TÉCNICO LISBOA

Deep Learning

MEEC

Homework 1 Report

Authors:

99990 João Penetra joao.penetra@tecnico.ulisboa.pt
100102 Tomás Martins tomasmartins77@tecnico.ulisboa.pt

Group 63

2024/2025 – 1º Semester, P2

Contribution of each member

Each member of the group solved the following questions:

João Penetra: 1.1 a), 1.3 a), 2.2 b,c), 3.1), 3.2), 3.3)
Tomás Martins: 1.2 a,b,c,d), 2.1 a), 2.2 a), 3.1), 3.3), 3.4)

Question 1

1.1 a)

We implemented the weight update function of the Perceptron class with the help of the theoretical classes and slides. The training of 100 epochs obtained the following results:

Table 1: Accuracy scores of a single Perceptron after 100 epochs

Train Acc	0.5598
Validation Acc	0.3868
Final Test Acc	0.3743

Table 1 shows very poor results across all metrics with the final accuracy of the test dataset being just 0.3853. This shows the lack of ability of the Perceptron of understanding the complex spatial features of an image.

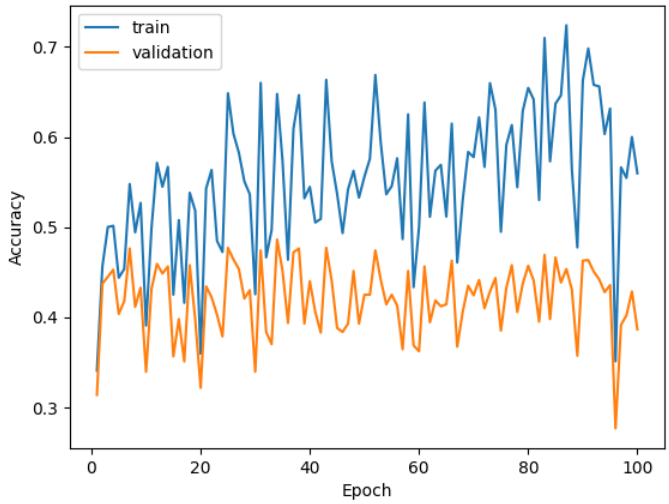


Figure 1: Training and validation accuracies as a function of the epoch number

Figure 1 highlights the limitations of the Perceptron in handling a complex dataset. The accuracy of both training and validation does not increase across epochs, indicating a lack of learning capabilities. Additionally, its performance is very unstable and as the number of epochs increase, both metrics are starting to grow apart. These facts implicit that there may not exist a linear decision boundary that separates the data, this problem is most likely non-linear, which a single Perceptron cannot solve.

1.2 a)

In this problem we implemented a non-regularized logistic regression and trained for 100 epochs with a learning rate of 0.001, the results are the following:

Table 2: Logistic regression performance on the last epoch without regularization

Train Acc	0.6694
Validation Acc	0.4623
Final Test Acc	0.4597

Table 2 shows an increase in training accuracy, although test accuracy still continues below 50%, this algorithm continues not to learn the dataset correctly.

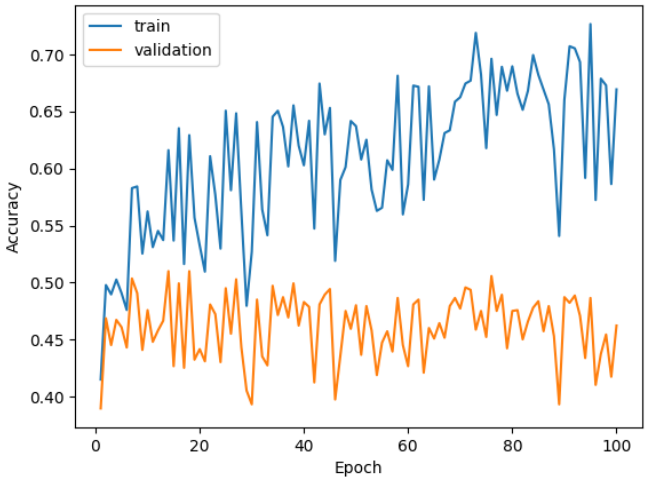


Figure 2: Logistic regression performance without regularization

In Figure 2, training accuracy seems to be increasing as the number of epochs grow, but performance continues low an unstable. Validation accuracy, on the other hand, does not grow whatsoever, demonstrating that the model is still not learning the dataset.

1.2 b)

In this problem we implemented the $l2$-norm regularization technique into the logistic regression problem but adding the gradient of the norm to the weight update equation.

Table 3: Logistic regression performance on the last epoch with $l2$-norm

Train Acc	0.5683
Validation Acc	0.4972
Final Test Acc	0.5053

Table 3 suggests a slight increase in the performance of the logistic regression, as the test accuracy reaches 50%, although still very low. However, one more noticeable change appears in Figure 3, where the validation accuracy can better follow the training accuracy, due to the smaller weights derived from the regularization term.

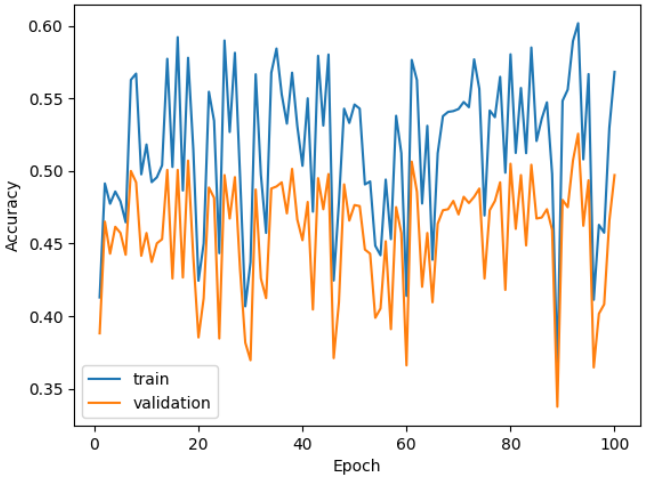


Figure 3: Logistic regression performance with $l2$-norm

1.2 c)

In this problem, we are asked to report on the $l2$-norm of the regularized and non-regularized versions of the logistic regression.

Table 4: Logistic regression performance on the last epoch without regularization

Without $l2$-norm	With $l2$-norm
23.0806	4.3217

Table 4 shows the final value of the $l2$-norm after 100 epochs on both classifiers. Without regularization, the norm reaches a lot higher values than the classifier with the regularization technique. This technique stimulate lower weights by, in the weight update equation adding a new term that diminishes the overall value of the weight by a changeable λ parameter:

$$W^{(k+1)} \longleftarrow (1 - \eta_k \lambda) W^{(k)} - \eta_k \sum_{t=1}^{N} \bigtriangledown L(W; (x_t, y_t)) \tag{1}$$

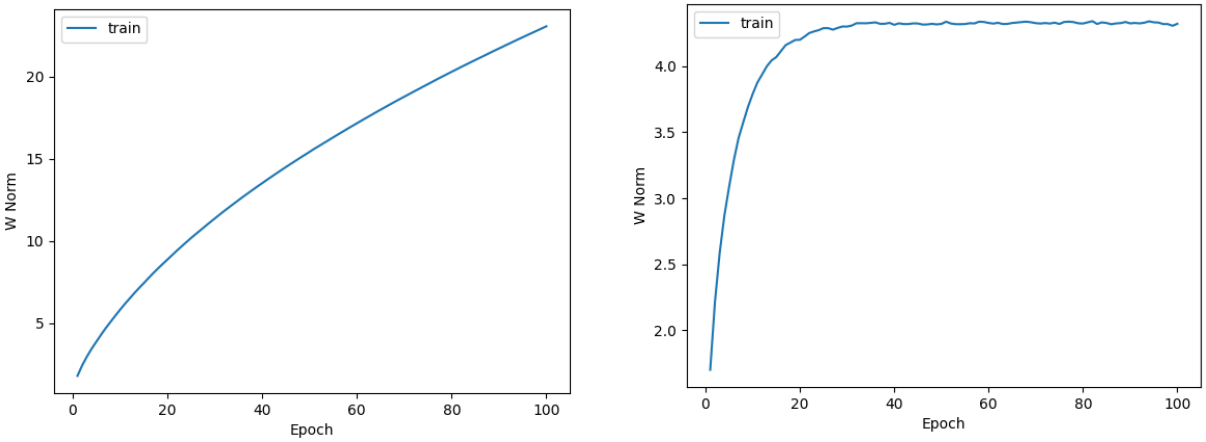
Figure 4 better shows the effect of the $l2$-norm, where the weight decay stops the overall weights from being too high, and saturates them at approximately $l2\text{-norm} = 4.3217$. Without regularization, the weights continue to increase indefinitely.

1.2 d)

The $l2$-norm adds to the loss function $L(W)$ a penalty proportional to the square of the weights:

$$\Omega(W) = \frac{1}{2}||W||_2^2 = \frac{1}{2}\sum_{i,j} W_{i,j}^2 \tag{2}$$

This term promotes smaller weights without leading them directly to zero, due to its squared nature. It balances weight distribution, preventing some weights of driving predictions.



(a) Without $l2$ regularization

(b) With $l2$ regularization

Figure 4: Effect of a regularization technique on the $l2$-norm of the weights

On the other hand, the $l1$-norm adds instead a penalty proportional to the absolute value of the weights. It also leads to smaller weights like the $l2$-norm but they can be driven to zero. This encourages sparsity in weight distribution.

$$\Omega(W) = ||W||_1 = \sum_{i,j} |W|_{i,j} \tag{3}$$

The gradient of this regularized expression is simple to calculate as it becomes a sign function (-1 for negative weights and 1 for positive weights). This norm is more robust to outliers than the $l2$-norm because less important features can be zero.

Both of these regularization techniques prevent overfitting and improve generalization of the logistic regression.

1.3 a)

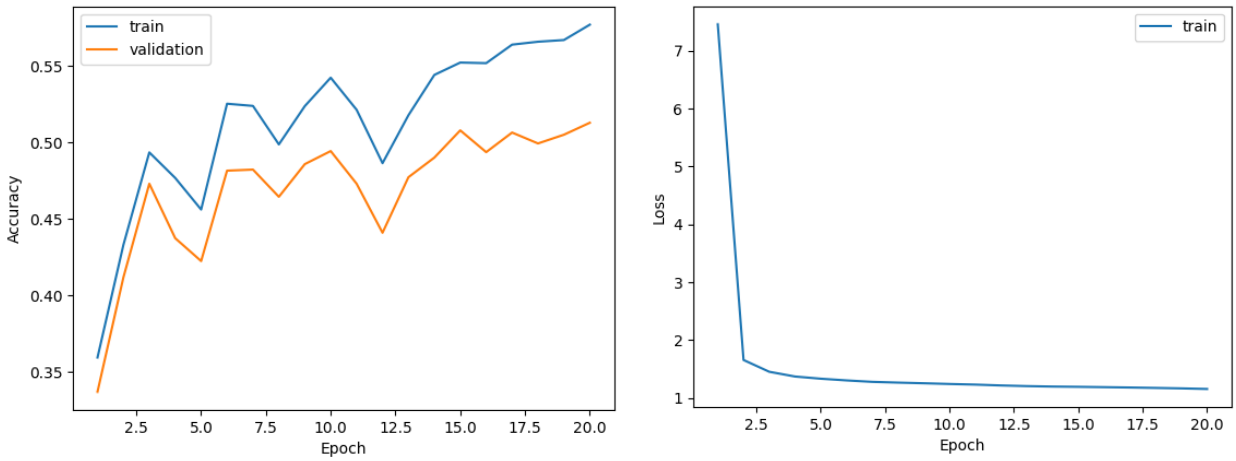
For this problem, a neural network with one hidden layer was implemented and trained to classify data using backpropagation and stochastic gradient descent. The network architecture included a ReLU activation function in the hidden layer and a softmax activation in the output layer, enabling multi-class classification.

Table 5: Scores of the MLP after 20 epochs

Loss	1.1579
Train Acc	0.5770
Validation Acc	0.5128
Final Test Acc	0.5193

The training process involved calculating the forward pass to compute predictions and the cross-entropy loss. During the backward pass, gradients of the loss with respect to the network

parameters (weights and biases) were computed layer by layer using the chain rule of calculus. The computed gradients were then used to update the parameters via stochastic gradient descent, a single example at a time. The total loss was averaged across all examples for the epoch.



(a) Accuracy (b) Loss

Figure 5: MLP model performance.

Although the results showed a slight improvement compared to the previous exercises, the difference is not that significant. The MLP has the ability to model non-linear relationship but the chosen architecture was too simple for the given task, this could be potentially addressed by increasing the number of hidden units, adding more layers, or training for more epochs.

Question 2

2.1)

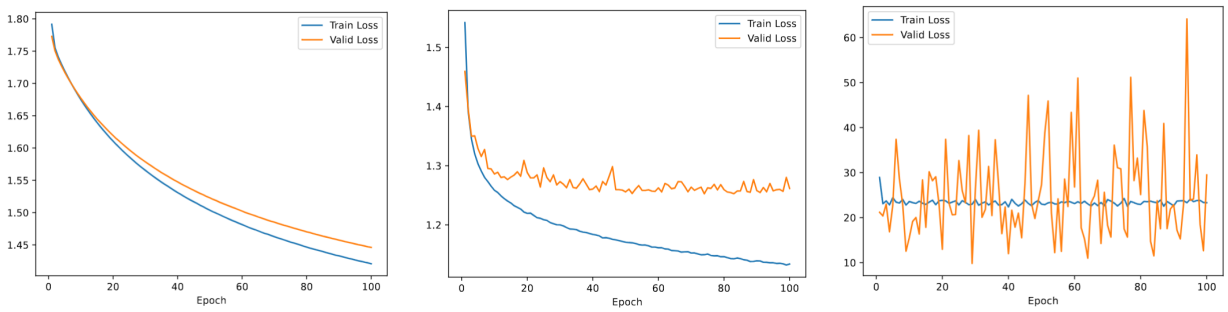
In this question, we implemented the logistic regression, but this time with the help of pytorch. We trained the model with 100 epochs and experimented different learning rates. The results are the following:

Table 6: Scores of the logistic regression after 100 epochs with different learning rates

	Learning rate 0.00001	Learning rate 0.001	Learning rate 0.1
Train loss	1.4207	1.1338	23.3119
Validation loss	1.4462	1.2616	29.4802
Validation Acc	0.4694	0.5264	0.3533
Final Test Acc	0.4623	0.5247	0.3580

From Table 6, we can see that a learning rate of 0.001 gives the best results for this problem. It achieves the highest accuracy and the lowest loss compared to other learning rates. On the other hand, the loss for a learning rate of 0.1 is much higher than the losses for the other values.

In Figure 6, the lowest learning rate results in smaller steps during gradient descent. This makes the loss curve smoother, but the accuracy does not reach higher values as with a learning rate of 0.001. On the other hand, with a learning rate of 0.1, the validation loss becomes very unstable because the larger steps in gradient descent cause more fluctuations and spikes between epochs. These large updates cause the model to overshoot the minimum, neglecting the model from ever finding an optimal solution.



(a) Learning rate 0.00001 (b) Learning rate 0.001 (c) Learning rate 0.1

Figure 6: Logistic regression loss with different learning rates

The best model is achieved with a learning rate of 0.001, which balances the advantages of the other two rates. A lower learning rate allows the gradient descent to move more precisely toward the minimum but slows down the process, resulting in lower accuracy because it hasn't fully reached the minimum. It also increases the risk of the model never reaching a global minimum, specially with this number of epochs. In contrast, a learning rate of 0.001 makes the gradient descent slightly less stable but ultimately leads to a lower loss value and higher accuracy.

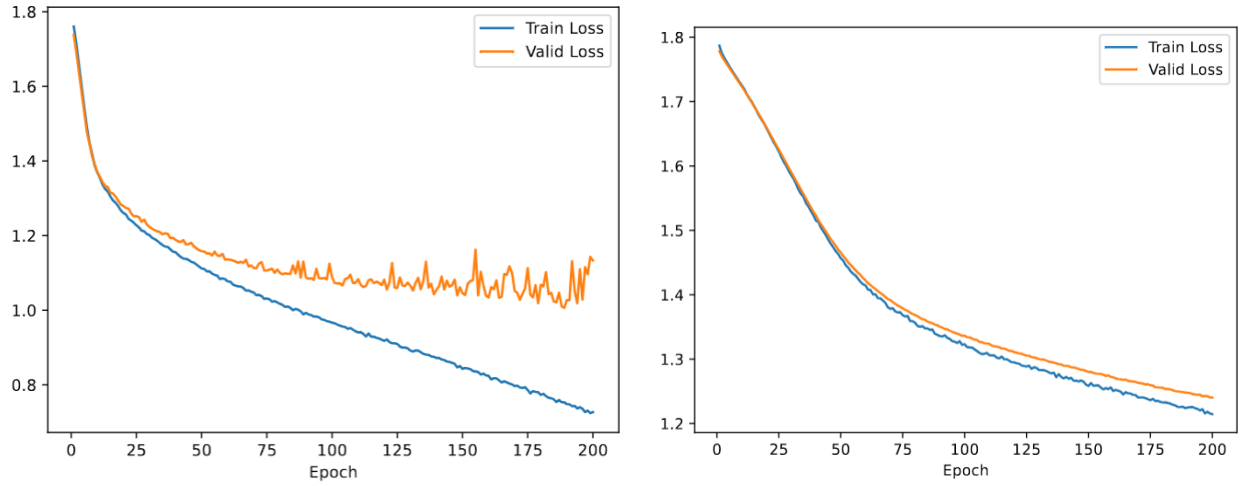
2.2 a)

In this section, we experimented the effects of changing the training batch on the model's performance.

Table 7: Scores of the MLP after 20 epochs

	Batch size 64	Batch size 512
Train loss	0.7270	1.2146
Validation loss	1.1336	1.2404
Validation Acc	0.5819	0.5903
Final Test Acc	0.5730	0.5350
Time	2 min 20 sec	52 sec

In Table 7, increasing the training batch leads to lower accuracies and higher losses, but the computation time decreases considerably.



(a) Batch size 64 (b) Batch size 512

Figure 7: MLP loss with different batch sizes

Figure 7 shows that using smaller batch sizes decreases training stability. A model with a batch size of 512 performs 8 times more weight updates per epoch compared to a batch size of 64. While larger batch sizes can lead to faster computation per step, they result in slower convergence and lower performance for the same number of epochs.

2.2 b)

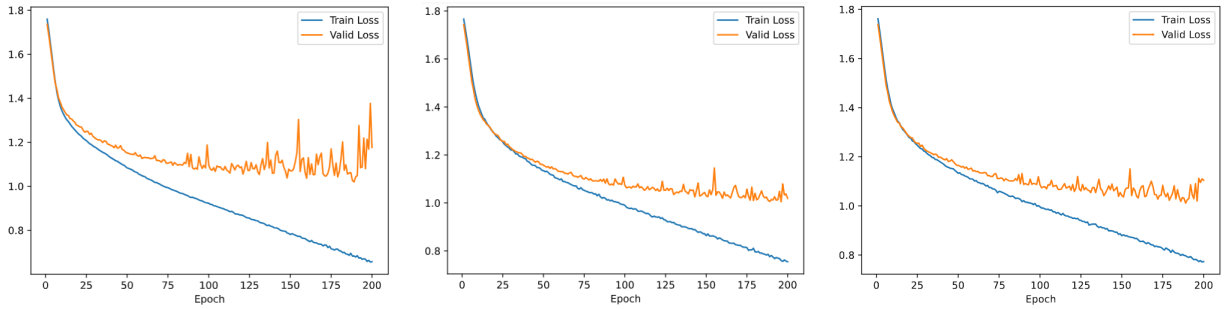
For this question we conducted experiments on our MLP model using dropout rates of 0.01, 0.25, and 0.50 while keeping all other parameters at their default settings.

Table 8: Scores of the MLP after 200 epochs with different dropout rates

	Dropout 0.01	Dropout 0.25	Dropout 0.5
Train loss	0.6571	0.7150	0.7732
Validation loss	1.1763	1.1287	1.1042
Validation Acc	0.5769	0.5912	0.5769
Final Test Acc	0.5683	0.5787	0.5777

From the results, we observed that dropout rate of 0.01 yielded slightly worse accuracy and higher loss compared to 0.25 and 0.50, which both got similar results. By analyzing the graphs we see that at higher dropout rates (0.50), the loss stabilizes more over epochs, but at lower dropout rates (0.01), the loss shows less stability and more fluctuations. However, the validation loss between 0.25 and 0.5 remained very similar.

A model is considered to be overfitting when the training and validation losses start to diverge by a considerable amount, meaning the model performs much better on the training data than on the validation data. This happens because the model closely aligns with the training data but fails to generalize, leading to poorer performance on unseen data.



(a) Dropout 0.01 (b) Dropout 0.25 (c) Dropout 0.50

Figure 8: MLP loss with different dropout rates.

From Figure 8, we find that higher dropout rates improves stability by preventing overfitting. Dropout works by randomly deactivating certain neurons during training, which prevents the model from relying too heavily on specific features or nodes. As a result, the model learns more generalized patterns and performs better on the validation dataset. A dropout of between 0.25 and 0.5 appears to balance regularization and learning capability of the MLP model.

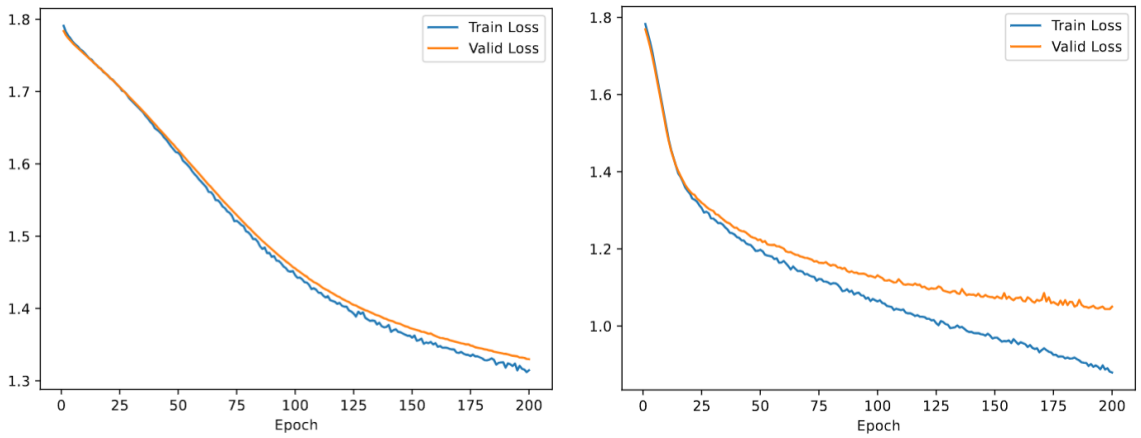
2.2 c)

For this question we study the effects on momentum on our MLP model. All other parameters were kept at their default settings.

Table 9: Scores of the MLP after 200 epochs with and without momentum.

	No Momentum	Momentum 0.9
Train loss	1.3145	0.8795
Validation loss	1.3299	1.0500
Validation Acc	0.4786	0.5962
Final Test Acc	0.5037	0.5910

Without momentum, training and validation losses were consistently higher compared to when momentum was applied. Using momentum resulted in significantly better accuracy, with the accuracy curve stabilizing faster and reaching a higher value than in the no-momentum scenario. This happens because momentum regularization helps the optimizer update in the direction of steepest descent by combining the current gradient with a fraction of the previous update.



(a) No momentum (b) Momentum 0.9

Figure 9: MLP loss for different momentum values.

However, with momentum, training and validation losses began to diverge as training progressed, indicating the onset of overfitting. This suggests that while momentum initially improves convergence speed and performance, it may eventually lead to overfitting.

Question 3

For the following questions, ChatGPT was used to help understand the problem and assist in developing reasoning for the problem solution.

3.1)

In this question, we want to write h as a function of $A_\theta \phi(x)$ for a certain feature transformation ϕ such that: $R^D \longrightarrow R^{\frac{(D+1)(D+2)}{2}}$ independent of θ. For this purpose, we can first express the activation function $g(z) = z(1-z)$ as $g(z) = z - z^2$. Knowing this, h can be written as:

$$h = g(Wx + b) = (Wx + b) \odot (1 - (Wx + b)) \tag{4}$$

Expanding for each hidden unit, and defining $z_i = \sum_{j=1}^{D} W_{ij}x_j + b_i$:

$$h_i = (\sum_{j=1}^{D} W_{ij}x_j + b_i) \cdot (1 - \sum_{j=1}^{D} W_{ij}x_j + b_i)^2 \tag{5}$$

Lets see an example for $D = 1$ and $K = 1$, it becomes:

$$h = (W_1x_1 + b)(1 - (W_1x_1 + b)) = W_1x_1 + b - W_1^2x_1^2 - 2W_1x_1b - b^2 \tag{6}$$

Knowing this, the term $\phi(x)$ needs to have $dim(\phi(x)) = \frac{(D+1)(D+2)}{2}$ to handle the linear, quadratic and squared terms that x has in the transformation. It can be written as:

$$\phi(x) = \{1, x_1, x_1^2\}^T \tag{7}$$

On the other hand, the matrix A_θ has the following implementation:

$$A_\Theta = \begin{bmatrix} b_1 & W_{11} & W_{11}^2 \end{bmatrix} \tag{8}$$

For another example with $D = 2$ and $K = 2$, $\phi(x)$ would be:

$$\phi(x) = \{1, x_1, x_2, x_1^2, x_1x_2, x_2^2\}^T \tag{9}$$

Where 1 is to handle the bias term, $x_1, ..., x_D$ are the linear terms and then we handle the quadratic terms, that include the self-term of the squared version like x^2. The matrix A_Θ becomes:

$$A_\Theta = \begin{bmatrix} b_1 & W_{11} & W_{12} & W_{11}^2 & W_{11}W_{12} & W_{12}^2 \\ b_2 & W_{21} & W_{22} & W_{21}^2 & W_{21}W_{22} & W_{22}^2 \end{bmatrix} \tag{10}$$

But for D=3 and K=3, both terms $\phi(x)$, A_Θ become:

$$\phi(x) = \{1, x_1, x_2, x_3, x_1^2, x_1x_2, x_1x_3, x_2^2, x_2x_3, x_3^2\}^T \tag{11}$$

$$A_\Theta = \begin{bmatrix} b_1 & W_{11} & W_{12} & W_{13} & W_{11}^2 & W_{11}W_{12} & W_{11}W_{13} & W_{12}^2 & W_{12}W_{13} & W_{13}^2 \\ b_2 & W_{21} & W_{22} & W_{23} & W_{21}^2 & W_{21}W_{22} & W_{21}W_{23} & W_{22}^2 & W_{22}W_{23} & W_{23}^2 \\ b_3 & W_{31} & W_{32} & W_{33} & W_{31}^2 & W_{31}W_{32} & W_{31}W_{33} & W_{32}^2 & W_{32}W_{33} & W_{33}^2 \end{bmatrix} \tag{12}$$

Generalizing, for an input $x \in \mathbb{R}^D$, the feature transformation $\phi(x)$ with dimension $\frac{(D+1)(D+2)}{n}$, becomes:

$$\phi(x) = \{1, x_1, x_2, ..., x_D, x_1^2, x_1x_2, ..., x_D^2\}^T \tag{13}$$

For $\theta = (W, b, v, v_0)$, weights W $\in \mathbb{R}^{K\times D}$ and biases b $\in \mathbb{R}^K$, we have:

$$A_\Theta = \begin{bmatrix} b_1 & W_{11} & W_{12} & ... & W_{1D} & W_{11}^2 & W_{11}W_{12} & ... & W_{1D}^2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ b_K & W_{K1} & W_{K2} & ... & W_{KD} & W_{K1}^2 & W_{K1}W_{K2} & ... & W_{KD}^2 \end{bmatrix} \tag{14}$$

3.2)

In this question, in order to write $\hat{y}$ as a linear transformation of $\phi(x)$ we first substitute $h = A_\Theta\phi(x)$ into the equation for $\hat{y}$:

$$\hat{y} = v^T h + v_0 = v^T A_\Theta \phi(x) + v0 \tag{15}$$

By defining $c_\Theta = [v_0; v^T A_\Theta]^T$ and extending $\phi(x)$ to include a constant term to account for the bias term v_0, $\hat{y}$ can be expressed as:

$$\hat{y} = c_\Theta^T \phi(x) \tag{16}$$

Which shows that $\hat{y}$ is a linear transformation of $\phi(x)$. Although, while $\hat{y}$ appears to be linear in terms of c_Θ, the model is not linear in the original parameters $\Theta = (W, b, v, v_0)$. This is because the mapping from Θ to c_Θ is non-linear, A_Θ depends quadratically on the terms W and b.

3.3)

3.4)

From the squared loss defined with the training data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$ with $N > \frac{(D+1)(D+2)}{2}$, we want to minimize:

$$L(c_\Theta; \mathcal{D}) = \frac{1}{2} \sum_{n=1}^{N} \left(\hat{y}(x_n; c_\Theta) - y_n\right)^2 \tag{17}$$

We previously defined $\hat{y}(x; c_\theta) = c_\theta^T \phi(x)$, substituting it in the previous expression we get:

$$L(c_\Theta; \mathcal{D}) = \frac{1}{2} \sum_{n=1}^{N} \left(c_\theta^T \phi(x) - y_n\right)^2 \tag{18}$$

If we define a matrix $X \in \mathbb{R}^{N \times \frac{(D+1)(D+2)}{2}}$ where its rows are the feature vectors $\{\phi(x_n)\}_{n=1}^{N}$, then we can define $\hat{y} = Xc_\theta$. Rewriting the loss function in terms of matrices and imputing this new matrix into it, we get:

$$L(c_\Theta; \mathcal{D}) = \frac{1}{2} ||Xc_\theta - y||^2 \tag{19}$$

To minimize the loss function, we need to differentiate with respect to c_θ and equal it to zero.

$$\frac{\partial L}{\partial c_\theta} = \mathbf{X}^T(\mathbf{X}c_\theta - \mathbf{y}) = 0 \tag{20}$$

Which gives us:

$$X^T X c_\theta = X^T y \longleftrightarrow \hat{c_\theta} = (X^T X)^{-1} X^T y \tag{21}$$

This way, we can solve for $\hat{c_\theta}$ just like a linear regression. The reparametrization into c_θ makes this problem linear and convex. Normally, in Neural Networks, global minimization is intractable due to its non-linear transformations in the activation function (ReLu, Tanh, ...) that make the function non-convex. This reparameterization transforms it into a convex optimization problem, that has a global minimum, which easier to find.