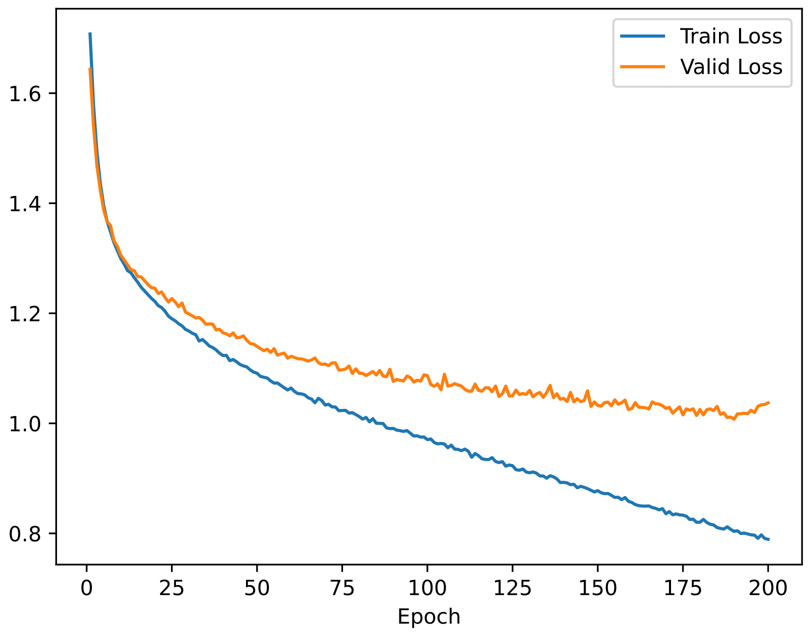

Figure 11. Train loss and validation loss curves for a batch size of 64.

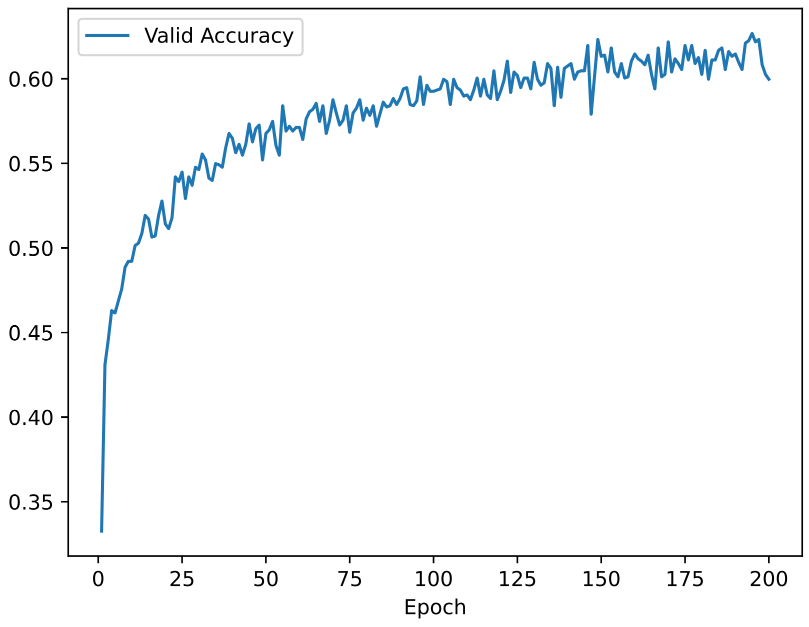

Figure 12. Validation accuracies for a batch size of 64.

For a batch size of 512:

The final validation accuracy was 0.5313 and the final test accuracy was 0.5440.
Also, the training took 0 minutes and 49 seconds to be executed.

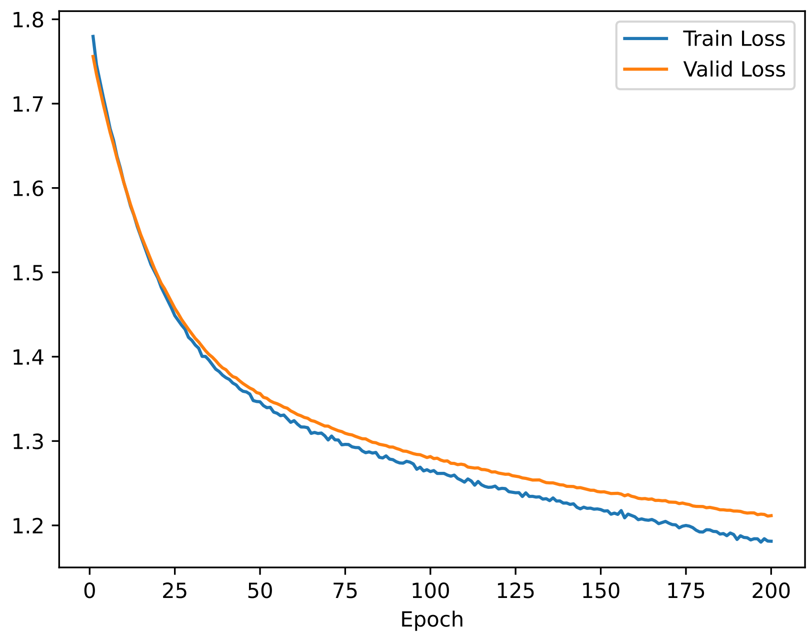

Figure 13. Train loss and validation loss curves for a batch size of 512.

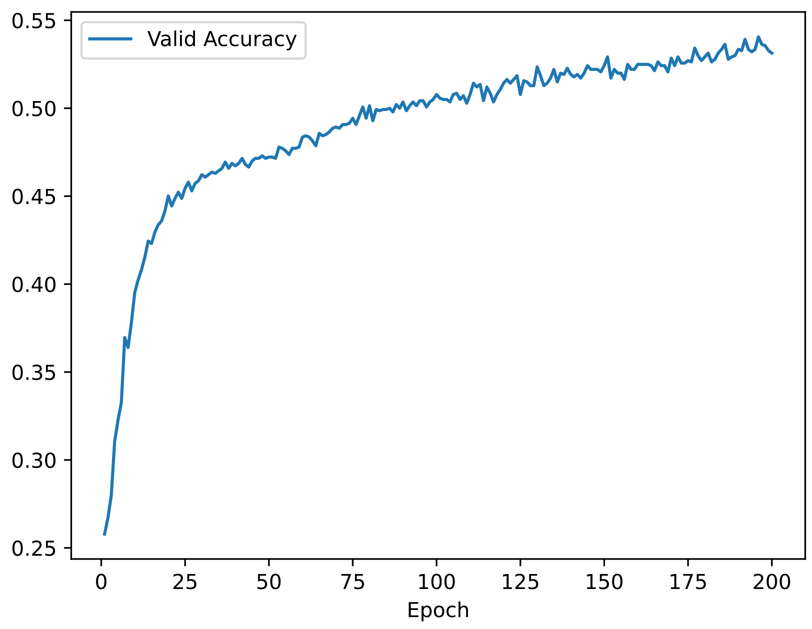

Figure 14. Validation accuracies for a batch size of 512.

The batch size 64 achieved higher validation accuracy (0.5997) and better generalization, though it showed slight overfitting as the validation loss plateaued and increased slightly toward the end. In contrast, the batch size 512 had lower validation accuracy (0.5313) but exhibited less overfitting, as training and validation losses remained close. However, its final loss values were higher, indicating weaker generalization. Moreover, the training time differed significantly. While with the batch size 64 took 1 minute and 27 seconds, with the batch size 512 completed in 49 seconds due to fewer iterations per epoch. Then, we can conclude that the batch size 64 offers better accuracy and generalization but slower training, while batch size 512 provides faster training at the cost of reduced accuracy and potential underfitting.

b)

For a dropout of 0.5:
The final validation accuracy was 0.6040 and the final test accuracy was 0.6033.

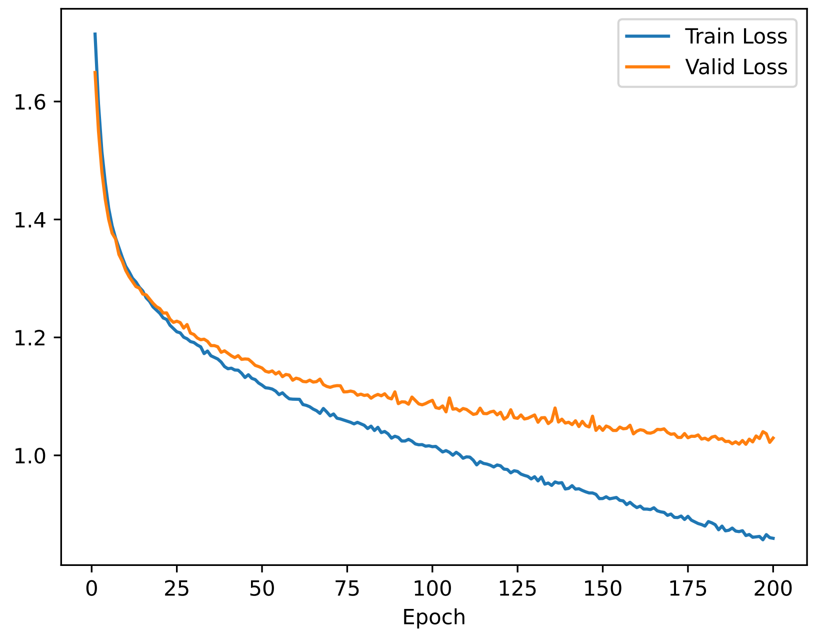

Figure 15. Train loss and validation loss curves for a dropout of 0.5.

For a dropout of 0.25:
The final validation accuracy was 0.5976 and the final test accuracy was 0.5973.

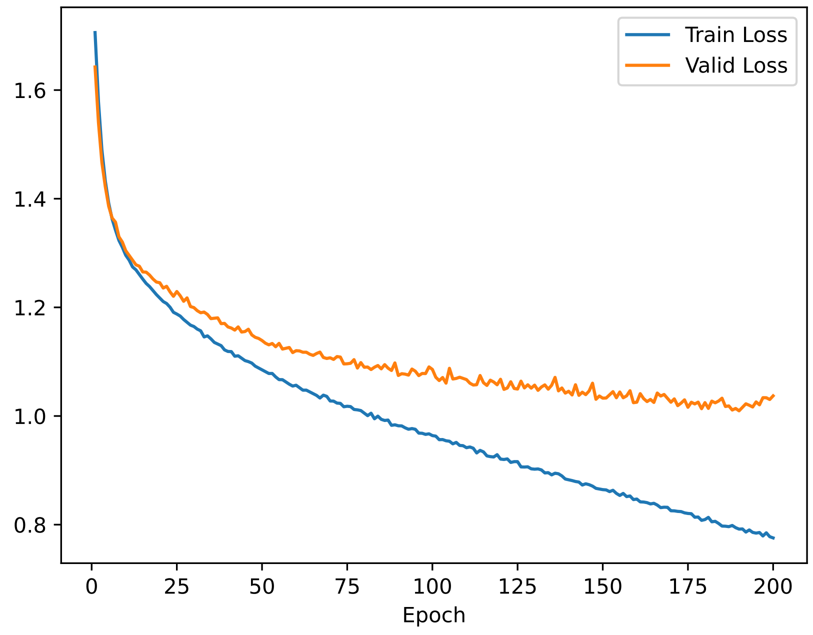

Figure 16. Train loss and validation loss curves for a dropout of 0.25.

For a dropout of 0.01:

The final validation accuracy was 0.5812 and the final test accuracy was 0.5850.

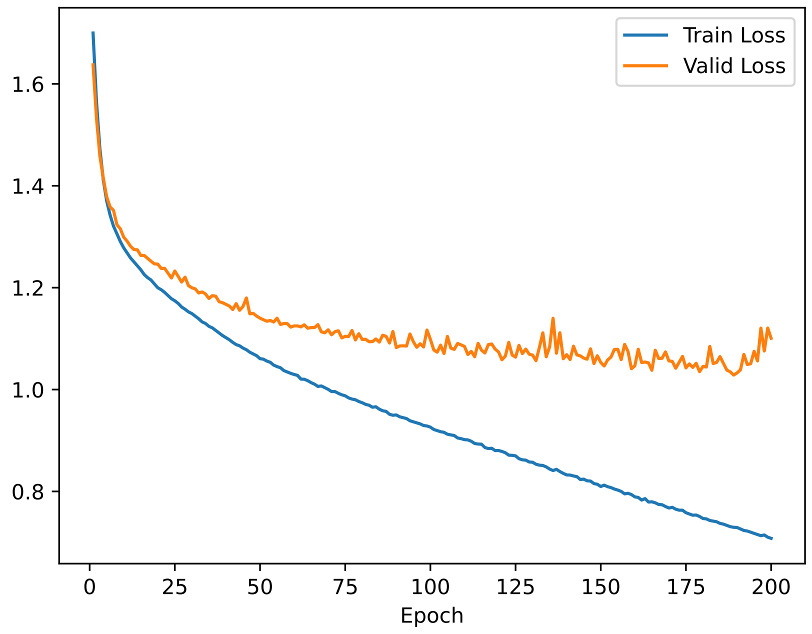

Figure 17. Train loss and validation loss curves for a dropout of 0.01.

A dropout rate of 0.5 results in a training loss that decreases steadily and smoothly over the epochs, while the validation loss decreases initially but stabilizes at a higher value. The validation loss curve remains relatively stable with minimal fluctuations. This behavior can be attributed to strong regularization, as a high dropout rate randomly drops half of the neurons during training. While this prevents overfitting, it also slows down learning since the reduced capacity of the network makes it harder for the model to converge to lower training and validation loss values. The final validation accuracy achieved was 0.6040, and the final test accuracy was 0.6033, highlighting strong generalization despite slower convergence.

With a dropout rate of 0.25, the training loss decreases steadily and reaches lower values compared to a dropout rate of 0.5. The validation loss also decreases initially and stabilizes at a higher value, remaining stable with minor fluctuations. However, the gap between the training and the validation loss is bigger than the previous one. A moderate dropout rate like 0.25 allows the model to learn more effectively, because it leads to better generalization, as seen in the higher validation loss compared to the dropout rate of 0.5. The final validation accuracy for this configuration was 0.5976, with a test accuracy of 0.5973, showing effective but slightly lower generalization compared to the higher dropout rate.

At a very low dropout rate of 0.01, the training loss decreases rapidly and reaches the lowest values among the three configurations. However, the validation loss fluctuates significantly after about 100 epochs and shows an upward trend toward the end, with the largest gap between training and validation loss. This indicates minimal regularization, which allows the model to memorize the training data rather than learning generalizable features. The rapid decrease in training loss and the instability in validation loss suggest overfitting, where the model performs well on the training set but poorly on unseen data. This is reflected in the final validation accuracy of 0.5812 and test accuracy of 0.5850, the lowest among the three configurations.

c)

For a momentum of 0.0:

The final validation accuracy was 0.5014 and the final test accuracy was 0.5203.

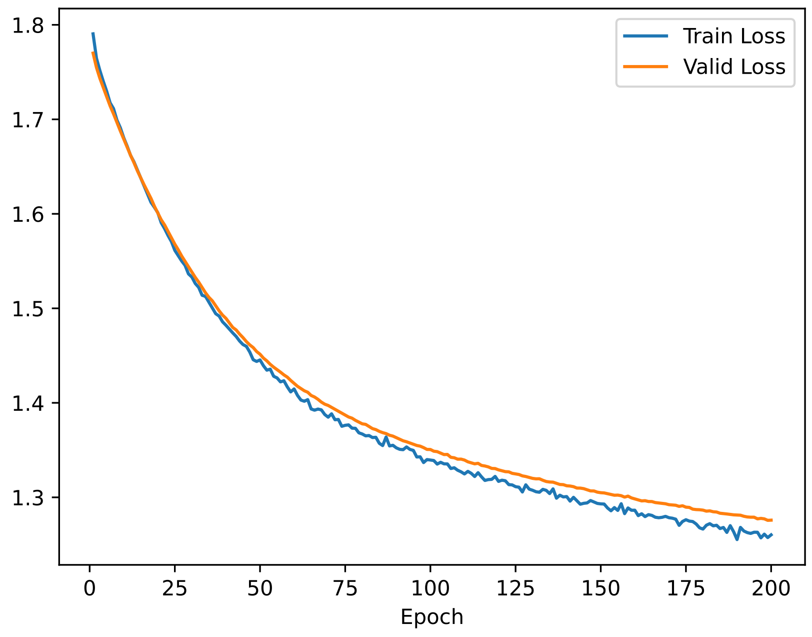

Figure 18. Train loss and validation loss curves for a momentum of 0.0.

For a momentum of 0.9:

The final validation accuracy was 0.6019 and the final test accuracy was 0.5993.

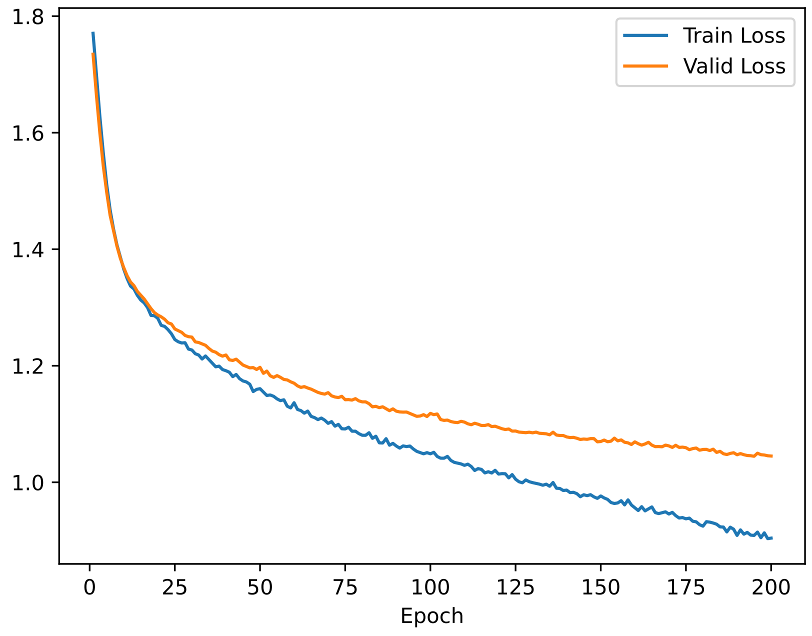

Figure 19. Train loss and validation loss curves for a momentum of 0.9.

When momentum is 0.9, the training process becomes faster because momentum helps the optimizer move more effectively through the loss landscape, smoothing out small oscillations and speeding up progress toward better solutions. This results in higher validation and test accuracies, with values of 0.6019 and 0.5993, compared to the slower and less efficient training process when momentum is set to 0.0, which achieves only 0.5014 and 0.5203. Moreover, With a value of 0.9, the model fits the training data more closely, which creates a noticeable gap between the training and validation losses, which is an indicator of overfitting. This happens because the rapid convergence allows the model to focus more on the training data, sometimes at the expense of its ability to generalize well to new data. On the other hand, when momentum is set to 0.0, the training and validation losses remain closely aligned, showing little to no overfitting. But this alignment comes at the cost of slower learning and weaker overall performance since the optimizer lacks the acceleration provided by momentum.

Question 3.

1.

Let w_i^T be the ith row of $\mathbf{W}$. Then,

$$h_i = g(w_i^T\mathbf{x} + b_i) = (w_i^T\mathbf{x} + b_i)(1 - (w_i^T\mathbf{x} + b_i)) = (w_i^T\mathbf{x} + b_i) - (w_i^T\mathbf{x} + b_i)^2 =$$

$$= \left(\sum_{j=1}^{D} w_{ij}x_j + b_i\right) - \left(\sum_{j=1}^{D} w_{ij}x_j + b_i\right)^2 = \left(\sum_{j=1}^{D} w_{ij}x_j + b_i\right) - \left(\sum_{j=1}^{D} w_{ij}^2x_j^2 + \sum_{j\neq k} w_{ij}w_{ik}x_jx_k + 2b_i\sum_{j=1}^{D} w_{ij}x_j + b_i^2\right) =$$

$$= (b_i - b_i^2) + (1-2b_i)\sum_{j=1}^{D} w_{ij}x_j - \sum_{j=1}^{D} w_{ij}^2x_j^2 - \sum_{j\neq k} w_{ij}w_{ik}x_jx_k = (b_i - b_i^2) + (1-2b_i)\sum_{j=1}^{D} w_{ij}x_j - \sum_{j=1}^{D} w_{ij}^2x_j^2 - 2\sum_{1\leq j<k\leq D} w_{ij}w_{ik}x_jx_k.$$

Thus, taking $\phi : \mathbb{R}^D \to \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$ such that

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_D \\ x_1^2 \\ \vdots \\ x_D^2 \\ 2x_1x_2 \\ \vdots \\ 2x_{D-1}x_D \end{bmatrix},$$

and taking

$$A_{\Theta_i} = \begin{bmatrix} b_i - b_i^2 \\ (1-2b_i)w_{i1} \\ \vdots \\ (1-2b_i)w_{iD} \\ -w_{i1}^2 \\ \vdots \\ -w_{iD}^2 \\ -w_{i1}w_{i2} \\ \vdots \\ -w_{i(D-1)}w_{iD} \end{bmatrix} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}},$$

then we can write $h_i = A_{\Theta_i}^T\phi(x)$ and, hence,

$$h = A_\Theta\ \phi(x), \tag{1}$$

with

$$A_\Theta = \begin{bmatrix} A_{\Theta_1}^T \\ \vdots \\ A_{\Theta_K}^T \end{bmatrix} \in \mathbb{R}^{K\times\frac{(D+1)(D+2)}{2}}.$$

2.

It is important to notice that

$$\hat{y} = v^Th + v_0 \overset{(1)}{=} v^T(A_\Theta\ \phi(x)) + v_0 = c_\Theta^T\ \phi(x) \implies c_\Theta^T = v^TA_\Theta + [v_0\ 0\ ...\ 0] \implies c_\Theta = A_\Theta^Tv + \begin{bmatrix} v_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

where

$$v_0 \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}.$$

Furthermore, since c_Θ is not a linear function (because A_Θ involves quadratic terms), then the model is nonlinear in terms of the original parameters Θ.

3.

$$\hat{y} = v^T h + v_0 = \sum_{i=1}^{K} v_i h_i + v_0 = \sum_{i=1}^{K} v_i\left((w_i^T x + b_i) - (w_i^T x + b_i)^2\right) + v_0 = \sum_{i=1}^{K} v_i\left((w_i^T x + b_i) - (x^T w_i + b_i)(w_i^T x + b_i)\right) +$$
$$+ v_0 = \sum_{i=1}^{K} v_i\left((w_i^T x + b_i) - (x^T w_i w_i^T x + x^T w_i b_i + b_i w_i^T x + b_i b_i)\right) + v_0 = \sum_{i=1}^{K} v_i\left(w_i^T x + b_i - x^T w_i w_i^T x - 2x^T w_i b_i - b_i^2\right) + v_0 =$$
$$= \sum_{i=1}^{K} v_i\left(-x^T w_i w_i^T x + w_i^T x - 2x^T w_i b_i + b_i - b_i^2\right) + v_0 = \sum_{i=1}^{K} v_i\left(-x^T w_i w_i^T x + x^T w_i - 2x^T w_i b_i + b_i - b_i^2\right) + v_0 = \sum_{i=1}^{K} v_i\left(-x^T w_i\right.$$
$$\left. w_i^T x + (1-2b_i)x^T w_i + b_i - b_i^2\right) + v_0 = \sum_{i=1}^{K} -v_i x^T w_i w_i^T x + \sum_{i=1}^{K} v_i (1-2b_i) x^T w + \sum_{i=1}^{K} v_i (b_i - b_i^2) + v_0 = *$$

Take

$B_i = b_i - b_i^2$; $C_i = (1-2b_i)w_i^T$; $D_i = -w_i w_i^T$, $\longrightarrow w_i = \begin{bmatrix} w_{i1} \\ \vdots \\ w_{iD} \end{bmatrix}$

where w_i^T is i-th row of W (dim $= D+1$). Then,

$$C_i = (1-2b_i)w_i = (1-2b_i)\begin{bmatrix} w_{i1} \\ \vdots \\ w_{iD} \end{bmatrix} = \begin{bmatrix} (1-2b_i)w_{i1} \\ \vdots \\ (1-2b_i)w_{iD} \end{bmatrix} \in \mathbb{R}^{D\times 1}$$

$$D_i = -\begin{bmatrix} w_{i1} \\ \vdots \\ w_{iD} \end{bmatrix}\begin{bmatrix} w_{i1} & \cdots & w_{iD} \end{bmatrix} = -\begin{bmatrix} w_{i1}^2 & w_{i1}w_{i2} & \cdots & w_{i1}w_{iD} \\ w_{i2}w_{i1} & w_{i2}^2 & \cdots & w_{i2}w_{iD} \\ \vdots & \vdots & \ddots & \vdots \\ w_{iD}w_{i1} & w_{iD}w_{i2} & \cdots & w_{iD}^2 \end{bmatrix} \in \mathbb{R}^{D\times D}$$

Thus,

$$* = x^T \sum_{i=1}^{K} \left(v_i(-w_i w_i^T)\right) x + x^T \sum_{i=1}^{K} \left(v_i(1-2b_i)w_i\right) + \sum_{i=1}^{K} v_i(b_i - b_i^2) + v_0 = x^T \sum_{i=1}^{K} v_i D_i x + x^T \sum_{i=1}^{K} v_i C_i + \sum_{i=1}^{K} v_i B_i + v_0 = **$$

Taking now

$M = \sum_{i=1}^{K} v_i D_i$; $N = \sum_{i=1}^{K} v_i C_i$; $O = \sum_{i=1}^{K} v_i B_i$,

that is,

$$M = \begin{bmatrix} \sum_{i=1}^{K} v_i(-w_{i1}^2) & \sum_{i=1}^{K} v_i(-w_{i1}w_{i2}) & \cdots & \sum_{i=1}^{K} v_i(-w_{i1}w_{iD}) \\ \sum_{i=1}^{K} v_i(-w_{i2}w_{i1}) & \sum_{i=1}^{K} v_i(-w_{i2}^2) & \cdots & \sum_{i=1}^{K} v_i(-w_{i2}w_{iD}) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{K} v_i(-w_{iD}w_{i1}) & \sum_{i=1}^{K} v_i(-w_{iD}w_{i2}) & \cdots & \sum_{i=1}^{K} v_i(-w_{iD}^2) \end{bmatrix} \in \mathbb{R}^{D\times D} \longrightarrow$$

Similar to matrix D_i, matrix M is symmetric and has $\frac{(D+1)D}{2}$ different entries.
↳ Consider the main diagonal and the lower triangular part of matrix M.

$$N = \begin{bmatrix} \sum_{i=1}^{K} v_i(1-2b_i)w_{i1} \\ \vdots \\ \sum_{i=1}^{K} v_i(1-2b_i)w_{iD} \end{bmatrix} \in \mathbb{R}^{D\times 1}$$

$$O = \sum_{i=1}^{K} v_i(b_i - b_i^2) \in \mathbb{R}$$

then,

$** = x^T M x + x^T N + O + v_0$. Let $Q = O + v_0$.

Let c be the vector containing $\underbrace{\text{all the different entries}}$ of the matrices M, N and Q, that is,

→ Elements of main diagonal and the lower triangular part of the matrix M.

$$c = \begin{bmatrix} \sum_{i=1}^{K} v_i(b_i - b_i^2) + v_0 \\ \sum_{i=1}^{K} v_i(1-2b_i)w_{i1} \\ \vdots \\ \sum_{i=1}^{K} v_i(1-2b_i)w_{iD} \\ \sum_{i=1}^{K} v_i(-w_{i1}^2) \\ \vdots \\ \sum_{i=1}^{K} v_i(-w_{iD}^2) \\ \sum_{i=1}^{K} v_i(-w_{i1}w_{i2}) \\ \vdots \\ \sum_{i=1}^{K} v_i(-w_{i(D-1)}w_{iD}) \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} D+1 \\ \left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\\a\end{matrix}}\right\} D + \frac{D(D-1)}{2} = \frac{2D + D(D-1)}{2} = \frac{D(2+D-1)}{2} = \frac{D(D+1)}{2} \end{matrix}$$

$$D+1+\frac{D(D+1)}{2} = D+1\left(1+\frac{D}{2}\right) = \frac{(D+1)(D+2)}{2} \Rightarrow c \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}} \checkmark$$

Remembering that $\underset{\text{Ex. 3.1}}{\phi(x)} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_D \\ x_1^2 \end{bmatrix}$, then, in fact, $\hat{y} = c^T \phi(x) = \sum_{i=1}^{K} v_i(b_i - b_i^2) + v_0 + \sum_{i=1}^{K} v_i(1-2b_i)w_{i1}x_1 + \ldots + \sum_{i=1}^{K} v_i(1-2b_i)w_{iD}x_D + \sum_{i=1}^{K} v_i(-w_{i1}^2)x_1^2 + \ldots + \sum_{i=1}^{K} v_i(-w_{iD}^2)x_D^2 + \sum_{i=1}^{K} v_i(-w_{i1}w_{i2})\,2\cdot$

$$\begin{bmatrix} \vdots \\ x_D \\ x_1^2 \\ \vdots \\ x_D^2 \\ 2x_1x_2 \\ \vdots \\ 2x_{D-1}x_D \end{bmatrix}$$

$$x_1x_2 + \dots + \sum_{i=1}^{K} v_i(-w_{i(D-1)}w_{iD})\, 2 \cdot x_{(D-1)}x_D \quad \checkmark$$

The vector c has $\frac{(D+1)(D+2)}{2}$ entries and, hence, we can construct matrices $D \times D$, $D \times 1$ and 1×1. On the other hand, we need to check if it is possible to construct the vector c at the expense of the other matrices, that is, whether these matrices exist for all values of c. In fact, for $K < D$, we cannot guarantee that there is a choice of $\overset{\text{the}}{\vee}$ original parameters Θ such that $\|c_\Theta - c\| < \varepsilon$, for any $\varepsilon > 0$. For instance, for $1 = K < D = 2$, we can consider the system:

$$\begin{cases} v_1(b_1 - b_1^2) + w_0 = 1 \\ v_1(1-2b_1)w_{11} = 1 \\ v_1(1-2b_1)w_{12} = 1 \\ v_1(-w_{11}^2) = 0 \\ v_1(-w_{12}^2) = 1 \\ v_1(-w_{11}w_{12}) = 1 \end{cases}$$

Impossible

Thus, if $K \geq D$, it is possible obtain Θ from c, following the complementary method to the one presented.

4.

Assume that the matrix $\mathbf{X} \in \mathbb{R}^{N\times\frac{(D+1)(D+2)}{2}}$ whose rows are the feature vectors $\{\phi(\mathbf{x}_n)\}_{n=1}^N$ has rank $\frac{(D+1)(D+2)}{2}$. Thus, all columns of $\mathbf{X}$ are linearly independent. Let $\mathbf{y} = (y_1, ..., y_N)$. Then, the squared loss can be rewritten as

$$L(c_\Theta; \mathcal{D}) = \frac{1}{2}\sum_{n=1}^{N}(\hat{y}(x_n; c_\Theta) - y_n)^2 = \frac{1}{2}\sum_{n=1}^{N}(c_\Theta^T\phi(x_n) - y_n)^2 = \frac{1}{2}||\mathbf{X}c_\Theta - \mathbf{y}||_F^2. \qquad \text{(Least Squares Problem)} \qquad (2)$$

Therefore, we want to minimize the lost function (2) with respect to c_Θ. Since this is a least squares problem, then the solution in given by

$$\frac{\partial L(c_\Theta, \mathcal{D})}{\partial c_\Theta} = \mathbf{X}^T(\mathbf{X}c_\Theta - \mathbf{y}) = 0 \Leftrightarrow c_\Theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

and, therefore, we can find a closed form solution $\hat{c}_\Theta$:

$$\hat{c}_\Theta = \mathbf{X}^+\mathbf{y} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

It is important to notice that, since, by assumption, all columns of $\mathbf{X}$ are linearly independent, then $\mathbf{X}^T\mathbf{X}$ is invertible (and, therefore, $(\mathbf{X}^T\mathbf{X})^{-1}$ exists) and $\hat{c}_\Theta$ is a global minimum.

Since the neural network has the activation function $g(z) = z(1 - z)$ (which it is a concave function), it allows the model to be tackled as a linear model after a reparameterization. In particular, taking into account the reparameterizations of the part 2, it is possible to reduce the neural network to a linear least squares regression problem in the feature space $\phi(\mathbf{x})$. As a result of this, we can rewrite the squared loss function as a convex function of c_Θ, reducing the problem to a optimization problem which enable us to find the global minimizer. Thus, in conclusion, what makes our problem special is the considered activation function, which presupposes a linear relation between y and $\phi(x)$.