TÉCNICO LISBOA

Deep Learning

MEEC

Homework 1

Authors:

Diogo José Pereira Araújo (93906) diogoparaujo@tecnico.ulisboa.pt
Maria Margarida Domingues Rita (93130) margarida.rita@tecnico.ulisboa.pt

Group 40

2022/2023 – 1^{st} **Semester, P2**

Individual contribution

Both members contributed equally to the resolution of this homework assignment. Maria Margarida implemented anwser questions Q1-1, Q1-2-a and Q2-1. Diogo did questions Q1-2-b, Q2-2 and Q2-3. Both members did Q3. Q3-3 was discussed in colaboration with group 66.

Question 1

1.

(a)

The perceptron accuracy on the validation and test sets can be seen in Figure 1 as a function of the number of epochs. For the last epoch, the validation set's accuracy was 0.7236 and the test set's accuracy was 0.6193. For the first epoch, the validation set had an accuracy of 0.7416 and the test set had an accuracy of 0.6489. As illustrated in Figure 1, this simple perceptron did not converge, justifying the fact that the accuracy for the last epoch is practically the same as in the first one, for both validation and test sets.

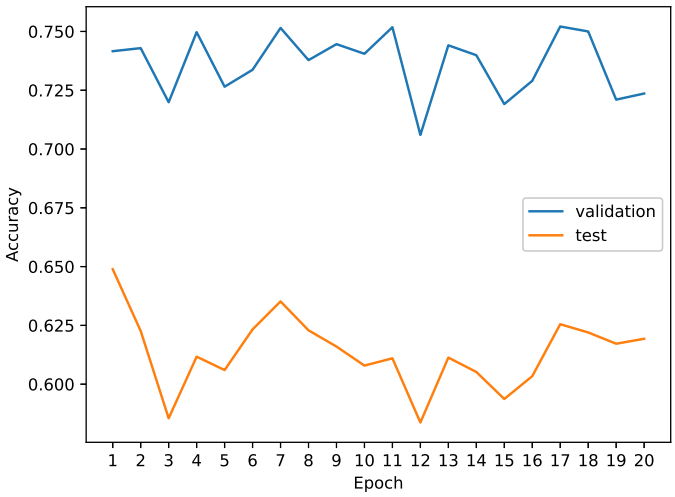


Figure 1: Perceptron validation and test set accuracies for 20 epochs.

(b)

The logistic regression accuracy on the validation and test sets is represented as a function of the number of epochs in Figure 2. For the last epoch, the validation set accuracy was 0.8251 and the test set accuracy was 0.7028.

Comparing the results of the previous question with the results of this one, it is clear that the logistic regression algorithm (with SGD as the training algorithm) outperforms the percpetron for the dataset in question.

There could be many reasons why the logistic regression with SGD perform better that the perceptron for the Kuzushiji-MNIST dataset. Although the SGD updates for logistic regression

and the updates for the perceptron appear similar, there are a few subtle differences that may make the logistic regression algorithm more appropriate for this dataset. One explanation, for instance, could be because only the logistic regression is a probabilistic classifier.

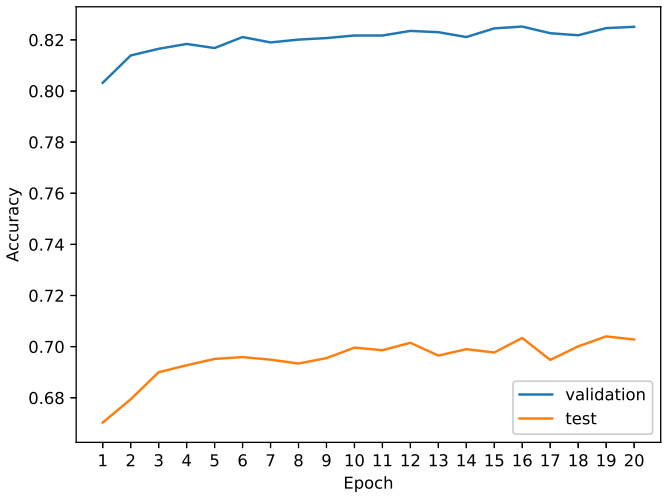


Figure 2: Logistic regression accuracies on the validation and test sets for 20 epochs.

2.

(a)

First, it is important to define the concept of expressiveness in this context. In deep learning, the expressiveness of a neural network represents its ability to approximate any function. It turns out that the network's expressiveness is increased by applying a non-linear activation function at the end of each hidden layer.

Given that each hidden layer computes a representation of its input and propagates it forward, if we use the right non-linear activation function at the end of each hidden layer, then the network will be able to learn more complex non-linear features in the deeper layers. This allows the network to learn non-linear decision boundaries.

Being a simple classifier, the perceptron can only learn linear decision boundaries. This makes the perceptron much more suitable for problems with linearly separable data than with non-linear separable data.

In the case of multi class classification on the Kuzushiji-MNIST dataset, the perceptron will not be able to use correlations between the different pixels that make up an image, i.e., it will not have the ability correlate information between the different elements of a given input. This is not a problem for the multi-layer perceptron. As stated in the previous paragraph, an MLP with a non-linear activation function can learn more complex features in the hidden layers. Given this, as well as the fact that the hidden layers are fully connected, the MLP will be able to extract more interesting features that can only come with the shared information between different pixels within an image.

Using a linear activation function at the end of each hidden layer is equivalent to only having a single linear layer. As previously stated, each hidden layer computes a representation of its

input and propagates it forward. If at the end of each hidden layer a linear activation function is used, then the output of each layer will be a linear function its input. As a result, the network's output will be a linear function of the input. So, in essence, a multi-layer perceptron with many hidden layers and a linear activation function at the end of each hidden layer, has the same expressiveness as a single hidden layer with a linear activation function.

(b)

In Figure 3 it is represented the MLP accuracy on the validation and test sets, as a function of the number of epochs. For the last epoch the accuracy for the validation set was 0.9329 and for the test set was 0.8541.

When compared to previous results, it is clear that the multi-layer perceptron with one hidden layer and a non-linear activation function outperformed the simple perceptron and logistic regression on the Kuzushiji-MNIST dataset. These findings support the previous question's response. Given the dataset's complexity, linear classifiers will perform worse than non-linear classifiers. This happens because, linear classifiers lack the ability to learn non-linear decision boundaries. However, as stated in the previous question, an MLP with a hidden layer and a non-linear activation function can approximate more complex functions, being able to learn non-linear decision boundaries.

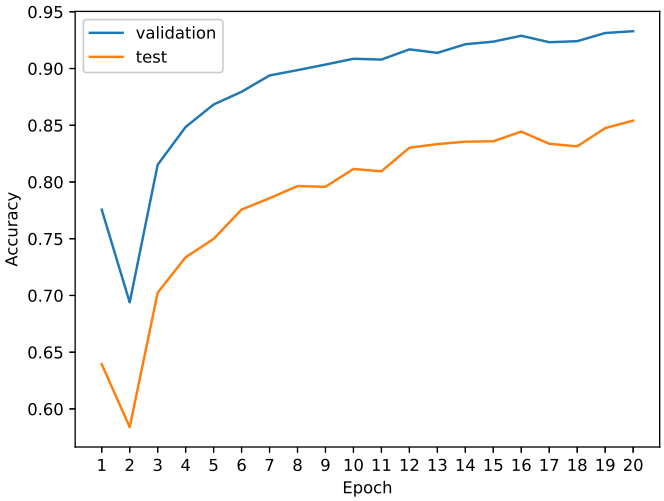


Figure 3: MLP accuracy on the validation and test sets for 20 epochs.

Question 2

1.

In this exercise the group implemented a linear model with a logistic regression, using stochastic gradient descent as training algorithm. The validation and test set accuracy for various hyperparameters are represented in Table 1. Figures 4 and 5 show, respectively, the

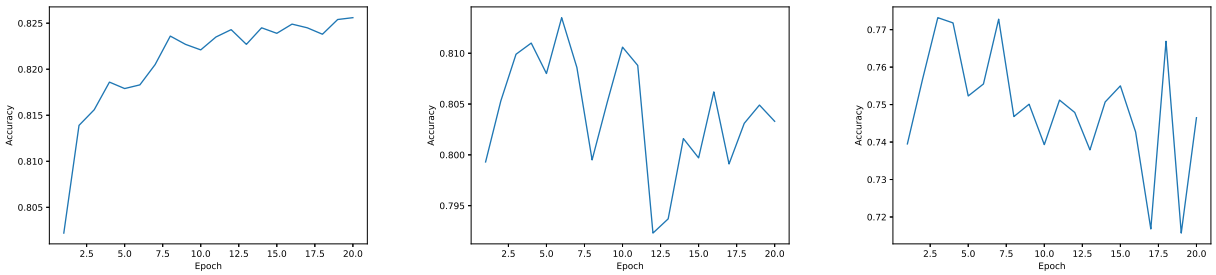
evolution of the accuracy and loss functions for the three best models, i.e., those that achieve the best accuracy on the test set.

As it is possible to see from the results displayed in table 1 using a learning rate of 0.001 produced better outcomes.

Learning rate	Validation set acc.	Test set acc.
0.001	**0.8256**	**0.7019**
0.01	0.8033	0.6806
0.1	0.7465	0.6151

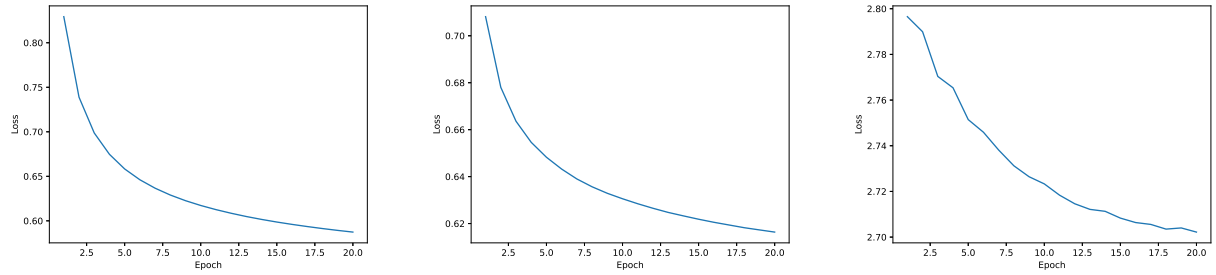
Table 1: Validation and test sets accuracies for the last epoch, for the learning rates: 0.001, 0.01 and 0.1.

Comparing the plots shown in figure 4 it is possible to see the influence that the learning rate has on the model performance. The learning rate determines how soon our model may converge to a local minimum, which means that it controls the rate or speed at which the model learns. Cases (b) and (c) are a perfect example of the learning rate being set too high. This causes the model to oscillate, rather than converging to a local minimum. On the other hand, although the convergence might be slower and the model still oscillates a bit, we can see that in case (a) the model is converging.



(a) (b) (c)

Figure 4: Evolution of the accuracy of the validation set for three different learning rates: (a) 0.01, (b) 0.01, (c) 0.1.



(a) (b) (c)

Figure 5: Evolution of the training loss for three different learning rates: (a) 0.01, (b) 0.01, (c) 0.1.

Figure 5 (c) shows that for a learning rate of 0.1, the training loss grows significantly. This shows that when the learning rate is set too high, the updated weights may reach levels that are excessively high. This increases the likelihood of the model missing the optimal solution. This causes the algorithm to be very unstable as it is possible to observe in figure 4 (c). This emphasizes the importance of properly tuning the learning rate.

2.

In this question the group implemented a feed-forward neural network with a single layer, using the pytorch python library. Table 2 shows the validation and test set accuracy for various hyperparameters. Figures 6 and 7 show the evolution of accuracies and loss functions for the three best models, i.e., those that achieve the best accuracies for the test set.

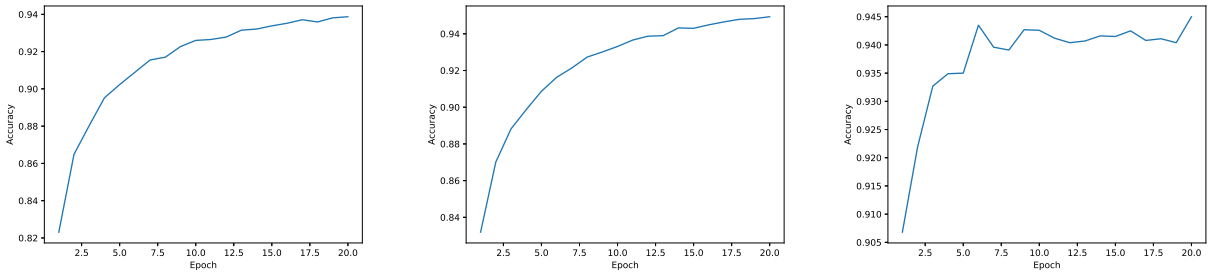
Hyperparameters	Validation set acc.	Test set acc.
All set to default	0.9387	0.8599
learning rate = 0.001	0.8657	0.7451
learning rate = 0.1	0.9450	0.8717
Hidden size = 200	**0.9493**	**0.8814**
Dropout = 0.5	0.9312	0.8409
Activation = tanh	0.9140	0.8257

Table 2: Validation and test sets accuracies for the last epoch, for three different number of hidden layers: 1, 2 and 3.

As shown in Table 2, selecting 200 hidden units yields the best results. This is possible because, given that each instance has 784 elements (784 pixels), using only 100 hidden units would not be enough to capture all of the intrinsic data information. The use of more hidden units allows the model to learn more complex patterns in the data. However, we must exercise caution because a large number of hidden units can result in overfitting. As a result, there is always a need to have a trade-off between model complexity and overfitting on training data. To determine the optimal number of hidden units, we would need to fine-tune the model to determine which number of hidden units performs the best.

Given the table, the ReLU activation function outperforms tanh. This is not a surprise since it has been showed many times that the ReLU activation function often performs well in practice.

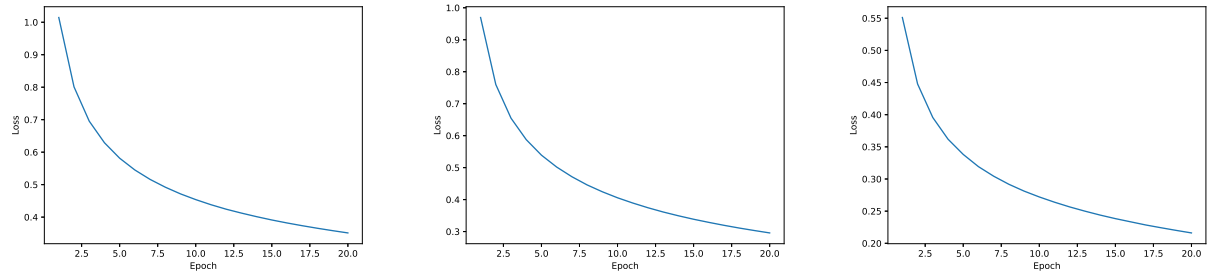
Increasing the dropout reduces the model's performance. This is due to the fact that increasing the dropout number causes the model to randomly drop out more neurons during training. Given that the default model only had 100 hidden units, removing 50% of the neurons results in only 50 hidden units remaining. This simplifies our model, reducing the model's ability to learn more complex details about the data.



(a) (b) (c)

Figure 6: Evolution of the accuracy for the validation set for three different cases: (a) all hyperparameters set to default, (b) hidden size = 200, (c) learning rate = 0.1. In each case the remaining hyperparameters were set as default.

Figure 6 (c) can be used to confirm the statement made in the previous question. When we compare (c) with (a) and (b), it is clear that the higher learning rate results in more unstable model.



(a) (b) (c)

Figure 7: Evolution of the training loss for three different cases: (a) all hyperparameters set to default, (b) hidden size = 200, (c) learning rate = 0.1. In each case the remaining hyperparameters were set as default.

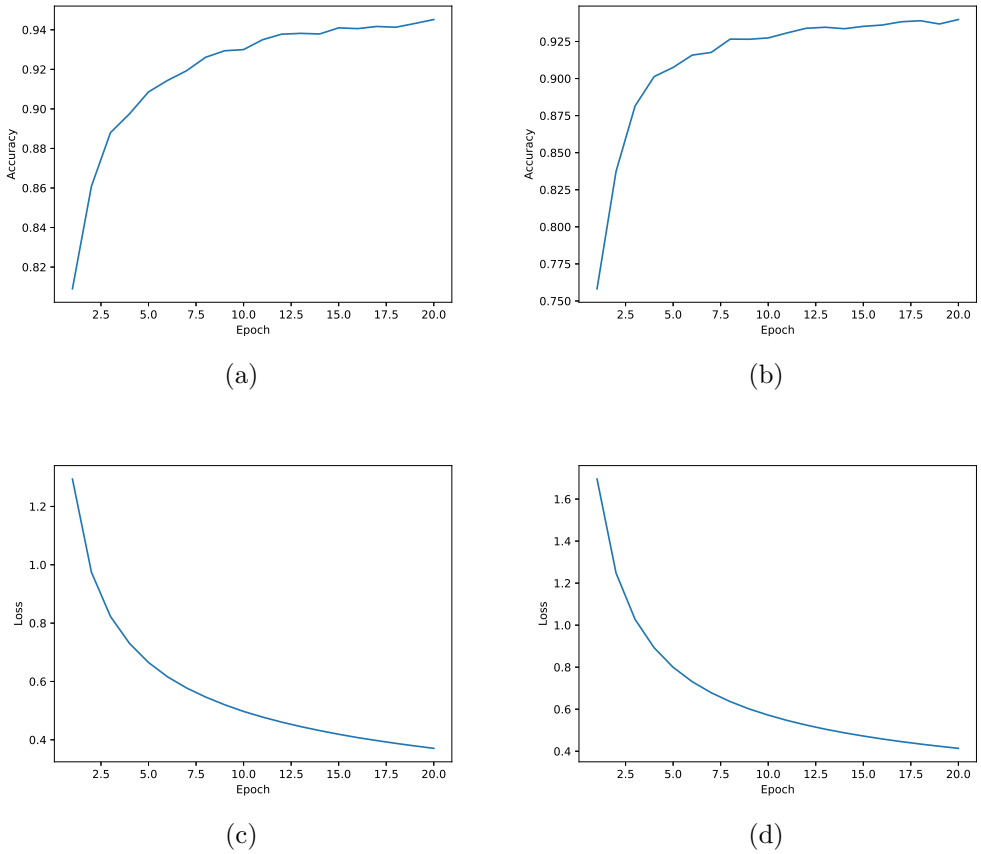
3.

Now we have to see the performance of the default model of the previous exercise for different number of hidden layers. Table 3 shows the validation and test set accuracy for three different situations. Figure 8 show the evolution of accuracy and training loss for the three best models, i.e., those that obtain the best accuracy for the test set.

Nº of hidden layers	Validation set acc.	Test set acc.
1	0.9387	0.8599
2	**0.9452**	**0.8723**
3	0.9398	0.8633

Table 3: Validation and test sets accuracies for the last epoch, for three different values of the learning rate: 0.001, 0.01 and 0.1.

As shown in table 3, the model with two hidden layers produced the best results. For this dataset, one hidden layer with 100 hidden units is insufficient to learn all of the data's complex details. With two hidden layers, we get a more complex model that can learn more complex patterns within the data and produce better results. However, simply increasing the number of hidden layers will not produce better results. The MLP with three hidden layers performs worse than the one with two. One of the main reasons for this could be that as the number of hidden layers increases, so does the likelihood of overfitting the training data. As a result, we must exercise extreme caution when increasing the number of hidden layers, as doing so increases the likelihood of our model overfitting the training data and failing to generalize the test data as well as would be expected.



(a) (b)

(c) (d)

Figure 8: Plots for four different scenarios: (a) accuracy evolution for two hidden layers, (b) accuracy evolution for three hidden layers, (c) training loss for two hidden layers, (d) training loss for three different layers. In each case the remaining hyperparameters were set as default.

Question 3

1.

In this problem we have a particular kind of activation function. Given that we have:

$$h = g(Wx) = (Wx)^2 \in \mathbb{R}^{K\times 1}, \tag{1}$$

we can first approach the problem for the k^{th} entry of h, i.e.,

$$\text{for } h = \begin{bmatrix} -h_1- \\ -h_2- \\ \vdots \\ -h_k- \end{bmatrix}, \text{consider the } k^{th} \text{ row: } h_k = (w_k x)^2, \text{for } w_k \in \mathbb{R}^{1\times D}. \tag{2}$$

Given that $w_k x$ is a real number, we have:

$$\begin{aligned} (w_k x)^2 &= (w_k x)^\top (w_k x) \\ &= x^\top w_k^\top w_k x \\ &= x^\top (w_k^\top w_k) x \ ^{(i)}, \end{aligned} \tag{3}$$

where:

$$(w_k^\top w_k) = \begin{bmatrix} W_{k1}^2 & W_{k1}W_{k2} & \dots & W_{k1}W_{kD} \\ W_{k2}W_{k1} & W_{k2}^2 & \dots & W_{k2}W_{kD} \\ \vdots & \vdots & \ddots & \vdots \\ W_{kD}W_{k1} & W_{kD}W_{k2} & \dots & W_{kD}^2 \end{bmatrix}, \text{and } x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix}. \tag{4}$$

Expanding into sum form:

$$x^\top (w_k^\top w_k) x = \sum_{j=1}^{D} x_j \left(\sum_{i=1}^{D} W_{kj} W_{ki} x_i \right) = \sum_{j=1}^{D} \sum_{i=1}^{D} W_{kj} W_{ki} x_j x_i. \tag{5}$$

Note that we would achieve the same result if we used the trace concept. Given that $w_k x$ is a real number, we have that:

$$\begin{aligned} (w_k x)^2 &= tr((w_k x)^2) \\ &= tr((w_k x)^T (w_k x)) \\ &= tr(x^T (w_k^T w_k) x) \\ &= tr(x x^T (w_k^T w_k)). \end{aligned} \tag{6}$$

Using some algebric manipulation we can expand the sum in (5) into the following form:

$$\sum_{j=1}^{D} W_{kj}^2 x_j^2 + 2 \sum_{j=1}^{D-1} \sum_{i=j+1}^{D} W_{kj} W_{ki} x_j x_i. \tag{7}$$

[(i)] Using the associative propriety of matrix multiplication.

Now we have all the thing we need to deduce the mapping $\phi(x)$:

$$\phi(x) = \begin{bmatrix} x_1^2 & x_2^2 & \dots & x_D^2 & 2x_1x_2 & \dots & 2x_1x_D & 2x_2x_3 & \dots & 2x_2x_D & \dots & 2x_{D-1}x_D \end{bmatrix}^\top. \tag{8}$$

This transformation converts the input vector x to a vector of length $\frac{D(D+1)}{2}$. Now we can take all the K entries of h and define A_Θ in such a way that $h = A_\Theta\phi(x)$:

$$A_\Theta = \begin{bmatrix} W_{11}^2 & \dots & W_{1D}^2 & W_{11}W_{12} & \dots & W_{11}W_{1D} & W_{12}W_{13} & \dots & W_{1D-1}W_{1D} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ W_{K1}^2 & \dots & W_{KD}^2 & W_{K1}W_{K2} & \dots & W_{K1}W_{KD} & W_{K2}W_{K3} & \dots & W_{KD-1}W_{KD} \end{bmatrix}, \tag{9}$$

Where $A_\Theta \in \mathbb{R}^{K\times\frac{D(D+1)}{2}}$. This demonstrates that h can be expressed as a linear function of the mapping $\phi(x)$.

2.

Since $h = A_\Theta\phi(x)$, we have:

$$\hat{y} = v^\top h = v^\top A_\Theta\phi(x) = (A_\Theta^\top v)^\top\phi(x) \in \mathbb{R}. \tag{10}$$

Therefore, we can define $c_\Theta = A_\Theta^\top v$. So for the parameters c_Θ, $\hat{y}$ is a linear transformation of $\phi(x)$. However, as we can see in (13), $\phi(x)$ is not a linear mapping of x. Thus, the predicted output is not a linear transformation of the input x. In a similar vein, A_Θ is not linear in relation to the weights, e.g., the first element of the matrix A_Θ is $A_{\Theta 11} = W_{11}^2$. Consequently, c_Θ will not be linear in relation to the parameters Θ, which means that the model is not linear in terms of the parameters Θ.

3.

Unfortunately the group did not arrive to any conclusion in this exercise. Despite that, the group still dedicated a lot of time to this question and we think that we are close, but there is still something missing. Since this was the question that consumed time, the group decided to still present the work done.

First, the group tried to solve this question by using the notion of under-determined, well-determined and over-determined systems. However this is not applicable in non-linear systems, which is the case, given that c_Θ as a non-linear relationship with the Θ parameters. Despite this, the group still thinks that this notions might play an important role in this exercise, so we will try to explain the group's line of thought.

To prove that for any given c_Θ that is corresponding Θ in the original model so the two models are equivalent, we need to prove that the system $c_\Theta = A_\Theta v$ has a solution. In order to prove that, we started to look at the number of variables and equations of the system.

For $K \geq D$, then $K \times D + K > \frac{D(D+1)}{2}$ this means that we have more unknowns that equations. If this was a linear system, this would mean that we had an infinite amount of solutions. However this is not applicable in non-linear systems. For example, $x^2 + y^2 = -1$ is an under-determined system and is also impossible.

So the group decided to take another approach. The group decided to expand the system to see if we could find any relevant information about K and D. In order to do that the group decided to consider $K = 3$ and $D = 2$:

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \Rightarrow (w_k^\top w_k) = \begin{bmatrix} w_{k1}^2 & w_{k1}w_{k2} \\ w_{k2}w_{k1} & w_{k2}^2 \end{bmatrix}, \text{ where } w_k \in \mathrm{R}^{1\times D} \tag{11}$$

In order to simplified the computations, we will make the following alteration to the matrix A_Θ presented in (9):

$$A_\Theta = \begin{bmatrix} w_{11}^2 & w_{12}^2 & \sqrt{2}w_{11}w_{12} \\ w_{21}^2 & w_{22}^2 & \sqrt{2}w_{21}w_{22} \\ w_{31}^2 & w_{32}^2 & \sqrt{2}w_{31}w_{32} \end{bmatrix}, \tag{12}$$

note that this modification is still valid as long as we consider:

$$\phi(x) = \begin{bmatrix} x_1^2 & x_2^2 & \dots & x_D^2 & \sqrt{2}x_1x_2 & \dots & \sqrt{2}x_1x_D & \sqrt{2}x_2x_3 & \dots & \sqrt{2}x_2x_D & \dots & \sqrt{2}x_{D-1}x_D \end{bmatrix}^\top, \tag{13}$$

which is also valid. Considering the concept of Frobenius norm we have:

$$\|(w_k^T w_k)\|_F = \sqrt{w_{k1}^4 + 2w_{k1}w_{k2} + w_{k2}^4} = \sqrt{(w_{k1}^2 + w_{k2}^2)^2} = w_{k1}^2 + w_{k2}^2, \tag{14}$$

equaling $\|A_{\Theta k}\|_2$ and $\|(w_k^T w_k)\|_F$, we obtain:

$$\|A_{\Theta_k}\|_2 = \|(w_k^T w_k)\|_F \Leftrightarrow \sqrt{w_{k1}^4 + 2w_{k1}w_{k2} + w_{k2}^4} = w_{k1}^2 + w_{k2}^2 \Leftrightarrow (w_k^T w_k) \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = tr(w_k^T w_k),$$

where the vector $[110]^T \in \mathbf{R}^{\frac{D(D+1)}{2}}$ has D entries equal to 1, and $\frac{D(D-1)}{2}$ entries equal to 0. Considering now all the K rows, we have:

$$\begin{aligned} A_\Theta \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} &= \begin{bmatrix} tr(A_1) \\ tr(A_2) \\ tr(A_3) \end{bmatrix} \\ \Leftrightarrow A_\Theta \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} &= \begin{bmatrix} w_1 w_1^\top \\ w_2 w_2^\top \\ w_3 w_3^\top \end{bmatrix} \\ \Leftrightarrow A_\Theta \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} &= \begin{bmatrix} w_1 & 0^\top & 0^\top \\ 0^\top & w_2 & 0^\top \\ 0^\top & 0^\top & w_3 \end{bmatrix} \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}^\top \\ \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} A_\Theta^\top v &= \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} w_1^\top & 0 & 0 \\ 0 & w_2^\top & 0 \\ 0 & 0 & w_3^\top \end{bmatrix} v. \end{aligned}$$

As we can see we obtained an equation that relates $c_\Theta = A_\Theta v$ with the parameters Θ. However, this equation does not seem to depend the inequality $K \geq D$. We think that we were on the right track, but there was something missing in order to relate the developments made by the group with the inequality $K \geq D$. The group thinks that it still might be related with the rank of one (or more) those matrices. In other words, although the concept of under-determined and over-determined systems is not applicable in non-linear systems, we still need to keep in mind the degrees of freedom, the number of equations and most importantly what could the rank of matrix A_Θ tells us about $c_\Theta = A_\Theta v$ having (or not) a solution. Unfortunately, we could not conclude anything.

4.

As it was shown in Q3-2, for c_Θ, $\hat{y}$ is a linear transformation of $\phi(x)$. So as we are showing in this exercise, with the right reparametrization the quadratic activation function can be tackled as a linear model. Thus, this output layer can be considered a linear regression problem.

Since X is full column rank, i.e., all the columns of the matrix X are linearly independent, which means the this matrix can be inverted. Thus, we can define its pseudo-inverse $((X^T X)^{-1} X^T)$.

Taking the points mentioned previously into account, and taking the fact that we have a quadratic loss function, then we can obtain the closed-form solution from the normal equation. We obtain the following closed-form solution:

$$\hat{c}_\Theta = (X^T X)^{-1} X^T y. \tag{15}$$

Due to the non-linear activation functions typically used for feedforward neural networks (e.g., ReLU, tanh, etc.), the loss function of this networks is usually very complex a will probably have many candidates for global minimum, which makes global minimization often intractable.

In this particular case, we showed that with the use of a quadratic activation function and the proper reparameterization, this problem can be considered a linear least squares problems (given that X is full column rank). For this case we have a closed-form solution (15) to find the global minimum.

References

[1] Activation Functions in Neural Networks

[2] Comparison of Sigmoid, Tanh and ReLU Activation Functions

[3] Global Optimization for Low-Dimensional Switching Linear Regression

[4] Lecture Slides