Deep Learning

1st Semester 2022/23

Homework 1

Authors:

Duarte Venâncio Leão Ribeiro da Silva, Nº 93243
Gonçalo Da Silva Dias Moura de Mesquita , Nº 94196

Group: 66

December 23, 2022

Contents

Contribution

Duarte Silva took the lead in solving Q1, Gonçalo Mesquita implemented the code for Q2, and both solved Q3 together. And in question 3.3 we collaborated with group 40.

1 Question 1

1.1

1.1.1

In figure 1 it is possible to see that the percetron didn't converge, i.e. training on a bigger number of epochs does not translate to better. Therefor if the number of epochs increases, it won´t affect the accuracy in the test.

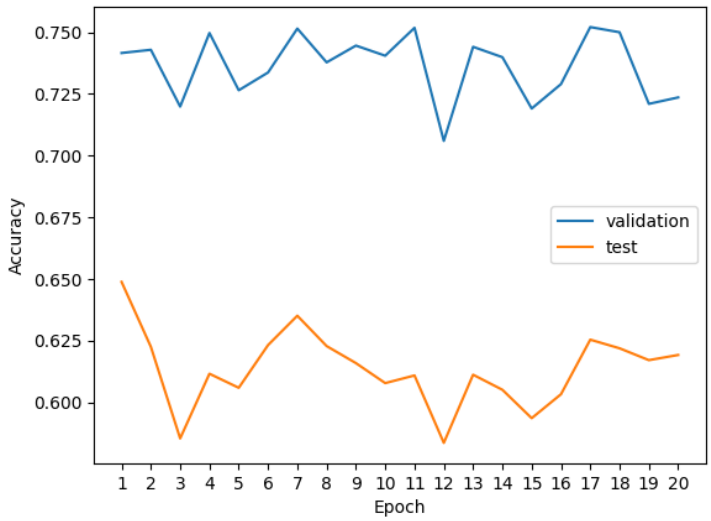


Figure 1: Perceptron accuracy on validation and test set

Epoch	Validation Accuracy	Test Accuracy
1	0.7416	0.6489
10	0.7405	0.6079
15	0.7191	0.5937
20	0.7236	0.6193

Table 1: Accuracy of simple Perceptron model

1.1.2

In figure 2 it is possible to observe that the logistic regression model converges a small amount, training on a bigger number of epochs will increase the accuracy in the test set.

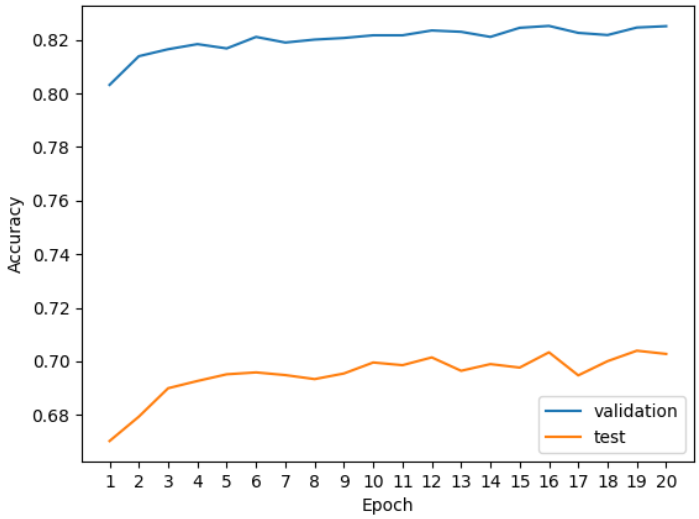


Figure 2: Logistic regression accuracy on validation and test set

Epoch	Validation Accuracy	Test Accuracy
1	0.8032	0.6703
10	0.8217	0.6996
15	0.8245	0.6977
20	0.8251	0.7028

Table 2: Accuracy of Logistic Regression model

1.2

1.2.1

Being a linear classifier, a simple perceptron can only learn linear decision boundaries. For problems like these where the data might not be linearly separable(e.g. different characters that have similar feature values), a multi class multi-layer perceptron (MLP) with non-linear activations can learn non-linear decision boundaries, making it more expressive and more adapted. Since this particular task can entail complicated patterns and relationships between the many classes of characters, the MLP can handle it better thanks to the added complexity of non-linear activation functions.

A MLP with a linear activations behaves like a simple perceptron and can only learn linear decision boundaries. This means that no matter how many hidden layers the MLP has, it will fail to distinguish samples that are not linearly separable and will have no advantage over a simple perceptron.

1.2.2

In figure 3 it is possible to observe that a multi layer perceptron model will converge, training on a bigger number of epochs will increase the accuracy in the test set. It is important to note, however, that simply increasing the number of epochs does not guarantee an improvement in the model's performance. Indeed, if the model is already fully converged, adding more epochs may result in over fitting, in which the model performs well on training data but poorly on test data.

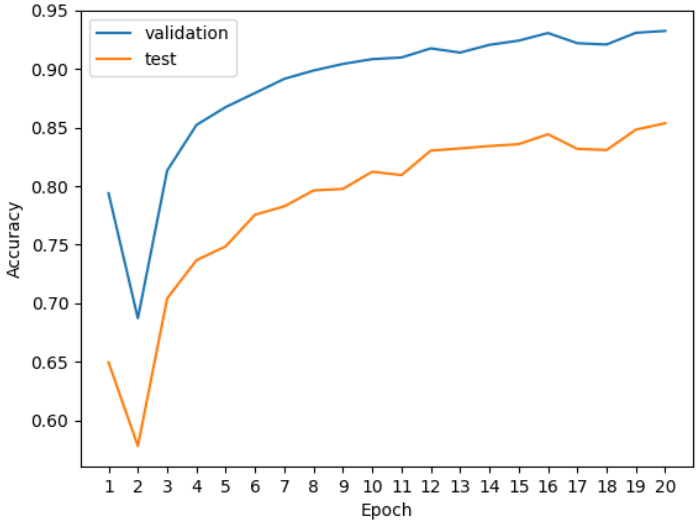


Figure 3: MLP accuracy on validation and test set

Epoch	Validation Accuracy	Test Accuracy
1	0.7938	0.6495
10	0.9084	0.8123
15	0.9242	0.8358
20	0.9325	0.8537

Table 3: Accuracy of MLP model

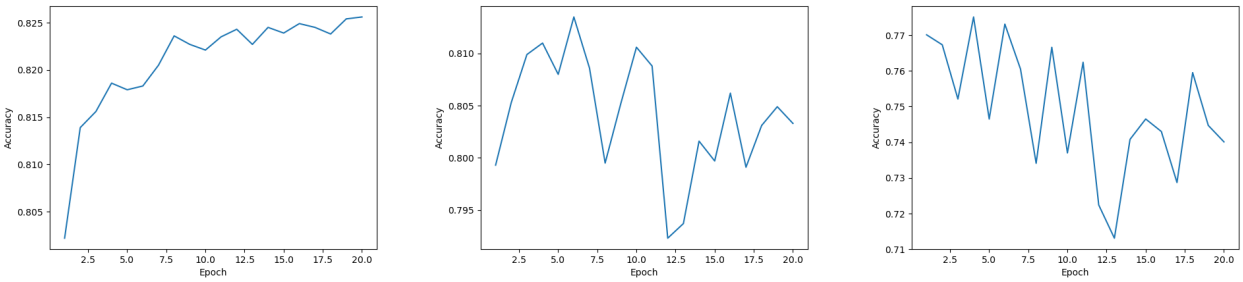
2 Question 2

2.1

In this section we implemented a logistic regression model using Pytorch. In order to better understand the hyperparameters and how they can effect the final prediction we tested this model with different values of learning rate, respectively [0.1, 0.01, 0.001]. The results obtain are presented in the table 4. Consequently we conclude that the best learning rate for this logistic regression model is 0.001 because it produces the most accurate prediction in the test set. The size of the steps the model takes to discover the best solution is controlled by the learning rate, a hyperparameter. As we can observe in the figure 4, a higher step leads to faster convergence, however it can also cause the model to miss the optimal solution, as it happens for the learning rate 0.1 and 0.01. A lower step leads to slower convergence, so it needs more iterations to find the optimal solution. In this case, the learning rate of 0.001 gives the best trade off between convergence speed and accuracy.

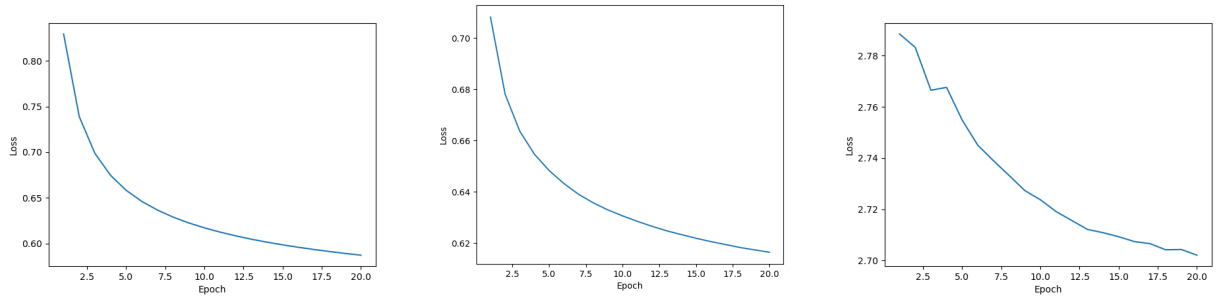
Learning rate	Accuracy Test	Accuracy Validation	Training loss
0.1	0.6128	0.7401	2.701
0.01	0.6806	0.8032	0.6114
0.001	0.7069	0.8256	0.5874

Table 4: Results of the Logistic Regression model



(a) (b) (c)

Figure 4: Validation accuracy: (a) learning rate = 0.001, (b) learning rate = 0.01, (c) learning rate = 0.1.



(a) (b) (c)

Figure 5: Training Loss: (a) learning rate = 0.001, (b) learning rate = 0.01, (c) learning rate = 0.1.

2.2

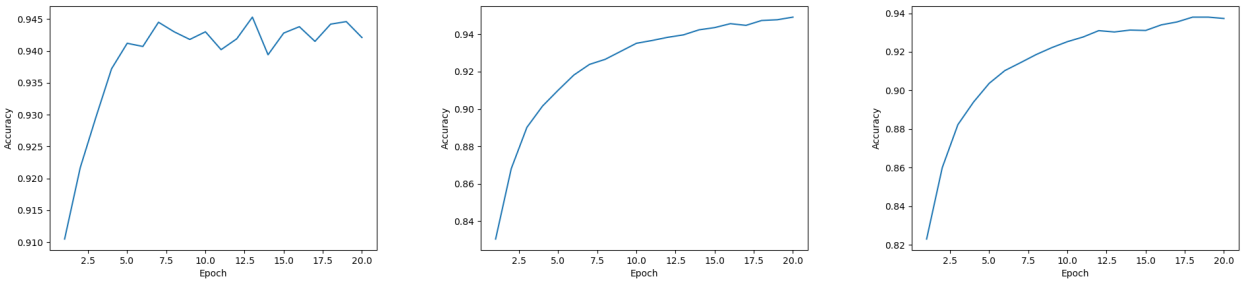
In this section we implemented a feed forward neural network with 1 layer using Pytorch. In order to get the best accuracy possible in the test set we had to tune the hyperparameters. Therefore we tested the model with different values of learning rate, hidden layers size, drop out probability and activation function. However, only one of these hyperparameters should be changed at a time, with the others remaining as default.

Following all of the combinations, the results obtained are presented in the table 5, allowing us to conclude that changing the hidden size hyperparameter to 200 provides the best accuracy in the test set therefore the best result. In the figure 7, we can also notice the best three results of these hyperparameter combinations as a function of the iterations.
This model with a lower learning rate leads to slower convergence, so it needs more iterations to find the optimal solution. Having only 20 iterations to run the model will make the learning

rate 0.001 to slow. When faced with a complex problem involving the classification of numbers in images, the model would produce good results with 100 neurons, however the accuracy of the test set was always higher with 200 neurons. The dropout probability is a hyperparameter that determines the probability of a neuron being set to zero during training, the best value for this parameter was 0.3 .A high dropout probability has the disadvantage of significantly slowing down the training process. This is due to the fact that the model must learn to make predictions using a smaller subset of neurons at each training iteration, which may require more training iterations to achieve convergence. As a result, the training loss with a 0.5 dropout probability was always higher and the accuracy in the test set was always lower. We also conclude that that the ***Relu*** activation function worked better with this type of data. All these conclusion were driven from the table 5 that exhibits the results of all combinations made to test the hyperparameters.

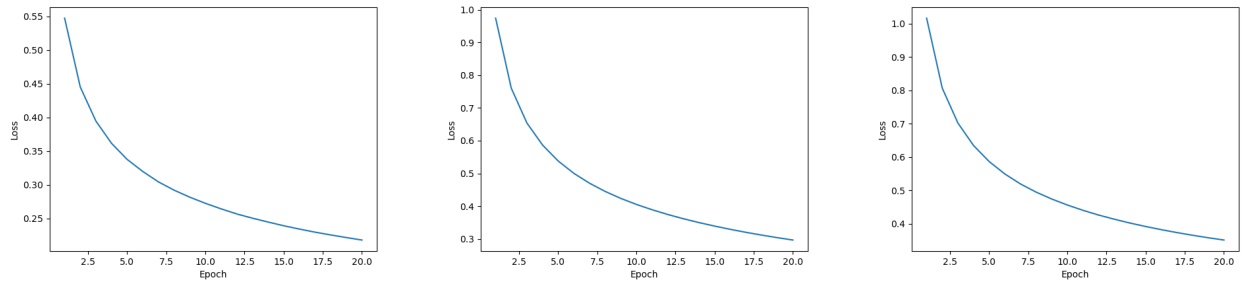
Hyperparameter	Accuracy Test	Accuracy Validation	Training loss
Default values	0.8614	0.9373	0.3512
Learning Rate: 0.1	0.8733	0.9421	0.2181
Learning Rate: 0.001	0.7397	0.8620	0.7934
Hidden size: 200	**0.8796**	**0.9492**	**0.2964**
Dropout: 0.5	0.8439	0.9281	0.4409
Activation: tanh	0.8237	0.9373	0.3512

Table 5: Results of the feed forward neural network with different hyperparameters combinations.



(a) (b) (c)

Figure 6: Accuracy validation of the 3 best configuration: (a) learning rate: 0.1, (b) hidden size: 200, (c) default values



(a) (b) (c)

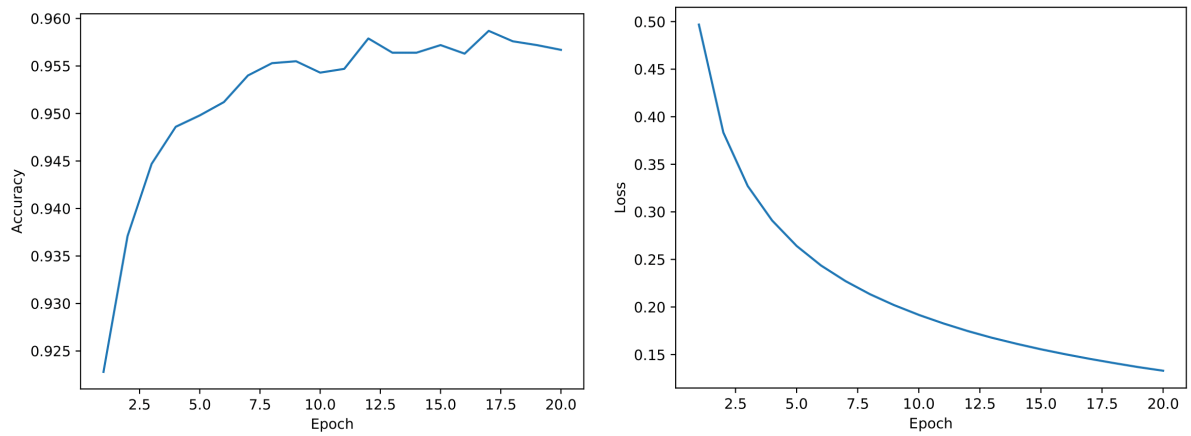
Figure 7: Training Loss of the 3 best configuration: (a) learning rate: 0.1, (b) hidden size: 200, (c) default values

We moved a little bit further out of curiosity. We put this model to the test using every feasible set of hyperparameters. to achieve the finest model tuning and, as a result, the best outcomes. The results of some of the combinations used to test the hyperparameters are shown in Table 6. We can observe by he table and the figure 8 that the result with best accuracy in test set is the following. We can observe from the table 6 and the figure 8 that the combination of hyperparameters with the best results are the following.

- Learning rate: 0.1.
- Hidden size layer: 200.
- Dropout probability: 0.3.
- Activation function: ReLu.

Learning rate	Hidden size	Dropout	Activation	Acc Test	Acc Validation	Training loss
0.1	100	0.3	Relu	0.8733	0.9410	0.2181
0.1	100	0.3	Tanh	0.8565	0.9314	0.3000
0.1	**200**	**0.3**	**Relu**	**0.8990**	**0.9567**	**0.1330**
0.01	200	0.3	Relu	0.8796	0.9492	0.2900
0.1	200	0.5	Relu	0.8900	0.9544	0.2300

Table 6: Results of the feed forward neural network with different hyperparameters combination.



(a) (b)

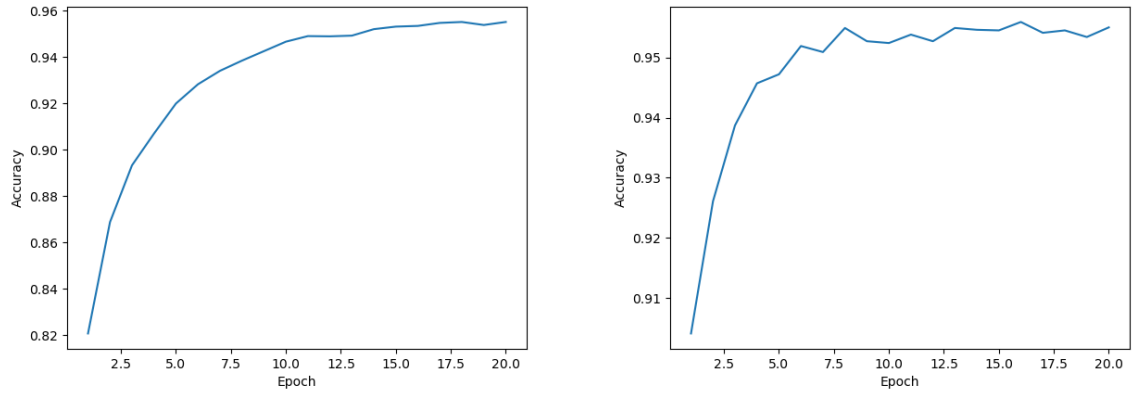
Figure 8: Results of the best configuration: (a) Accuracy of the validation set, (b) Loss of the training set

2.3

Applying the best configuration of the hyperparameters shown in the section 2.2 table 5 and increasing the number of layers to the same feed forward neural network, we obtain the results observed in table 7 and the figure 9.

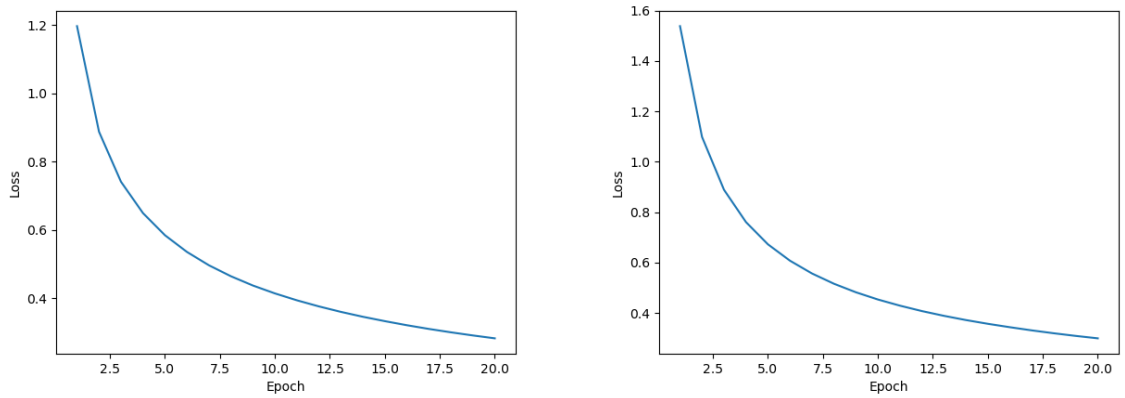
Number of layers	Acc Test	Acc Validation	Training loss
2	0.8955	0.9552	0.2830
3	0.8911	0.9537	0.2991

Table 7: Results of with more than 1 layer



(a) (b)

Figure 9: Validation accuracy: (a) Number layers = 2, (b) Number layers = 3



(a) (b)

Figure 10: Training loss: (a) Number layers = 2, (b) Number layers = 3

If a two layer feed forward neural network can recognize the patterns in the data just as effectively as a three layer feed forward neural network, this means that the problem isn´t complex enough to use a more deeper Neural Network. Taking this into account and observing the table 7, figure 9 and 10 we conclude that the results are better in two layer feed forward neural network.

3 Question 3

3.1

To show that h can be written as $h = A_\Theta \phi(x)$, where ϕ is a certain feature transformation from R^D to $R^{D(D+1)/2}$ and $A_\Theta \in R^{K \times D(D+1)/2}$, we can follow the following steps:

We start by writing h as $h = g(Wx)$, where g is the quadratic activation function and W is the matrix of model parameters.

Since the quadratic activation function is given by $g(z) = z^2$, we can write the elements of h as:

$$h_i = (Wx)_i^2 = (\sum_{j=1}^{D} W_{ij}x_j)^2$$

Next, we can expand the latter to write $(\sum_{j=1}^{D} W_{ij}x_j)^2$ as:

$$\begin{aligned}
(\sum_{j=1}^{D} W_{ij}x_j)^2 &= \sum_{j=1}^{D}\sum_{k=1}^{D} W_{ij}W_{ik}x_jx_k \\
&= \sum_{j=1}^{D} W_{ij}^2x_j^2 + \sum_{j=1}^{D}\sum_{\substack{k=1\\k\neq j}}^{D} W_{ij}W_{ik}x_jx_k \\
&= \sum_{j=1}^{D} W_{ij}^2x_j^2 + 2\sum_{j=1}^{D-1}\sum_{\substack{k=1\\k>j}}^{D} W_{ij}W_{ik}x_jx_k \\
&= \sum_{j=1}^{D} W_{ij}^2x_j^2 + 2\sum_{j=1}^{D-1}\sum_{k=j+1}^{D} W_{ij}W_{ik}x_jx_k
\end{aligned}$$

We can now define the feature transformation ϕ as follows:

$$\phi(x) = \begin{bmatrix} x_1^2 & \dots & x_D^2 & \sqrt{2}x_1x_2 & \dots & \sqrt{2}x_1x_D & \sqrt{2}x_2x_3 & \dots & \sqrt{2}x_2x_D & \dots & \sqrt{2}x_{D-1}x_D \end{bmatrix}^\top$$

This transformation maps the input vector x to a vector of length $D(D+1)/2$ that contains all the squared and cross-products of the elements of x.

We can now write h as a linear combination of the elements of $\phi(x)$ by setting the coefficients of the linear combination to be the elements of the matrix A_Θ:

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

where A_Θ is a $K \times D(D+1)/2$ matrix defined as follows:

$$A_\Theta = \begin{pmatrix} W_{11}^2 & \cdots & W_{1D}^2 & \sqrt{2}W_{11}W_{12} & \cdots & \sqrt{2}W_{11}W_{1D} & \sqrt{2}W_{12}W_{13} & \cdots & \sqrt{2}W_{12}W_{1D} & \cdots & \sqrt{2}W_{1D-1}W_{1D} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ W_{K1}^2 & \cdots & W_{KD}^2 & \sqrt{2}W_{K1}W_{K2} & \cdots & \sqrt{2}W_{K1}W_{KD} & \sqrt{2}W_{K2}W_{K3} & \cdots & \sqrt{2}W_{K2}W_{KD} & \cdots & \sqrt{2}W_{KD-1}W_{KD} \end{pmatrix}$$

This shows that h can be written as $h = A_\Theta\phi(x)$.

3.2

The predicted output $\hat{y}$ is a linear transformation of $\phi(x)$ using the coefficients c_Θ, which are given by $c_\Theta = A_\Theta^\top v$, based on the fact that h, the linear transformation of $\phi(x)$, is equal to $A_\Theta\phi(x)$:

$$\hat{y} = v^\top h = v^\top A_\Theta\phi(x) = (A_\Theta^\top v)^\top\phi(x)$$

However, c_Θ is not linear in relation to Θ, as it can be seen by the entries of A_Θ e.g. W_{11}^2, meaning the model is not linear in terms of the original parameters Θ.

3.3

Firstly we started by taking an approach where we used the theory of linear systems(which is presented below). However, we later realised that this theory wouldn't have a direct translation to this problem since the system is quadratic. So after that we took a different approach. Both of the approaches were left in the final report to show the different thought processes that were used throughout this problem.

3.3.1 First approach

If $K \geq D$, then it implies that $K \times D + K > \frac{D(D+1)}{2}$ and since $c_\Theta = A_\Theta^\top v$ this means that for any $c \in R^{D(D+1)/2}$, we need to choose the original parameters Θ in such a way that the following conditions are satisfied:

$$c = A_\Theta^\top v$$

Knowing that $K \times D + K > \frac{D(D+1)}{2}$ this becomes a under-determined system, meaning that, for any vector $c \in R^{D(D+1)/2}$, there exist a choice of the original parameters Θ that satisfy the conditions. From which it follows that we can equivalently parametrize the model with c_Θ instead of Θ. Since the output of the model is linear in terms of c_Θ, then this becomes a linear model in terms of c_Θ.

However if $K < D$, $K \times D + K > \frac{D(D+1)}{2}$ does not always verify, meaning that if $K \times D + K < \frac{D(D+1)}{2}$, the system above becomes over-determined and it seizes to exist a possible choice of parameters Θ that satisfy the conditions. Implying that, for this case, it isn't always possible to find the equivalent parametrization.

3.3.2 Second approach

Given that for $K = 3$ and $D = 2$:

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \Rightarrow w_k^\top w_k = A_k = \begin{bmatrix} w_{k1}^2 & w_{k1}w_{k2} \\ w_{k2}w_{k1} & w_{k2}^2 \end{bmatrix} \tag{1}$$

We obtain the following equation:

$$A_\Theta = \begin{bmatrix} w_{11}^2 & w_{12}^2 & \sqrt{2}w_{11}w_{12} \\ w_{21}^2 & w_{22}^2 & \sqrt{2}w_{21}w_{22} \\ w_{31}^2 & w_{32}^2 & \sqrt{2}w_{31}w_{32} \end{bmatrix} \Rightarrow \|A_\Theta\|_F = \sqrt{(w_{11}^2 + w_{12}^2)^2 + (w_{21}^2 + w_{22}^2) + (w_{31}^2 + w_{32}^2)} \tag{2}$$

$$\|A_k\|_F = \sqrt{w_{k1}^4 + 2w_{k1}w_{k2} + w_{k2}^4} = \sqrt{(w_{k1}^2 + w_{k2}^2)^2} = w_{k1}^2 + w_{k2}^2 \tag{3}$$

By equaling $\|A_{\Theta_k}\|_2$ and $\|A_k\|_F$:

$$\|A_{\Theta_k}\|_2 = \|A_k\|_F \tag{4}$$

$$\Leftrightarrow \sqrt{w_{k1}^4 + 2w_{k1}w_{k2} + w_{k2}^4} = w_{k1}^2 + w_{k2}^2 \tag{5}$$

$$\Leftrightarrow \quad A_{\Theta_k}\begin{bmatrix}1\\1\\0\end{bmatrix} = tr(A_k) \tag{6}$$

$$\Rightarrow \quad A_{\Theta}\begin{bmatrix}1\\1\\0\end{bmatrix} = \begin{bmatrix}tr(A_1)\\tr(A_2)\\tr(A_3)\end{bmatrix} \tag{7}$$

$$\Leftrightarrow \quad A_{\Theta}\begin{bmatrix}1\\1\\0\end{bmatrix} = \begin{bmatrix} w_1w_1^\top \\ w_2w_2^\top \\ w_3w_3^\top \end{bmatrix} \tag{8}$$

Where:

$$W = \begin{bmatrix} -w_1- \\ -w_2- \\ -w_3- \end{bmatrix} \tag{9}$$

Finally, we obtain:

$$A_{\Theta}\begin{bmatrix}1\\1\\0\end{bmatrix} = \begin{bmatrix}w_1 & 0^\top & 0^\top \\ 0^\top & w_2 & 0^\top \\ 0^\top & 0^\top & w_3\end{bmatrix}\begin{bmatrix}w_1 & w_2 & w_3\end{bmatrix}^\top \tag{10}$$

$$\Leftrightarrow \quad \begin{bmatrix}1\\1\\0\end{bmatrix}^\top A_{\Theta}^\top = \begin{bmatrix}w_1 & w_2 & w_3\end{bmatrix}\begin{bmatrix}w_1^\top & 0 & 0 \\ 0 & w_2^\top & 0 \\ 0 & 0 & w_3^\top\end{bmatrix} \tag{11}$$

$$\Leftrightarrow \quad \begin{bmatrix}1 & 1 & 0\end{bmatrix} A_{\Theta}^\top v = \begin{bmatrix}w_1 & w_2 & w_3\end{bmatrix}\begin{bmatrix}w_1^\top & 0 & 0 \\ 0 & w_2^\top & 0 \\ 0 & 0 & w_3^\top\end{bmatrix} v \tag{12}$$

$$\Leftrightarrow \quad \begin{bmatrix}\underbrace{1 \quad 1}_{\text{size } D} & \underbrace{0}_{\text{size } \frac{D\times(D-1)}{2}}\end{bmatrix} c_\Theta = \begin{bmatrix}w_1 & w_2 & w_3\end{bmatrix}\begin{bmatrix}w_1^\top & 0 & 0 \\ 0 & w_2^\top & 0 \\ 0 & 0 & w_3^\top\end{bmatrix} v \tag{13}$$

From this, we get a direct relationship between c_Θ and the pair (W, v). For simplicity of visualization this relationship was derived for $K = 3$ and $D = 2$, as previously said, but it's clear to see that this extends for all possible values of K and D. However we couldn't reach any conclusion about the necessity of $K \geq D$, since equation 13 seems to always have a solution independently of the values of K and D.

3.4

Since $\hat{y}$ is a linear transformation of $\phi(x)$ using coefficients c_Θ, this output layer is equivalent to a linear regression task. Given that X is full column rank and the loss, $L(c_\Theta; \mathcal{D})$, is quadratic, it is possible to use the normal equations to find a closed-form solution for c_Θ :

$$c_\Theta = (X^\top X)^{-1} X^\top y$$

The fact that global minimization is usually intractable for feedforward neural networks is due to the non-linear activation functions that are typically used. These non-linear functions

can result in loss functions that are highly complex and may not have a unique global minimum. What makes our problem special is that, since $\hat{y}$ is linear in c_Θ and X is full column rank the problem reduces to a Ordinary Least Squares linear regression problem, which has a close form solution to find the global minimum.