Técnico Lisboa

Deep learning

Homework #1

Karolina Kowalczyk
ist1105122
João Inês
ist73311

2022-12-23

Member's contribution

Question 1.1 a) - Karolina and João
Question 1.1 b) - João
Question 1.2 a) - João
Question 1.2 b) - João

Question 2.1 - Karolina
Question 2.2 - Karolina
Question 2.3 - Karolina

Question 3.1 - Karolina and João
Question 3.2 - we tried to solve together
Question 3.3 - we tried to solve together
Question 3.4 - we tried to solve together

Code source

Our code source is available on GitHub repository.

Question 1

Question 1.1

a)

On this first question, we were asked to build a simple update weights function method for the perceptron class representing a simple perceptron. After training the perceptron for 20 epochs we got the following results:

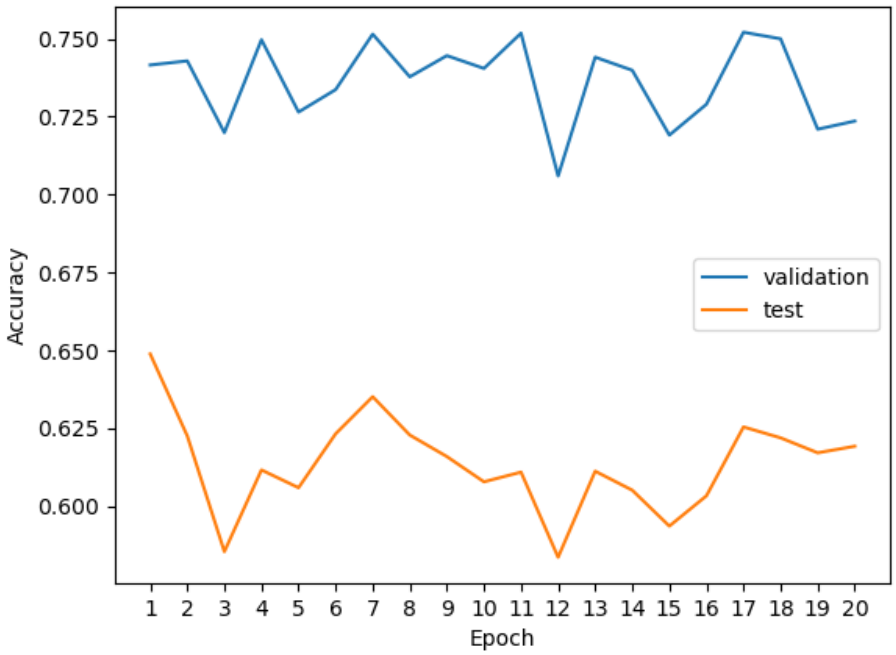


Figure 1: Validation and test accuracy for the Perceptron

b)

Following the same logic as before we implemented we used a logistic regression instead and used a stochastic gradient descent as our training algorithm. Our learning rate was fixed at $\mu = 0.001$

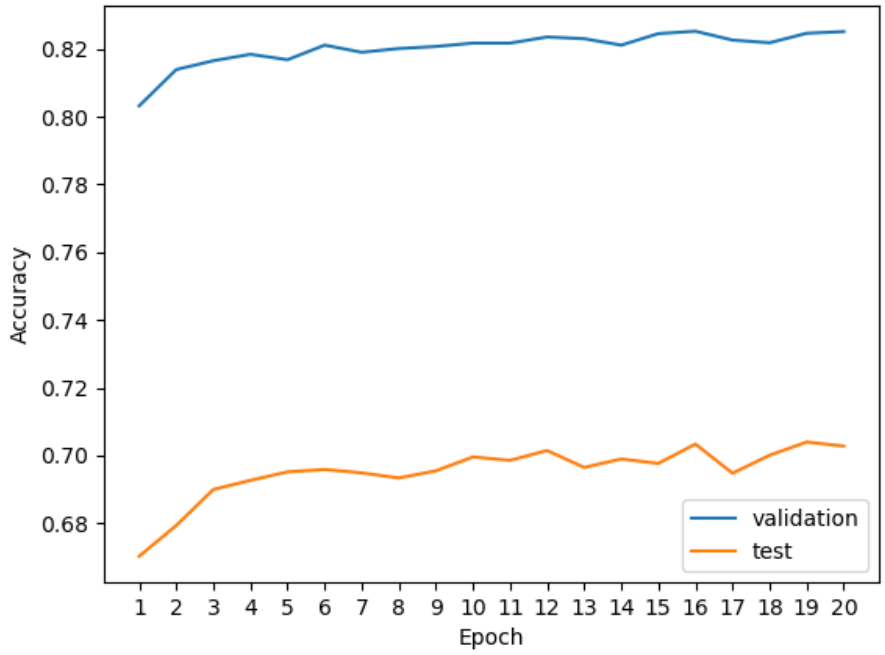


Figure 2: Validation and test accuracy for the logistic regression - learning rate=0.001

Question 1.2

a)

To answer shortly to "why multi-layer perceptrons with non-linear activations are more expressive than the simple perceptron" you can say that a single-layer perceptron can only learn and solve linear separable functions whereas a multilayer one can also learn non-linear functions. With the addition of extra layers, each new hidden layer will compute a representation of the input and propagates it forward. This increases the expressive power of the network, yielding more complex, non-linear, functions/classifiers, this can also be called a feed-forward neural network.
Summing up the advantage of multiple layers is that they can learn features at various levels of abstraction. For example, if we train a deep convolution neural network to classify images, we will find that the first layer will train itself to recognize very basic things like edges, the next layer will train itself to recognize collections of edges such as shapes, the next layer will train itself to recognize collections of shapes like eyes or noses, and the next layer will learn even higher-order features like faces. Multiple layers are much better at generalizing because they learn all the intermediate features between the input data and the high-level classification.
In the case of the activation function of the multi-layer perceptron being linear, we no longer have the power of solving these harder problems since our perceptron becomes simply a model of linear regression. The activation functions make this forward propagation possible by knowing gradients of the functions along with the error to update the weights and bias, because of this reason a

linear function would retorn us a constant value for the gradient something that doesn't happen on non-linear functions.

b)

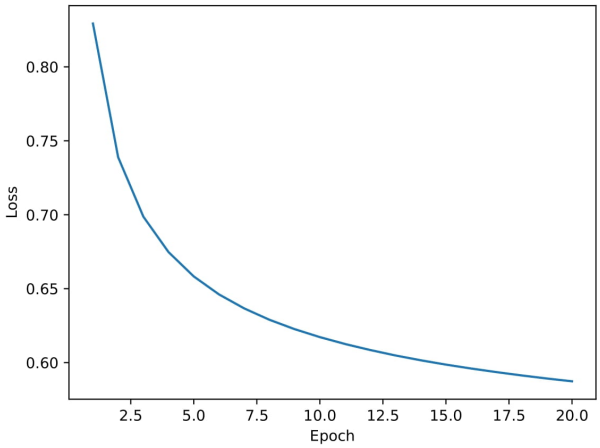
Question 2

Question 2.1

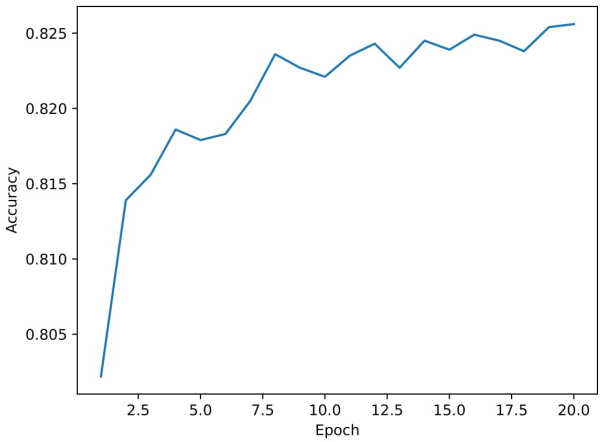
The first task of question 2 was about linear model with logistic regression. After implementing the given functions with the *pytorch* framework, we launched the program with default parameters. We only changed *learning rate* parameter (as specified in the task description). On the next page we present obtained result. The final accuracy were as follow:

learning rate	final accuracy
0.001	0.7019
0.01	0.6806
0.1	0.6098

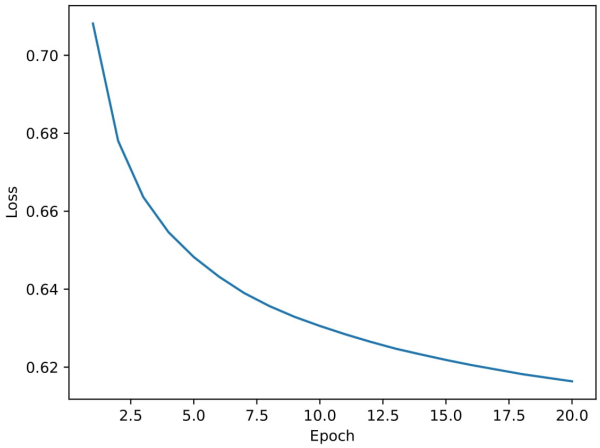
The best accuracy for 20 epoches was for 0.001 learning rate, what we can also see at plot. For higher learning rates higher accuracy result appeared in previous epoches. Trainig loss plot is more uniform for learning rate 0.001 and 0.01 than for 0.1. This leads to the conclusion that in this case learning rate = 0.1 is a little too high. The best results give us learning rate = 0.001, but learning rate = 0.01 seems to be good compromise, if we want to learn our model faster.



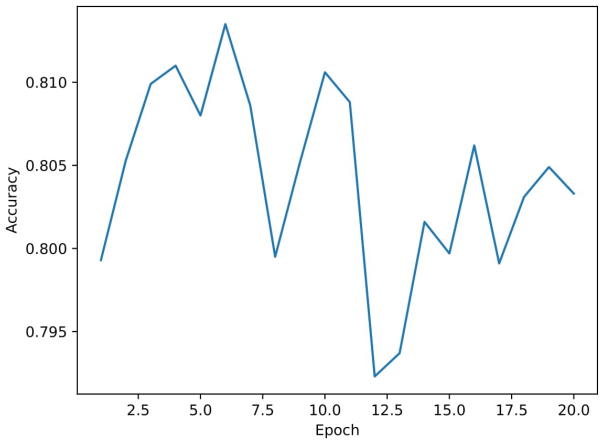
(a) Training loss - learning rate = 0.001



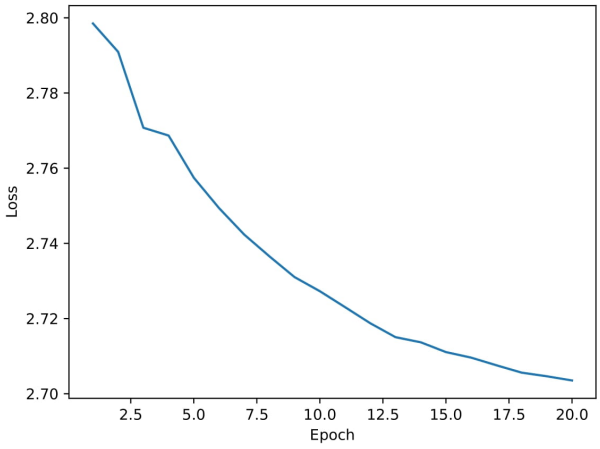
(b) Validation accuracy - learning rate = 0.001



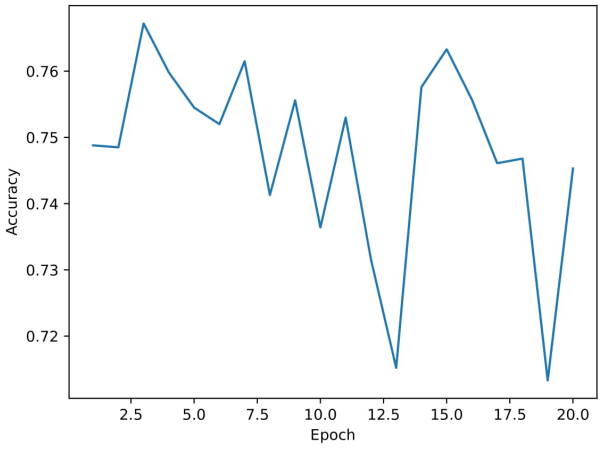
(c) Training loss - learning rate = 0.01



(d) Validation accuracy - learning rate = 0.01



(e) Training loss - learning rate = 0.1



(f) Validation accuracy - learning rate = 0.1

Figure 3: Logistic regression - training loss and validation accuracy plots for different learning rates.

Question 2.2

In the second task of question 2 we implemented feed-forward neural network. We tested all hyperparameters combinations for single layer. The best combination turned out to be the one with changing the hidden size to 200 (rest hyperparameters were default). The final accuracy was equal to 0.8743. Below we show plots for this case.

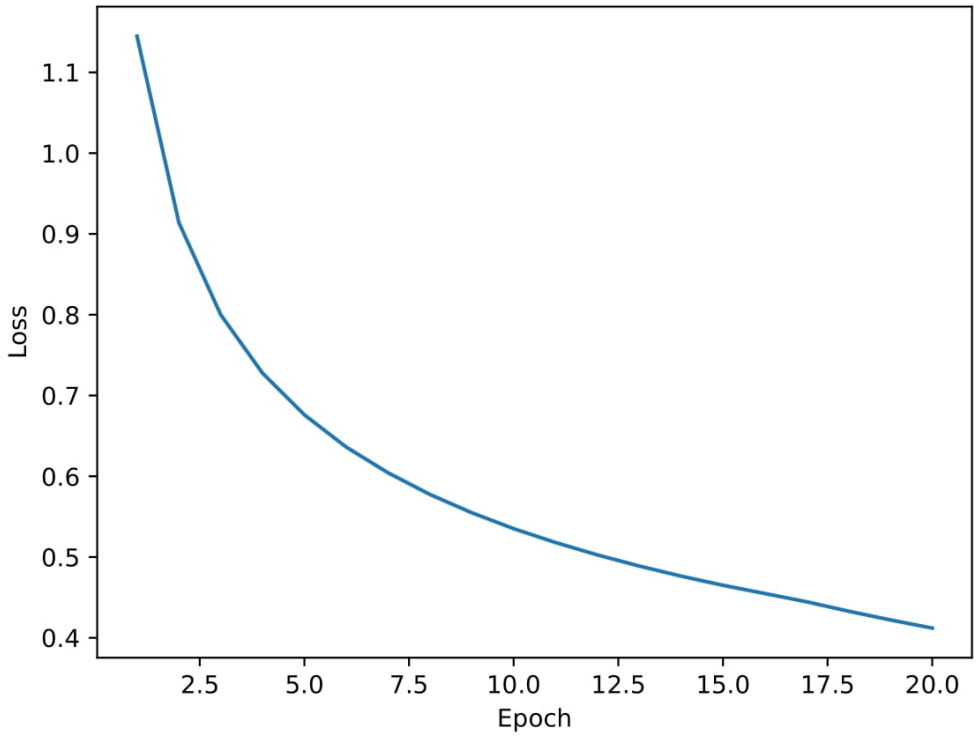


Figure 4: Training loss for feed-forward neural network - hidden size = 200

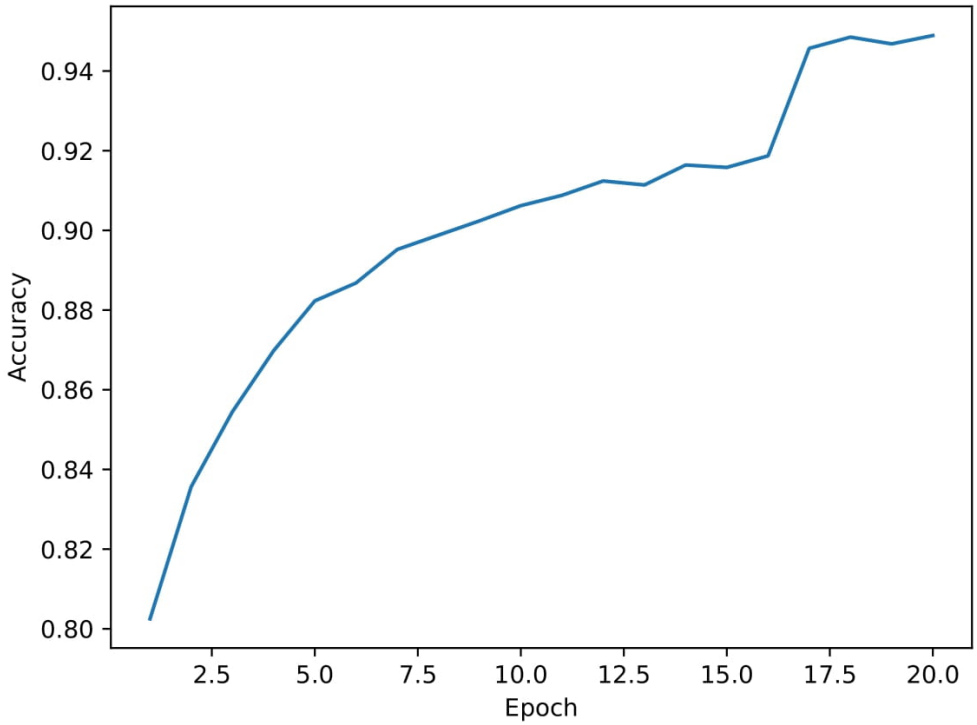


Figure 5: Validation accuracy for feed-forward neural network - hidden size = 200

Question 2.3

In third task of question 3 we had to test neural network from previous task for 2 and 3 layers (hyperparameters were default). Final accuracy is a little better for layer 2 (equal 0.8655) than for layer 3 (equal 0.8476). In case of layer 3 accuracy increases more dynamically.

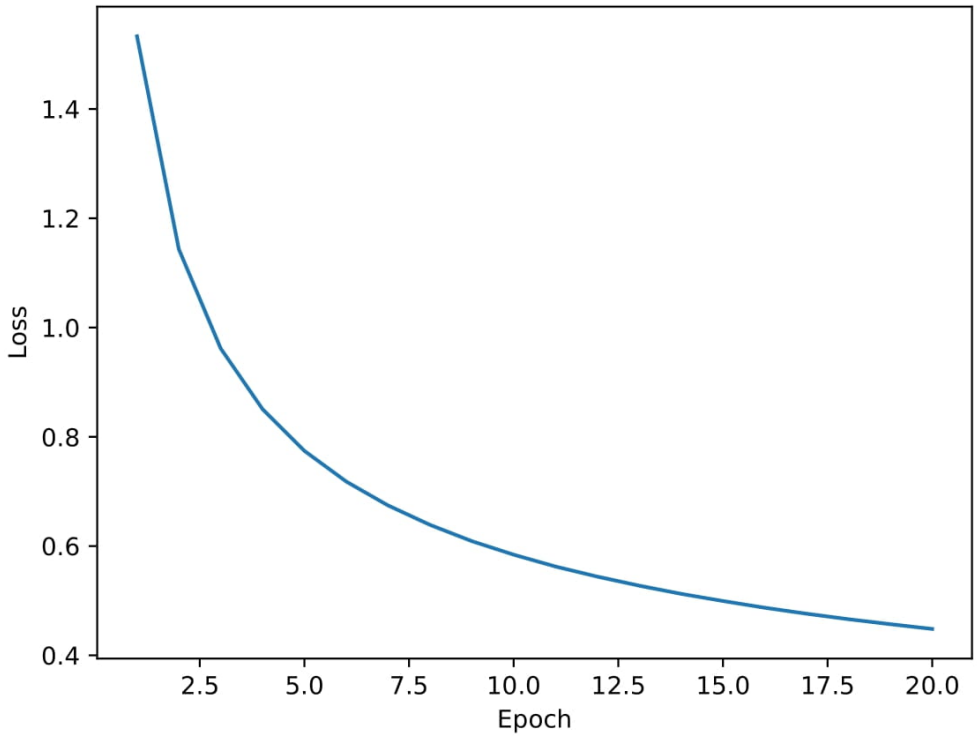


Figure 6: Training loss for 2 layers feed-forward neural network

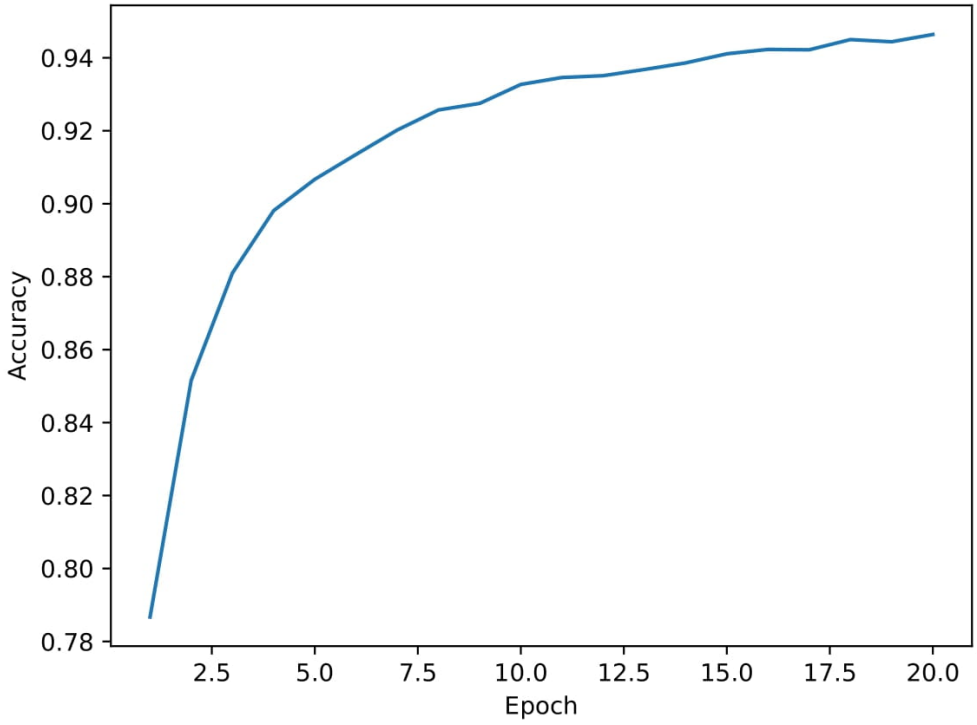


Figure 7: Validation accuracy for 2 layers feed-forward neural network

Question 3

Question 3.1

$$W_n = \begin{bmatrix} wn + \dots \\ \end{bmatrix}$$

$$g(Wx) = (WX^2) = \begin{bmatrix} W_{11} & +\dots+ & W_{18}X_8 \\ \dots & \dots & \dots \\ W_{k^n}X_1 & +\dots+ & W_{kd^{Xn}} \end{bmatrix} = \begin{bmatrix} \left(\sum_i^D W_{ni}X_i\right)^2 \\ \dots \\ \left(\sum_i W_{ki}X_i\right)^2 \end{bmatrix} =$$

$$= \begin{bmatrix} \left(\sum_i W_{ni}X_i\right)^2 + 2\sum_{i\subset j} W_{ni}X_i \dots W_{ij}X_j \\ \dots \\ \left(\sum_i W_{ki}X_i\right)^2 + 2\sum_{i\subset j} W_{hi}X_i \dots W_{kj}X_j \end{bmatrix} = \begin{bmatrix} \sum_i^D W_{ni}^2X_i^2 + 2\sum_{i\subset j} W_{1i}k_iW_{ij}X_j \\ \dots \\ \sum_i W_{ki}^2X_i^2 + 2\sum_{i\subset j} W_{1i}X_i \dots W_{1j}X_j \end{bmatrix} =$$

$$= \begin{bmatrix} W_1^{o2} & \dots & X^{o2} + 2\left(W_{11}X_1W_{12}X_2 + \dots + W_{11}X_1W_{1D}X_D + W_{12}X_2W_{13}X_3 + \dots\right) \\ \dots & \dots & +W_{12}X_bW_{1D}X_D + \dots + W_{1(D-1)}X_{D-1}W_{1D}X_D \\ W_k^{o2} & \dots & X^{o2} \dots \end{bmatrix}$$

$$A_\Theta = \begin{bmatrix} W_{11} & \dots & W_{1D} & 2W_{11}W_{12} & \dots & 2W_{11}W_{1D} & 2W_{12}W_{13} & \dots & 2W_{12}W_{1D} & \dots \\ W_{1(D-1))}W_{1D} & & & & & & & & & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & & & & & & & & & \\ W_{11} & \dots & W_{kD} & 2W_{k1}W_{k2} & \dots & 2W_{k1}W_{kD} & 2W_{k2}W_{k3} & \dots & 2W_{k2}W_{kD} & \dots \\ W_{k(D-1))}W_{kD} & & & & & & & & & \end{bmatrix}$$

$$A_\Theta = \begin{bmatrix} D & D-1 & D-1 & \dots & 1 \end{bmatrix}$$

$$\Theta(x) = \begin{bmatrix} X_1^2 \\ \dots \\ X_D^2 \\ 2X_1X_2 \\ \dots \\ 2X_1X_D \\ 2X_2X_3 \\ \dots \\ 2X_2X_8 \\ \dots \\ 2X_{D-1}X_D \end{bmatrix}$$

$D + D - 1 + D - 2 + \dots + 1 = \sum_{i=1}^{8} i = \frac{D(D+1)}{2}$

$A_\Theta \epsilon R^{k*\frac{D(D-1)}{2}}$

$\Theta(x) \epsilon R^{\frac{D(D-1)}{2}}$

Question 3.2

Question 3.3

Question 3.4