Deep Learning - Homework 1

Group 10
95572 Filipa Cotrim
95618 Leonor Barreiros

We started by dividing the coding part of this homework by the two of us. The questions 1.2 and 2.1 were done by Leonor, while the questions 1.1 and 2.2 were done by Filipa. Although we distributed between the both of us, we double-checked every implementation done by the other and worked together through some issues, specially in question 2.2. The theoretical questions were discussed between the two of us, and we both contributed to the making of the report. All the code was written by us, however, we took inspiration from the labs code solutions provided by the professors, some snippets of code written on the board during practical lessons, and some algebra results written on the board in theoretical and practical lessons.

Question 1

Question 1.1

1.1. a)

We measured performance with accuracy. The accuracy on the validation set was **72.36%** and on the test set it was **61.93%**.

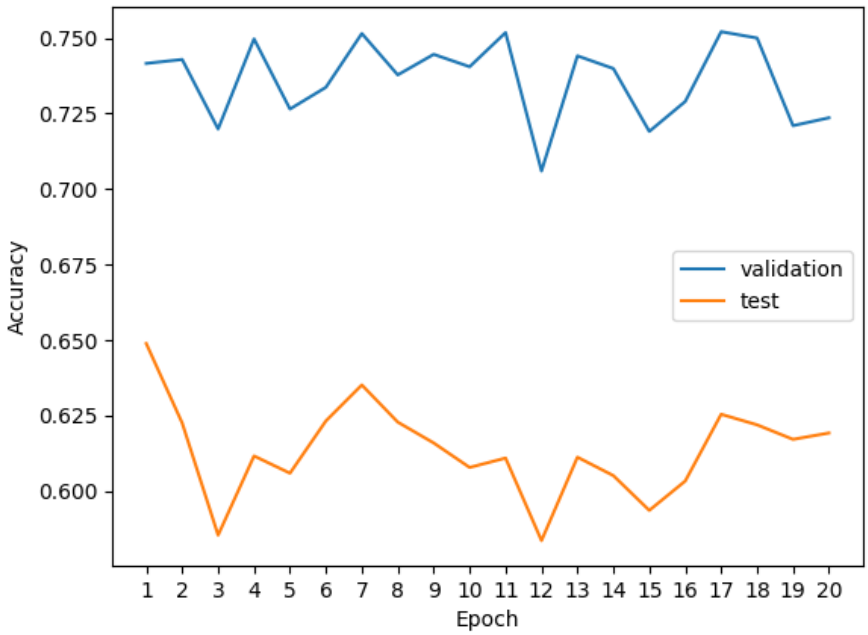

Figure 1: **Perceptron:** Validation and test accuracies as a function of the epoch number.

1.1. b)

We measured performance with accuracy. The accuracy on the validation set was **82.51%** and on the test set it was **70.28%**.

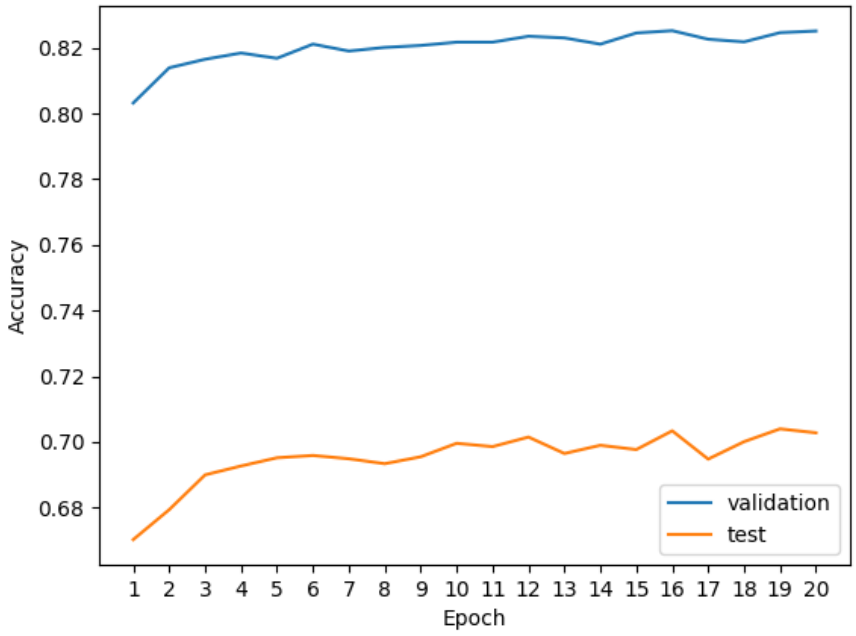

Figure 2: **Logistic Regression:** Validation and test accuracies as a function of the epoch number.

Question 1.2

1.2. a)

The perceptron is a type of **linear classifier**. For example, in the one we implemented before, we determined the equation of a hyperplane that separates between 10 classes (the 10 characters): the linear combination of W and x will allow the model to decide which is the correct label for the input x.

The limitation of this model, even when we combine several perceptrons (and construct a multi-layer perceptron) is that its expressiveness is limited to **linearly separable problems**. So, for non-linearly separable datasets it will never converge, since non-linearity means there's no line such that a class is represented on one side and the other on the other.

If we see a non-linear problem as a disjunction of conjunctions, we can represent it as a combination of perceptrons, a multi-layer perceptron. Additionally, it has a combination of linear functions, that results in a linear function as well and continuing to not being able to learn non-linearly separable problems.

So, if there isn't such a thing as a separating hyperplane (like in the XOR problem), then we need to transform the data (bend the straight lines) in a way that we can compute one: enter **activation functions**. These type of MLPs, by composing linear and non-linear transformations, classify many problems. In fact, only **by alternating linear and non-linear transformations are we able to represent any decision boundary**.

Summing up, MLPs with non-linear activation functions are more expressive than the perceptron because they can learn non-linearly separable datasets and this can only happen when we introduce non-linearity.

1.2. b)

We measured performance with accuracy. The accuracy on the validation set was **93.07%** and on the test set it was **85.12%**.

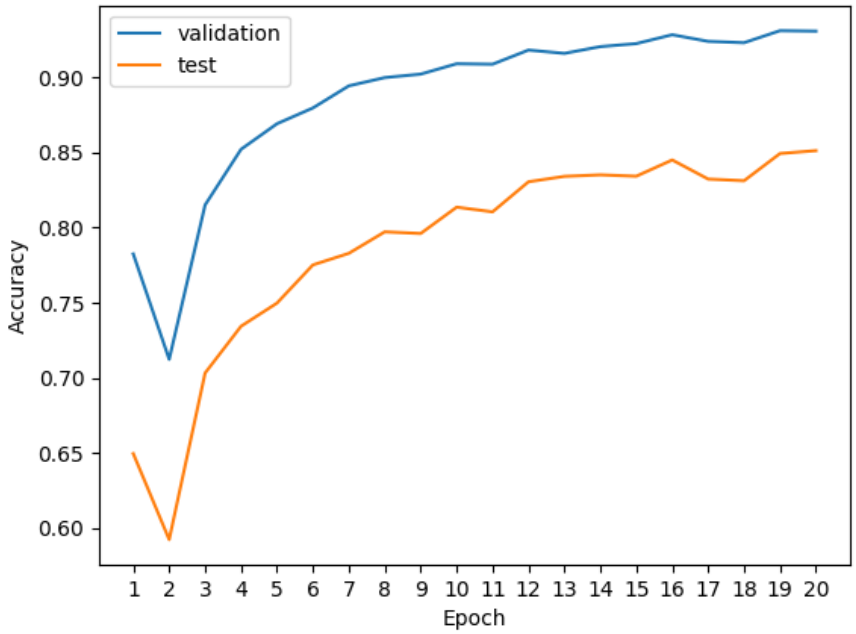

Figure 3: **Multilayer Perceptron:** Validation and test accuracies as a function of the epoch number.

Question 2

2.1

The best configuration (that is, the one that provided the highest validation accuracy) is with learning rate = **0.001**, which resulted in a final test accuracy of **70.19%**.

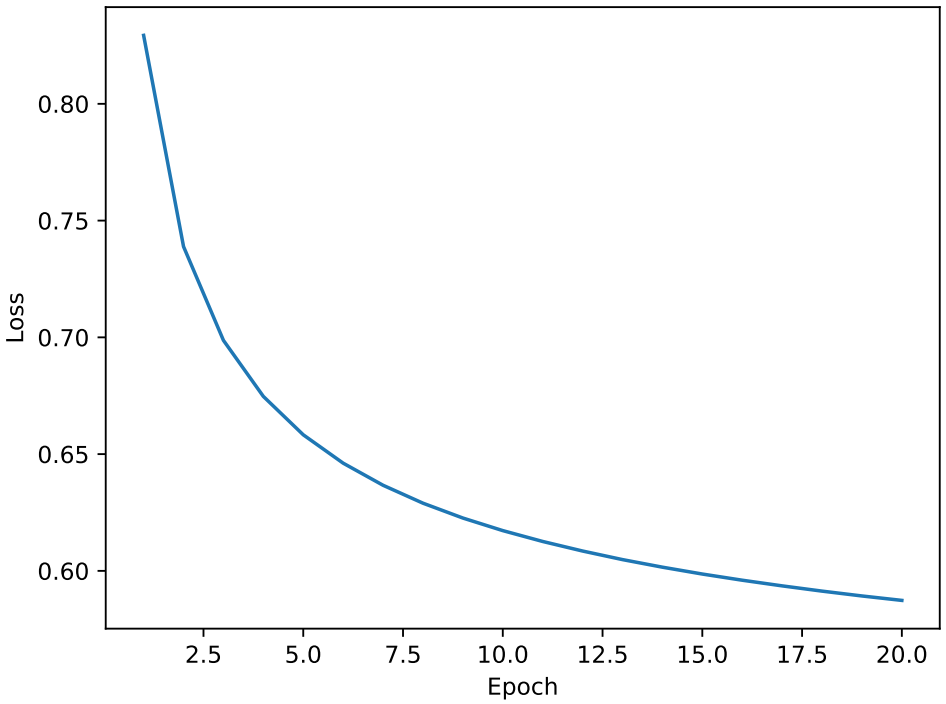

Figure 4: **PyTorch Logistic Regression:** Training loss as a function of the epoch number.

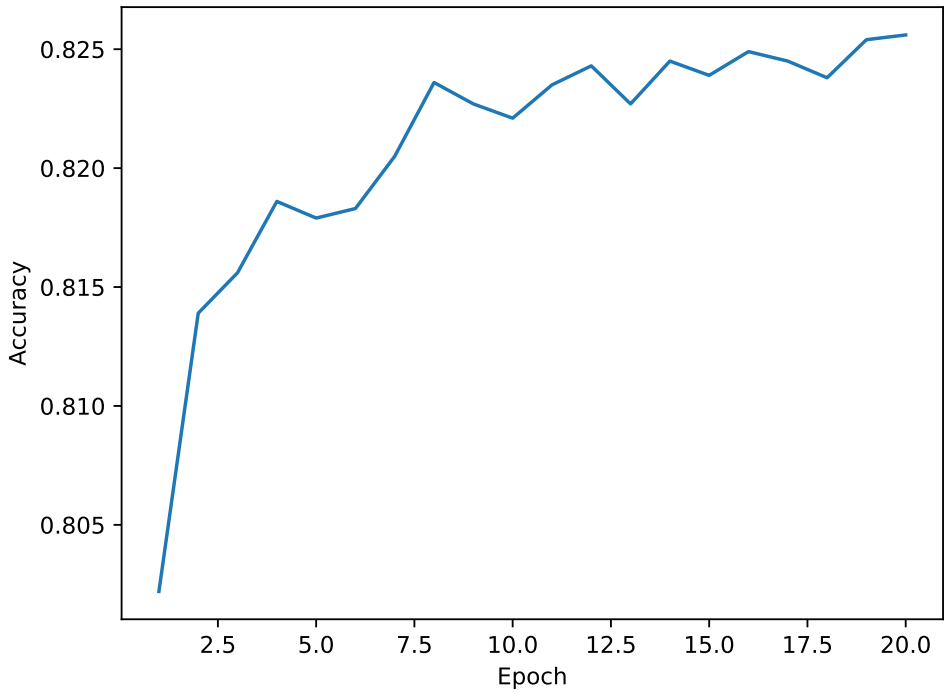

Figure 5: **PyTorch Logistic Regression:** Validation accuracy as a function of the epoch number.

2.2

The best configuration (that is, the one that provided the highest validation accuracy) is with hidden size = **200** (and the other parameters default), which resulted in a final test accuracy of **88.19%**.

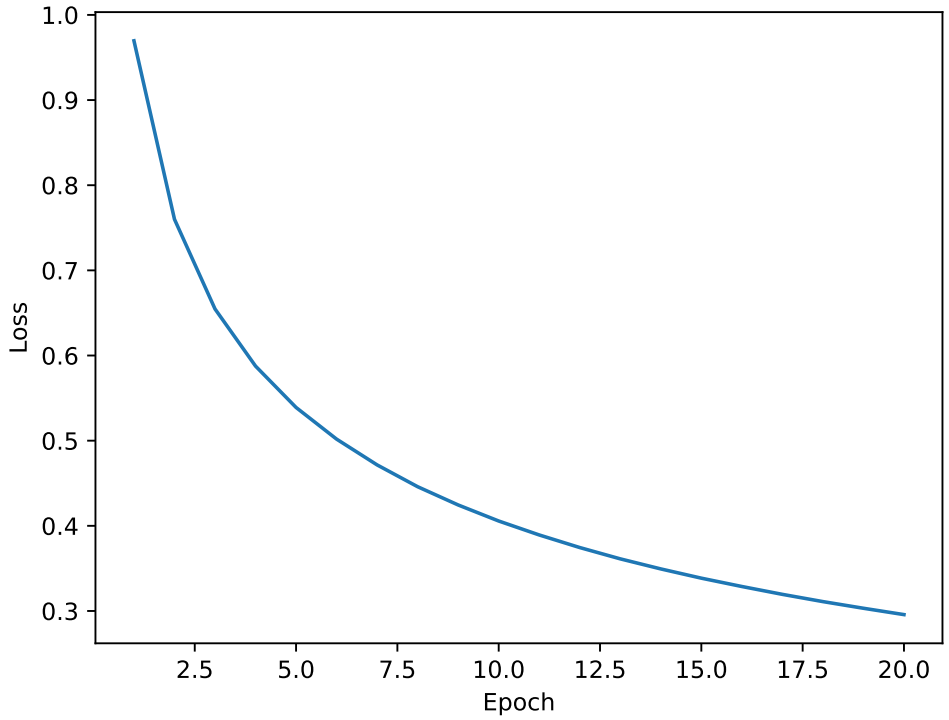

Figure 6: **PyTorch MLP:** Training loss as a function of the epoch number.

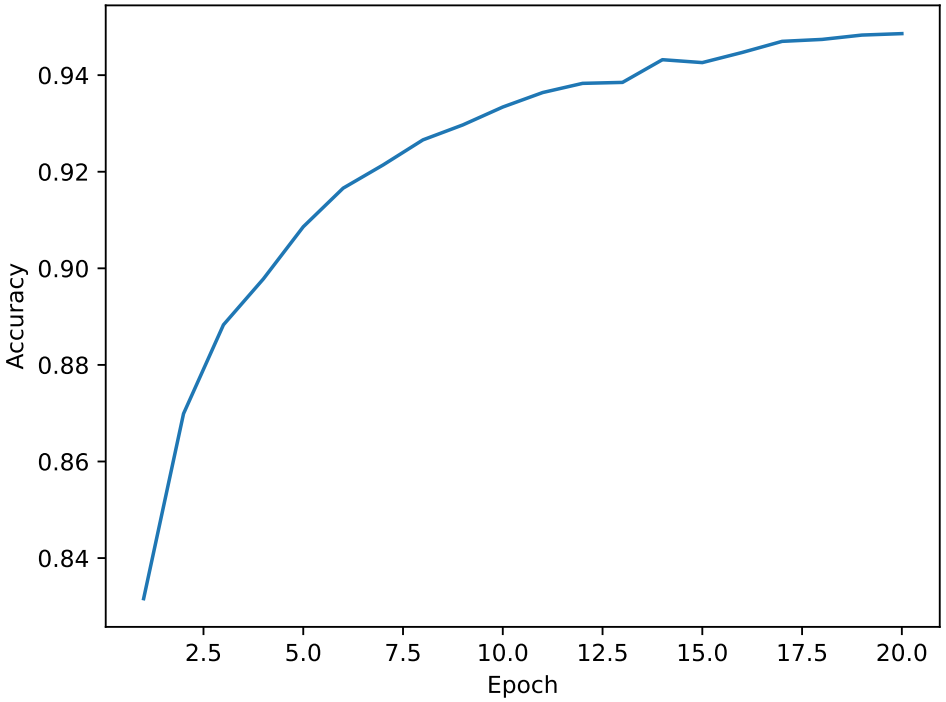

Figure 7: **PyTorch MLP:** Validation accuracy as a function of the epoch number.

2.3

The best configuration (that is, the one that provided the highest validation accuracy) is with hidden layers = **2** (and the other parameters default), which resulted in a final test accuracy of **87.25%**.

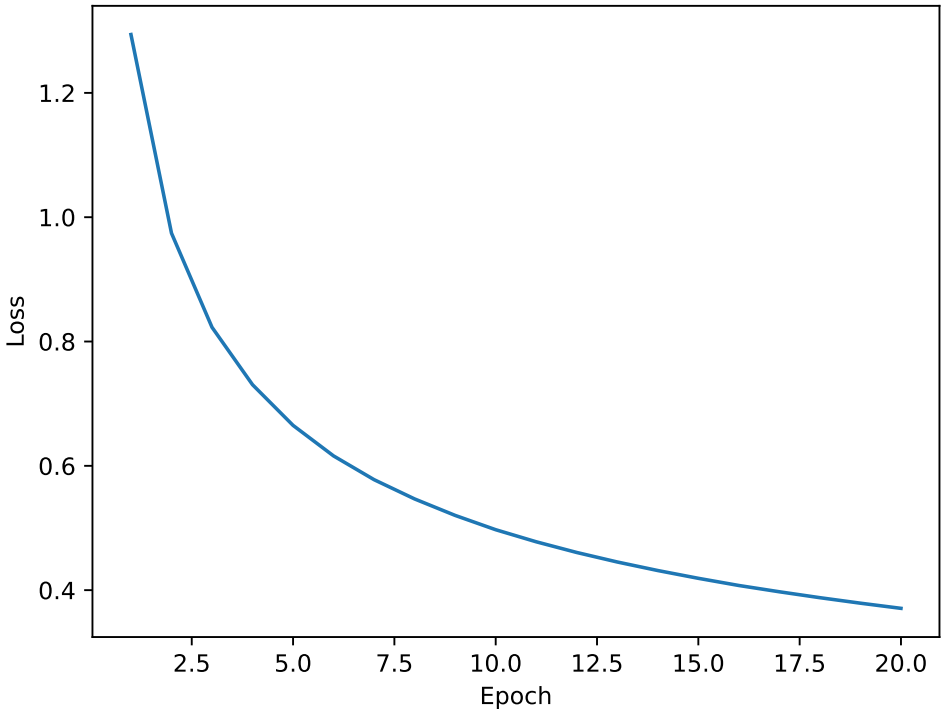

Figure 8: **PyTorch MLP:** Training loss as a function of the epoch number.

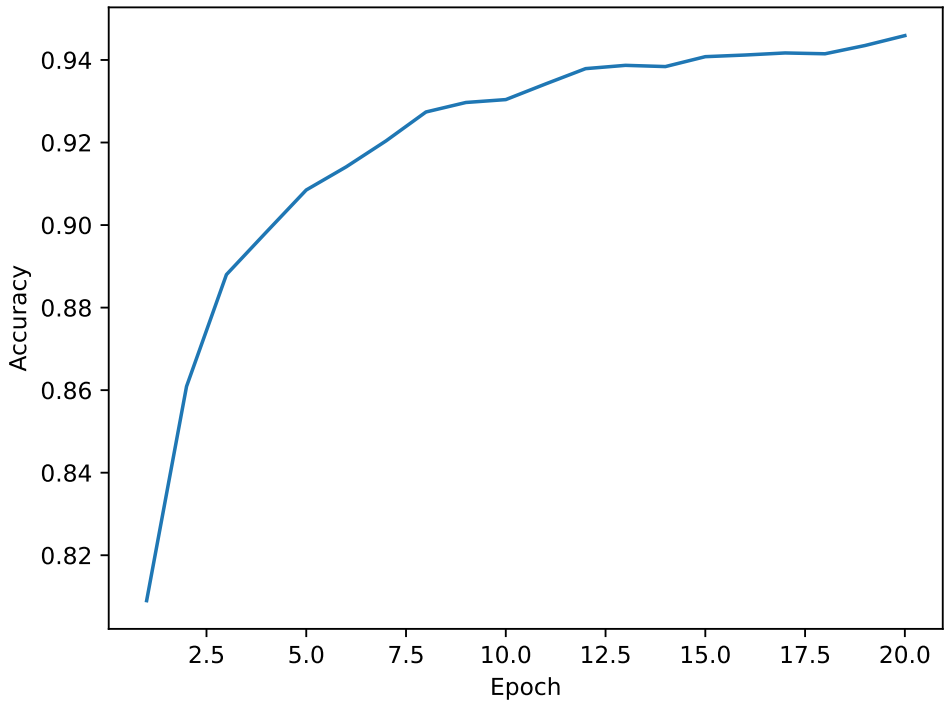

Figure 9: **PyTorch MLP:** Validation accuracy as a function of the epoch number.

Question 3

3.1

We want to prove **h is a linear transformation of** $\phi(x)$, by defining the mapping $\phi(x)$ and the matrix A_Θ. We have $h = g(Wx) = (Wx)^2$, and we want to write $h = A_\Theta \phi(x)$ such that:

- $\phi(x)$ is a transformation of x such that $\phi : R^D \to R^{\frac{D(D+1)}{2}}$ (independent of Θ)
- $A_\Theta \in R^{k \times \frac{D(D+1)}{2}}$

We begin by expanding the formulation of h, in a way that we'll be able to later extract $\phi(x)$ and A_Θ from it.

We have:

$$W_{k\times D} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1D} \\ w_{21} & w_{22} & \cdots & w_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k1} & w_{k2} & \cdots & w_{kD} \end{bmatrix} \quad x_{D\times 1} = \begin{bmatrix} x_{11} \\ \vdots \\ x_{1D} \end{bmatrix}$$

$h = g(W_{K\times D} x_{D\times 1}) = (W_{K\times D} x_{D\times 1})^2$. We can write each line of h as:

$$(w_k^T x)^2 = (\sum_{j=1}^{D} w_{kj} x_j)^2 = \sum_{j=1}^{D} (w_{kj} x_j)^2 + 2 \sum_{j=1}^{D} \sum_{i=1}^{j-1} (w_{kj} x_j)(w_{ki} x_i) = \sum_{j=1}^{D} (w_{kj} x_j)^2 + 2 \sum_{j=1}^{D} \sum_{i=1}^{j-1} (w_{kj} w_{ki})(x_j x_i) \tag{1}$$

Before we extract $\phi(x)$ and A_Θ, we notice the number of elements being summed in each line of h.

We have $D + \sum_{j=1}^{D} \sum_{i=1}^{j-1} (1) = D + \frac{(D-1)D}{2} = \frac{(D+1)D}{2}$, which is the exact number of elements in $\phi(x)$, and the number of columns in A_Θ.

Now, we are ready to determine the values in each column of A_Θ, and in each line of $\phi(x)$. We want to write $h = A_\Theta \phi(x)$, which is equivalent to writing $A_\Theta \phi(x) = (W_{K\times D} x_{D\times 1})^2$.

We know the value of each line of h. That corresponds to the dot product of each $A_{\Theta k}$ (each column of A_Θ) and $\phi(x)$. So, since each line of h can also be written as $A_{\Theta k} \phi(x)$, we can extract each element

summed in each line of h as each element being multiplied in that dot product - each element in each matrix/vector.

The goal is that $A_{\Theta k}\phi(x) = (w_k^T x)^2$, so we need to choose the values in a way to preserve each pair of multiplied values (and in a way that $\phi(x)$ is independent of Θ). As long as we preserve that, we can change the order of the entries in the vectors (as long as all pairs will still multiply with each other). That means we're showing one of many possible solutions.

$$a_k = \begin{bmatrix} w_{k1}^2 \\ \vdots \\ w_{kD}^2 \\ 2w_{k1}w_{k2} \\ \vdots \\ 2w_{k(D-1)}w_{kD} \end{bmatrix} \quad \phi(x) = \begin{bmatrix} x_1^2 \\ \vdots \\ x_D^2 \\ x_1x_2 \\ \vdots \\ x_{(D-1)}x_D \end{bmatrix}$$

3.2

In the previous exercise, we showed that h is a linear transformation of $\phi(x)$ as we can write $h = A_\Theta\phi(x)$.

Similarly, $\hat{y}$ is a linear transformation of $\phi(x)$ if we can write $\hat{y} = c_\Theta^T\phi(x)$ for some $c_\Theta \in R^{\frac{D(D+1)}{2}}$.

We have $\hat{y} = v^T h = v^T A_\Theta\phi(x)$. So, if we put $c_\Theta = (v^T A_\Theta)^T = A_\Theta^T v$, then we have $\hat{y}(x; c_\Theta) = c_\Theta^T\phi(x)$.

Given that, in relation to Θ, the entrances of c_Θ are not a linear transformation (since the new parameters are quadratic in relation to the original ones), ours is not a linear transformation in terms of Θ.

3.3

Our goal now is to show that for any $c \in R^{\frac{D(D+1)}{2}}$ there is a choice of the original parameters $\Theta = (W, v)$ such that $c_\Theta = c$. In other words, that it's **equivalent to parameterize the model with c_Θ instead of Θ.**

We begin by analyzing the value of $\hat{y}$. We have:

$$\hat{y} = v^T h = \sum_{k=1}^{K}(v_k h_k) = \sum_{k=1}^{K}(v_k(w_k^T x)^2) = \sum_{k=1}^{K}(v_k(w_k^T x)(x^T w_k)) = \sum_{k=1}^{K}((w_k^T v_k w_k)(xx^T)) \tag{2}$$

[1]

We can write that result in matrix form, and have $< W^T VW, xx^T >_F$, where V denotes a diagonal matrix whose non-zero entries are the values in v.

Note the matrix $W^T VW$ is symmetric: the product of a matrix and its transpose is a symmetric matrix, and the product of a diagonal matrix and a symmetric matrix is a symmetric matrix. By seeing an outline of it, we have an idea of it's structure, and are able to see the symmetry.

$$\begin{aligned}
&\mathrm{W}^T VW \\
&= \begin{bmatrix} w_{11} & \cdots & w_{K1} \\ \vdots & \ddots & \vdots \\ w_{1D} & \cdots & w_{KD} \end{bmatrix} \begin{bmatrix} v_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & v_K \end{bmatrix} \begin{bmatrix} w_{11} & \cdots & w_{1D} \\ \vdots & \ddots & \vdots \\ w_{K1} & \cdots & w_{KD} \end{bmatrix} \\
&= \begin{bmatrix} w_{11}v_1 & \cdots & w_{K1}v_K \\ \vdots & \ddots & \vdots \\ w_{1D}v_1 & \cdots & w_{KD}v_K \end{bmatrix} \begin{bmatrix} w_{11} & \cdots & w_{1D} \\ \vdots & \ddots & \vdots \\ w_{K1} & \cdots & w_{KD} \end{bmatrix} \\
&= \begin{bmatrix} w_{11}^2v_1 + \cdots + w_{K1}^2v_K & \cdots & w_{11}w_{1D}v_1 + \cdots + w_{K1}w_{KD}v_K \\ \vdots & \ddots & \vdots \\ w_{11}w_{1D}v_1 + \cdots + w_{K1}w_{KD}v_K & \cdots & w_{1D}^2 + \cdots + w_{KD}^2v_K \end{bmatrix}
\end{aligned} \tag{3}$$

[1] $Tr(ABC) = Tr(BCA)$. Since $(w^T x)^2$ is a scalar, we can do $(w^T x)^2 = Tr((w^T x)(x^T w) = Tr((w^T w)(xx^T))$. Furthermore, since v_k is a scalar, we can put it anywhere in the multiplication.

Now that we've defined $\hat{y}$ as the Frobenius product of two matrices, to show that parameterizing with c_Θ is equivalent as with Θ, we need to be able to extract c_Θ in a way that we don't lose expressive power (i.e., we don't lose any parameters).

The first thing we need to note is that W^TVW is a matrix, while c_Θ is a vector, so c_Θ will have to be a vectorization of W^TVW. Furthermore, W^TVW has $\sum_{i=1}^{D}(i) = \frac{D(D+1)}{2}$ different parameters (which correspond to the entries in the main diagonal and above/below): exactly the size of c_Θ. That means, even if W^TVW has full-column rank (i.e., is an orthogonal basis), we have space in c_Θ to put its information.

So, if we take those parameters and insert each of them into an entry of c_Θ, we are able to parameterize our model equivalently. Since we wrote c_Θ as a function of Θ, and since $\hat{y}$ is a linear transformation of $\phi(x)$, **our model is linear in terms of** c_Θ. Therefore, we just showed that, via the reparametrization c_Θ, we tackle our problem as a linear model.

If $K < D$, then W^TVW may have more parameters than c_Θ can hold. In this case, we have two situations:

- Some of W^TVW's columns are a linear combination of others, and so $rank(W^TVW) < K$. In that case, by extracting the orthogonal basis (of size $\leq K$) we can still construct c_Θ;
- The columns of W^TVW are linearly independent, and it's an orthogonal basis, or some of them are a linear combination of others but the rank of the matrix is still larger than the size of c_Θ. In that case, we have $rank(W^TVW) = D > K$ or $rank(W^TVW) = D - \alpha > K$ (respectively), and we don't have enough space in c_Θ to fit all the necessary parameters. So, **an equivalent parametrization might not exist if** $K < D$.

3.4

The squared loss is as follows:

$L(c_\Theta; D) = \frac{1}{2}\sum_{n=1}^{N}(\hat{y}(x_n; c_\Theta) - y_n)^2) = \frac{1}{2}\sum_{n=1}^{N}(y_n - c_\Theta^T X)^2 = \frac{1}{2}\|y_n - Xc_\Theta\|^2$, where $X \in R^{N\times \frac{D(D+1)}{2}}$ has $\phi(x_n)$ as rows.

To find the best $\hat{y}$, we minimize the loss (we're approximating the target). In our case, we have a closed form solution and can compute it with:

$\Delta_{c_\Theta}\frac{1}{2}\|y - Xc_\Theta\|^2 = \Delta_{c_\Theta}\frac{1}{2}(c_\Theta^T X^T X c_\Theta - 2c_\Theta^T X^T y + \|y\|^2) = X^T X c_\Theta - X^T y$

$\Delta_{c_\theta} = 0 \Leftrightarrow X^T X c_\Theta - X^T y = 0 \Leftrightarrow X^T X c_\Theta = X^T y \Leftrightarrow c_\theta = (X^T X)^{-1} X^T y$

We could find a closed-form solution, and determine the global minimum, because the problem at hands is linear - and we showed ours could be tackled as a linear model via a reparametrization. In that case, we have a compact description of the function given by the parameters, and we can take its gradient to compute the minimizer.

However, global minimization is usually intractable for Feed-Forward Neural Networks, since they introduce non-linearity. In that case, we can only minimize the loss by using iterative algorithms such as gradient descent.

Concluding, what makes our problem special, i.e., what makes our feed-forward neural network have a closed form solution, is that it can be dealt with as if it were a linear model. So, like in linear regression, we can analytically compute the closed-form solution.