TÉCNICO LISBOA

Deep Learning

Masters Degree in Mechanical Engineering

Homework 1 [EN]

Authors:

Francisco Pinto (ISTID 089888) francisco.v.pinto@tecnico.ulisboa.pt
Alexandre Gonçalves (ISTID 100121) alexandre.n.goncalves@tecnico.ulisboa.pt

Group 81

2024/2025 – 2nd Quarter

Introduction

This report contains our solutions to Homework 1 for the Deep Learning course offered at IST during the 2024-25 academic year. .

Contribution Statement:

- **Alexandre Gonçalves:** Implemented the Perceptron and logistic regression models, contributed to the PyTorch implementations, and analyzed regularization effects (Questions 1 and 2).
- **Francisco Pinto:** Designed and implemented the multi-layer perceptron with backpropagation, conducted hyperparameter tuning, and performed theoretical analyses for Question 3.

We certify that the work presented here reflects the collective efforts of all team members.

Exercise 1

1.1

a)

During the training process, the train accuracy reaches a global maximum of 0.6960 on the 99th epoch, the validation accuracy's biggest value is 0.4808 on the 85th epoch and the attained final test accuracy is 0.3853. In Figure 1 we can find a plot containing the train and validation accuracies as a function of the epoch number.

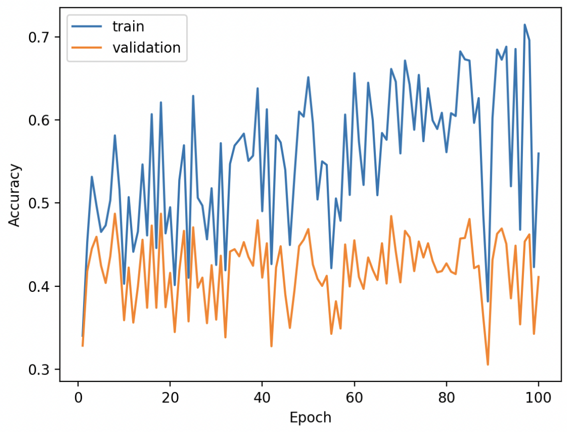

Figure 1: Train and validation accuracies as a function of the epoch number

As we can see, the two lines start growing further apart just after epoch 30, which might indicate the model is beginning to overfit.

1.2

a)

During the training process, the train accuracy reaches a global maximum of 0.7268 on the 95th epoch, the validation accuracy's biggest value is 0.5100 on the 14th epoch and the attained final test accuracy is 0.4597.

b)

During the training process, the train accuracy reaches a global maximum of 0.5746 on the 93th epoch, the validation accuracy's biggest value is 0.5036 on the 45th epoch and the attained final test accuracy is 0.4997. In Figures 2 and 3 we can find the train and validation accuracies as a function of the epoch number with and without regularization.

As we can see, the regularization lowers the discrepancy between the train and validation accuracies, and, therefore, the overfit.

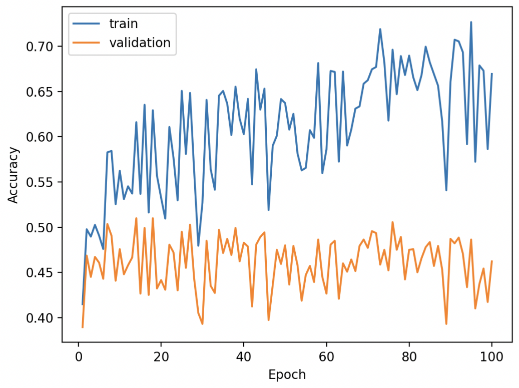

Figure 2: Training without regularization.

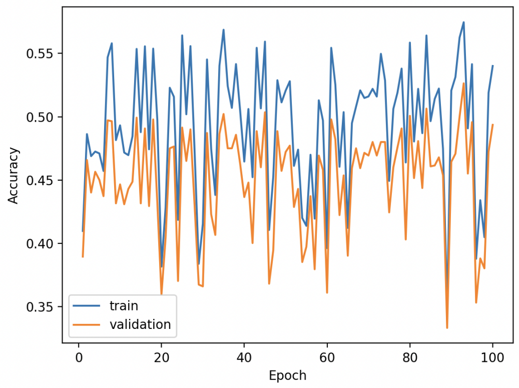

Figure 3: Training with regularization.

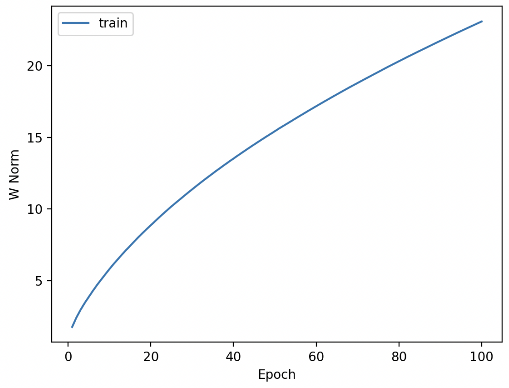

Figure 4: l_2-norm without regularization.

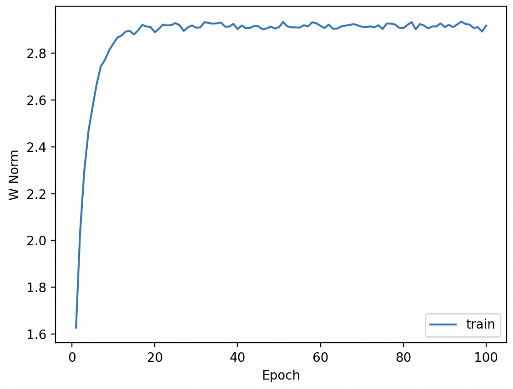

Figure 5: l_2-norm with regularization.

c)

As we can see in Figures 4 and 5, while the l_2-norm grows indefinitely without regularization, finishing training with a value of 23.0806, it plateaus early on at about 2.91 when using regularization.

d)

The l_1 regularization doesn't penalize big weights as much but can bring the weights to zero if necessary, therefore, we would expect the weights to be sparser and less evenly distributed at the end of training.

1.3

a)

In Figures 6 and 7 we can find the plots containing the evolution of the training and validation accuracies and the training loss, respectively. The final obtained test accuracy was 0.1750.

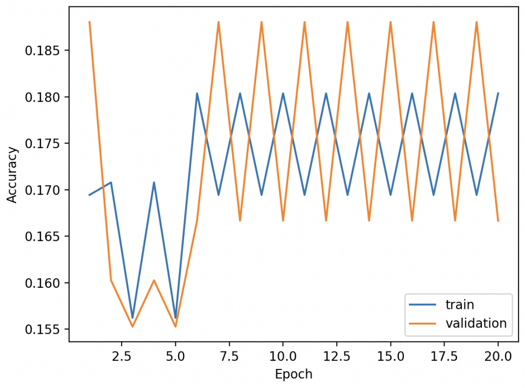

Figure 6: Training and validation accuracies.

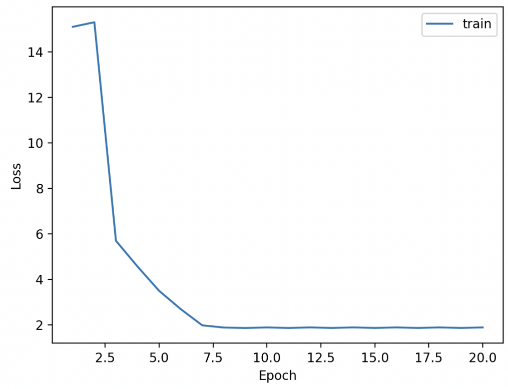

Figure 7: Training loss.

Exercise 2

1

Best parameters

After trying out the different suggested learning rate values, the best result was achieved with a learning rate of 0.001. This model's final test accuracy is 0.5300 and, as we can see on Figure 10, the best validation accuracy this model achieves is 0.5399.

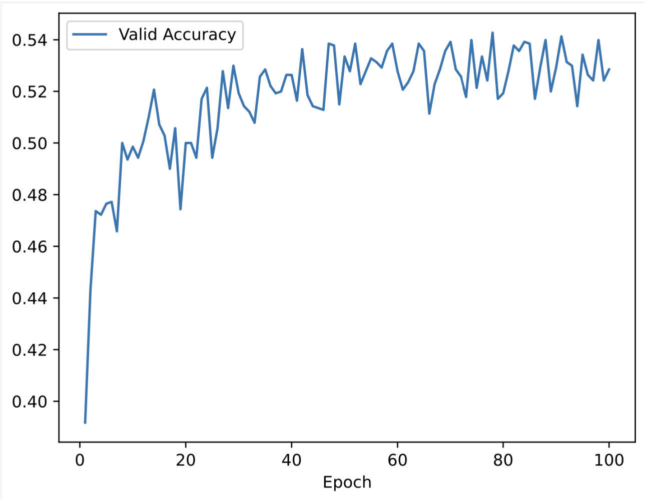

Figure 8: Train and validation accuracies as a function of the epoch number for a learning rate of 0.001.

Comparing the different loss functions

In Figures 9, 10 and 11 we can find the loss curves corresponding to the different learning rates.

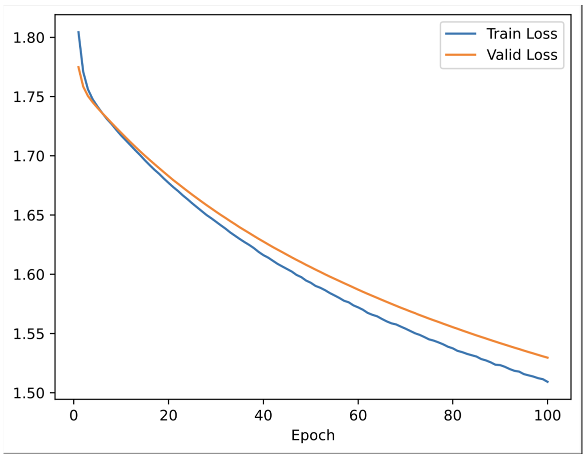

Figure 9: Train and validation losses for $l_r = 0.00001$.

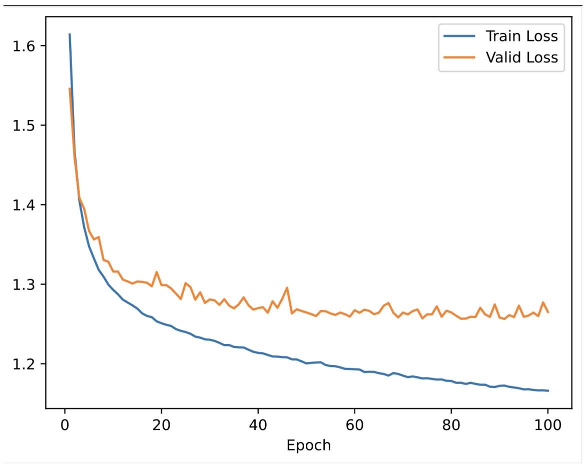

Figure 10: Train and validation losses for $l_r = 0.001$.

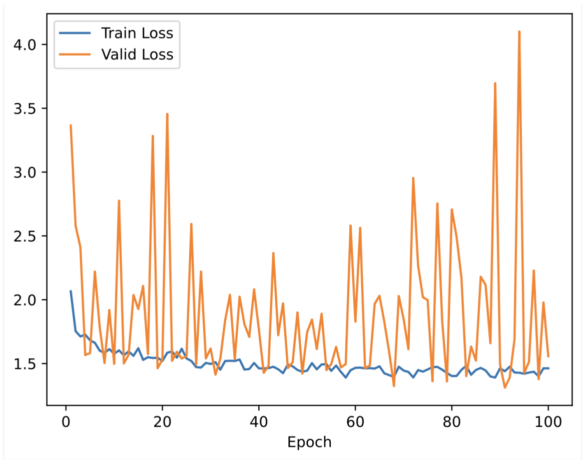

Figure 11: Train and validation losses for $l_r = 0.1$.

As we can see, the higher learning rate the more oscillations our loss curves display (specially when it comes to the validation set).

We can also notice that, although we can converge in a steadier way, having a learning rate that is too small makes this convergence very slow. By looking at the graphs we see that, although it hasn't stopped converging, our training with $l_r = 0.00001$ has the same loss at epoch 100 as the training with $l_r = 0.01$, which stopped converging a long ago, due to the high learning rate.

In our best model ($l_r = 0.001$), the loss curves indicate that the validation performance stopped increasing at about epoch 50, which might indicate that our model would benefit from some dropout or that we could implement early stopping.

Batch Size	Time	Final Validation Accuracy	Final Test Accuracy
64	2:39	0.6041	0.5893
512	1:33	0.6004	0.6067

Table 1: Performance values for different bach sizes.

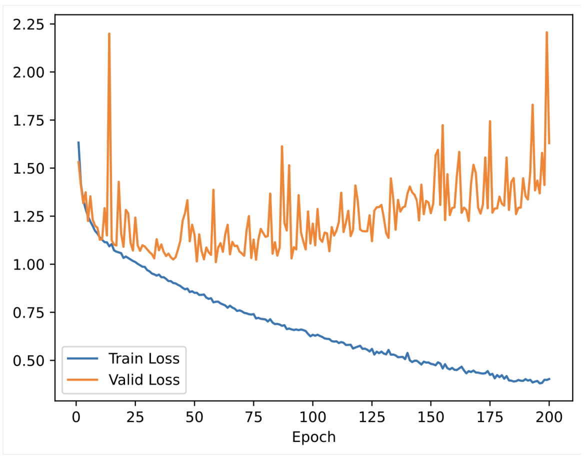

Figure 12: Train and validation losses for bs = 64.

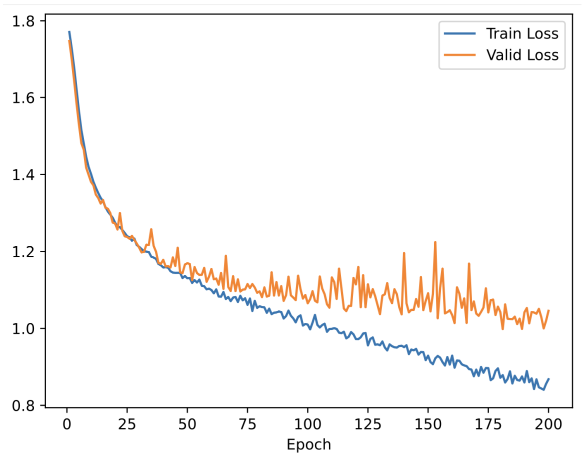

Figure 13: Train and validation losses for bs = 512.

2

Batch size

We can find the loss curves for both experiments on Figures 12 and 13 and the performance values on table 1. As expected, a higher batch size means the program runs faster, since more examples are being processed in parallel and, therefore, the number of steps required to process the full dataset in one epoch is reduced, but it also converges slower as shows more oscillations, since the gradient isn't calculated as carefully.

Dropout

We can find the loss curves for the experiments on Figures 14, 15 and 16 and the different performance values on table 2. As expected, the higher the dropout, the longer it takes for the model to converge but also the less it overfits (it takes longer for the validation loss to start increasing).

Dropout	Final Validation Accuracy	Final Test Accuracy
0.01	0.5648	0.5533
0.25	0.5769	0.5523
0.5	0.6026	0.6077

Table 2: Performance for different dropout values.

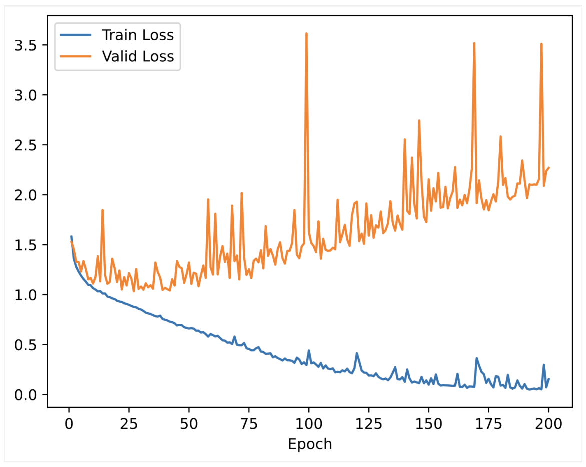

Figure 14: Train and validation losses for dropout = 0.01.

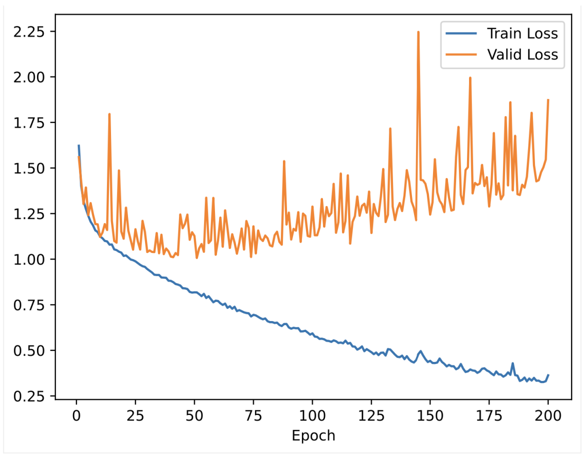

Figure 15: Train and validation losses for dropout = 0.25.

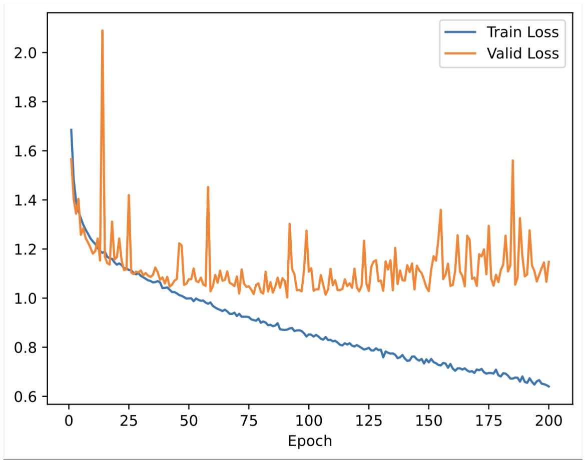

Figure 16: Train and validation losses for dropout = 0.5.

Momentum	Final Validation Accuracy	Final Test Accuracy
0	0.5819	0.5793
0.9	0.6011	0.5967

Table 3: Performance for different momentum values.

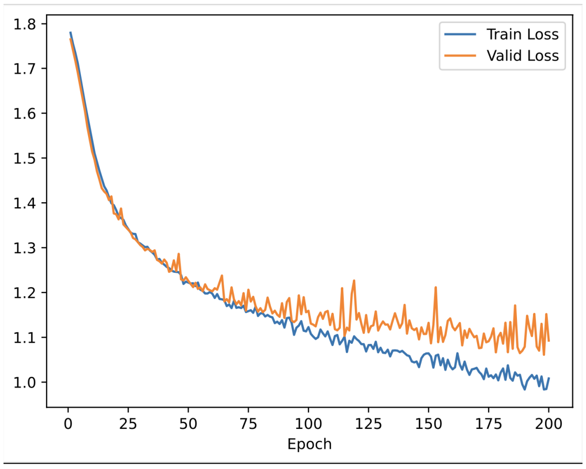

Figure 17: Train and validation losses with momentum 0.

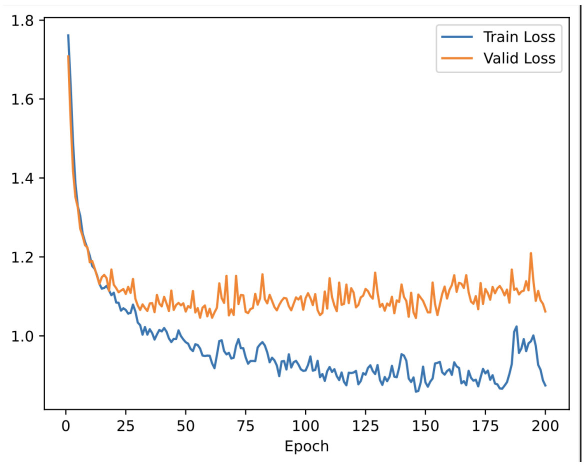

Figure 18: Train and validation losses with momentum 0.9.

Momentum

We can find the loss curves for the experiments on Figures 17 and 18 and the performance values on table 3. As we can see by the loss curves, a higher momentum may help the model converge faster on the first epochs, but it also leads to more oscillations and to a higher difficulty in convergence on higher epochs, possibly limiting the maximum performance achieved.

Exercise 3

Exercise 3.1

The hidden layer is defined as:

$$h = g(Wx + b),$$

where $g(z) = z(1 - z)$. With $u = Wx + b$:

$$h_i = g(u_i) = u_i(1 - u_i),$$

where:

$$u_i = w_i^\top x + b_i,$$

and $w_i^\top$ is the i-th row of W, while b_i is the i-th component of b.

Substituting u_i:

$$h_i = (w_i^\top x + b_i)\big[1 - (w_i^\top x + b_i)\big].$$

Expanding the product:

$$h_i = (w_i^\top x + b_i) - (w_i^\top x + b_i)^2.$$

Expanding the square:

$$(w_i^\top x + b_i)^2 = (w_i^\top x)^2 + 2b_i(w_i^\top x) + b_i^2.$$

Therefore:

$$h_i = w_i^\top x + b_i - \big((w_i^\top x)^2 + 2b_i(w_i^\top x) + b_i^2\big).$$

Distributing the negative sign:

$$h_i = (b_i - b_i^2) + (1 - 2b_i)(w_i^\top x) - (w_i^\top x)^2.$$

Focusing only on the quadratic term $(w_i^\top x)^2$:

$$(w_i^\top x)^2 = (x^\top w_i)(w_i^\top x) = x^\top (w_i w_i^\top) x.$$

With $W_i := w_i w_i^\top$, which is a $D \times D$ rank-1 matrix, we write:

$$(w_i^\top x)^2 = x^\top W_i x = \langle\langle W_i, xx^\top \rangle\rangle,$$

where $\langle\langle A, B \rangle\rangle = \mathrm{Tr}(A^\top B)$ is the Frobenius inner product.

So, $(w_i^\top x)^2$ can be viewed as the inner product between the matrix W_i and the outer product $xx^\top$.

Expressing h_i with a feature map $\phi(x)$:

The expression for h_i includes a constant term $(b_i - b_i^2)$, linear terms $(1 - 2b_i)(w_i^\top x)$, and quadratic terms $-(w_i^\top x)^2$. To represent h_i as a *linear* combination of fixed, parameter-independent features of x, we construct a feature map $\phi(x)$ that includes:

$$1, \quad x_1, x_2, \ldots, x_D, \quad x_1^2, x_2^2, \ldots, x_D^2, \quad x_1x_2, x_1x_3, \ldots, x_{D-1}x_D.$$

In vector form:

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_D^2 \\ x_1 x_2 \\ x_1 x_3 \\ \vdots \\ x_{D-1} x_D \end{bmatrix} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}.$$

This vector $\phi(x)$ is independent of the parameters W and b.

The coefficients for h_i, represented by the vector a_i, include terms derived from the i-th row of W:

$$a_i = \begin{bmatrix} w_{i1}^2 \\ w_{i2}^2 \\ \vdots \\ w_{iD}^2 \\ w_{i1} w_{i2} \\ w_{i1} w_{i3} \\ \vdots \\ w_{i(D-1)} w_{iD} \end{bmatrix}.$$

Stacking all $a_i^\top$ for $i = 1, \ldots, K$ into a matrix A_Θ, we have:

$$A_\Theta = \begin{bmatrix} a_1^\top \\ a_2^\top \\ \vdots \\ a_K^\top \end{bmatrix} \in \mathbb{R}^{K \times \frac{(D+1)(D+2)}{2}}.$$

Final representation:

The activation function $g(z) = z(1-z)$ turns h_i into a combination of constant, linear, and quadratic terms of x. By organizing these terms, the network essentially works like a linear model.

This means:

$$h_i = a_i^\top \phi(x),$$

and for all hidden units together:

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

Exercise 3.2

The hidden layer output can be expressed as:

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

where $\phi : \mathbb{R}^D \to \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$ is independent of Θ.

And, as said in the exercise, the network output is:

$$\hat{y}(x;\Theta) = v^\top h + v_0.$$

Substituting $h = A_\Theta \phi(x)$ into the expression for $\hat{y}(x;\Theta)$:

$$\hat{y}(x;\Theta) = v^\top (A_\Theta \phi(x)) + v_0.$$

This simplifies to:

$$\hat{y}(x;\Theta) = (v^\top A_\Theta)\phi(x) + v_0.$$

Bias Term

The feature map $\phi(x)$ includes a constant feature, which accounts for the bias term v_0. To incorporate v_0 into the linear expression:

$$c_\Theta := A_\Theta^\top v + (\text{bias adjustment for } v_0).$$

So, the network output can be expressed as:

$$\hat{y}(x;\Theta) = c_\Theta^\top \phi(x).$$

The resulting model shows that the output $\hat{y}(x;\Theta)$ is a linear transformation of the feature map $\phi(x)$.

However:

- The model is linear in $\phi(x)$ because $\hat{y}(x;\Theta)$ is a product between c_Θ and $\phi(x)$.
- The model is not linear in Θ because c_Θ depends on A_Θ, and all entries of A_Θ are quadratic in terms of the original weights W.

So, while $\hat{y}(x;\Theta)$ can be expressed as a linear transformation of the input feature map $\phi(x)$, the dependency on Θ introduces non-linearity due to the quadratic nature of A_Θ.

Exercise 3.3

We know that the network output can be expressed as:

$$\hat{y}(x;\Theta) = v^\top h = \sum_{i=1}^{K} v_i h_i,$$

where h_i represents the hidden layer activations. From previous results, we have:

$$h_i = \langle\langle w_i w_i^\top, xx^\top \rangle\rangle,$$

where w_i is the weight vector for the i-th hidden unit. Substituting this into the expression for $\hat{y}$, we get:

$$\hat{y}(x;\Theta) = \sum_{i=1}^{K} v_i \langle\langle w_i w_i^\top, xx^\top \rangle\rangle.$$

Reorganizing the terms, this can be written as:

$$\hat{y}(x;\Theta) = \langle\langle C, xx^\top \rangle\rangle,$$

where $C \in \mathbb{R}^{D\times D}$ is a symmetric matrix defined as:

$$C = W^\top \mathrm{Diag}(v) W,$$

with:

- $W \in \mathbb{R}^{K\times D}$ being the matrix of hidden layer weights,
- $\mathrm{Diag}(v)$ a diagonal matrix formed from the output weights v.

To relate C to c_Θ, we use the vectorization operator $\mathrm{vech}(C)$, which collects the elements of the lower triangular part of C into a vector.

$$c_\Theta = \mathrm{vech}(C).$$

The set of possible C matrices is then:

$$\{C = W^\top \mathrm{Diag}(v) W \mid W \in \mathbb{R}^{K\times D}, v \in \mathbb{R}^K\}.$$

This corresponds to the set of symmetric matrices of rank at most K.

When $K \geq D$, the set of rank-K symmetric matrices spans the entire space of symmetric $D\times D$ matrices. In this case, for any $c \in \mathbb{R}^{\frac{D(D+1)}{2}}$, we can find a matrix C such that $\mathrm{vech}(C) = c$. This ensures that we can recover the original parameters $\Theta = (W, v)$ up to ϵ-precision.

To find the parameters $\Theta = (W, v)$ from a given c, we start by constructing the symmetric matrix C from c using the reverse of the vech operation, which reconstructs a full matrix from its vectorized lower triangular part. Next, we need to decompose C using eigenvalue decomposition. This decomposition gives:

$$C = W^\top \mathrm{Diag}(\lambda) W,$$

where W is the matrix of eigenvectors, and λ are the eigenvalues. Then, set the output weights v as the diagonal entries of $\mathrm{Diag}(\lambda)$, ensuring they are non-negative for consistency. Finally, adjust W and v to ensure the precision condition $\|c_\Theta - c\| < \epsilon$ is met.

If $K < D$, the rank of C is at most K, which means it cannot span the entire space of symmetric $D \times D$ matrices. In this case, it may not be possible to find a Θ such that c_Θ exactly matches c.

When $K \geq D$, the model can equivalently be parametrized by c instead of Θ, as we can always construct a symmetric matrix C such that:

$$\mathrm{vech}(C) = c.$$

However, when $K < D$, the rank constraint on C means that an exact parametrization is not always possible.

Exercise 3.4

The loss $L(c_\Theta; \mathcal{D})$ is quadratic in c_Θ. To minimize the loss, we differentiate L with respect to c_Θ and set the gradient to zero:

$$\frac{\partial L}{\partial c_\Theta} = \sum_{n=1}^{N} \phi(x_n) \left(c_\Theta^\top \phi(x_n) - y_n \right) = 0.$$

This can be rewritten in matrix form as:

$$X^\top X c_\Theta = X^\top y,$$

where:

- $X \in \mathbb{R}^{N \times \frac{(D+1)(D+2)}{2}}$ is the feature matrix with rows $\phi(x_n)$,
- $y \in \mathbb{R}^N$ is the vector of target values y_n,
- $c_\Theta \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$ is the vector of coefficients to solve for.

Since X has rank $\frac{(D+1)(D+2)}{2}$, the Gram matrix $X^\top X$ is invertible. Therefore, the solution for c_Θ is:

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

This is the *closed-form solution* for minimizing the squared loss.

Why Is This Problem Special?

We can find a closed-form solution for this problem because the model is linear in c_Θ. Predictions $\hat{y}$ depend linearly on c_Θ, simplifying our problem. Such properties allow analytical solutions for parameter estimation.

Unlike this problem, standard feedforward neural networks are non-linear in their parameters. The parameters of a neural network interact in non-linear ways making the problem different from linear regression. Neural networks also necessitate iterative methods, such as gradient descent, to update parameters step by step until convergence.

As a conclusion, this problem allows for a closed-form solution:

$$\hat{c}_\Theta = (X^\top X)^{-1} X^\top y,$$

because the model is linear and the loss function is convex. These properties make it much easier to solve than the non-linear models used in neural networks.

GitHub Link

You can find the code on GitHub at:
https://github.com/AlexGoncalves21/DeepLearning

References

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press.
Mitchell, T. M. (1997). *Machine learning.* McGraw-Hill.