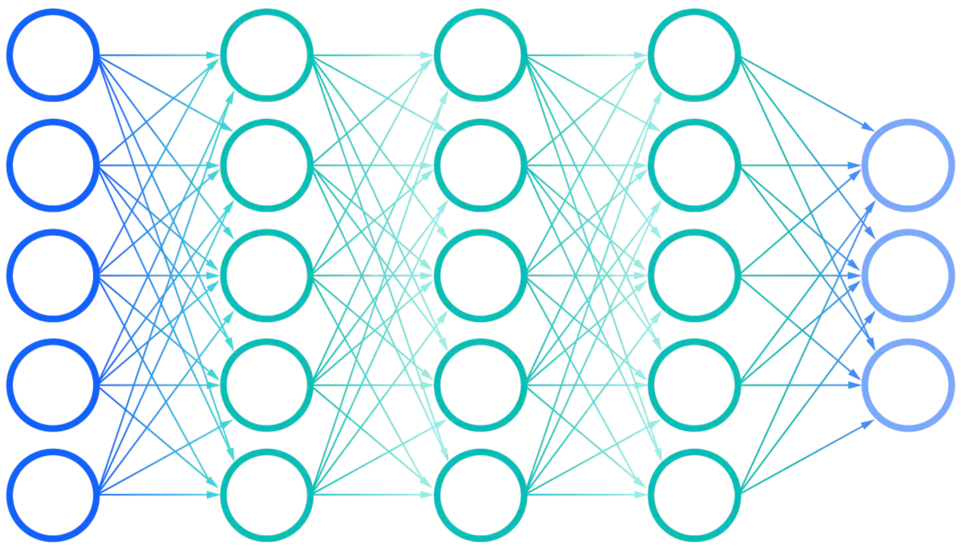
TÉCNICO LISBOA

Homework 1

Final Report



Deep Learning

Instituto Superior Técnico

Beatriz Cavaleiro de Ferreira | 73802

Pedro Zenário | 102348

Ricardo Fraga Simões | 93674

Vicente Sobral | 102134

12 de janeiro de 2022

Index

Introduction

The purpose of this project is to develop four regression and classification tasks, applying theoretical concepts throughout. We will apply a couple of Deep Learning methods to train data in order to obtain a model with the ability to generalize – make accurate predictions when reacting to new (test) data. These exercises allowed us to get in touch with new datasets and to apply the concepts and models learnt in class fully by ourselves.

Question 1

Exercise 1.1

$$\sigma(z) = \frac{1}{1+ e^{-z}}, z \in \mathbb{R}$$

$$\sigma'(z) = \frac{-1}{(1+ e^{-z})^2} \times (-e^{-z}) = \frac{1}{(1+ e^{-z})} \times \frac{e^{-z}}{(1+ e^{-z})} = \frac{1}{(1+ e^{-z})} \times \frac{e^{-z} + (1-1)}{(1+ e^{-z})} =$$

$$= \frac{1}{(1+ e^{-z})} \times \frac{(1+e^{-z})-1}{(1+ e^{-z})} = \frac{1}{(1+ e^{-z})} \times \left(\frac{(1+e^{-z})}{(1+ e^{-z})} + \frac{-1}{(1+ e^{-z})}\right) =$$

$= \frac{1}{(1+e^{-z})} \times \left(1 - \frac{1}{(1+e^{-z})}\right) = \sigma(z) \times (1-\sigma(z))$, as we wanted to show.

Exercise 1.2

With $y = 1$:

$$\mathrm{L}(\mathrm{z};\ \mathrm{y}) = -\frac{1+\mathrm{y}}{2} \times \log \sigma(\mathrm{z}) - \frac{1-\mathrm{y}}{2} \log\big(1 - \sigma(\mathrm{z})\big)$$

$$\mathrm{L}(\mathrm{z};\ \mathrm{y}=1) = -\frac{1+1}{2} \times \log \sigma(\mathrm{z}) - \frac{1-1}{2} \log\big(1 - \sigma(\mathrm{z})\big) = -\log \sigma(\mathrm{z})$$

$$\mathrm{L}'(\mathrm{z};\ \mathrm{y}=1) = \frac{\mathrm{d}}{dz}(-\log \sigma(\mathrm{z})) = -\frac{\sigma'(\mathrm{z})}{\sigma(\mathrm{z})} = -\frac{\sigma(\mathrm{z})\big(1 - \sigma(\mathrm{z})\big)}{\sigma(\mathrm{z})} = \sigma(\mathrm{z}) - 1$$

$$0 < \sigma(\mathrm{z}) < 1 \quad \Rightarrow \quad -1 < \sigma(\mathrm{z}) - 1 < 0$$

$$\mathrm{L}''(\mathrm{z};\ \mathrm{y}=1) = (\sigma(\mathrm{z}) - 1)' = \sigma'(\mathrm{z}) = \sigma(z) \times \big(1 - \sigma(z)\big) > 0$$

Since the second derivative is positive, we can conclude that the binary logistic loss is convex as a function of z

Exercise 1.3

$$J_{Softmax} = \begin{bmatrix} \frac{\partial s_1}{\partial z_1} & \cdots & \frac{\partial s_1}{\partial z_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial s_K}{\partial z_1} & \cdots & \frac{\partial s_K}{\partial z_K} \end{bmatrix}$$

$$s_j = \frac{e^{z_j}}{\sum_{l=1}^{K} e^{z_l}}, \forall j = 1, \ldots, K$$

$$\frac{\partial s_j}{\partial z_k} = \frac{\partial}{\partial z_k}\left(\frac{e^{z_j}}{\sum_{l=1}^{K} e^{z_l}}\right) = \begin{cases} \frac{e^{z_j} \times \left(\sum_{l=1}^{K} e^{z_l}\right) - e^{z_j} \times e^{z_k}}{(\sum_{l=1}^{K} e^{z_l})^2}, j = k \\ -\frac{e^{z_j} \times e^{z_k}}{(\sum_{l=1}^{K} e^{z_l})^2}, j \neq k \end{cases} = \begin{cases} \frac{e^{z_j} \times \left(\sum_{l=1}^{K} e^{z_l} - e^{z_k}\right)}{(\sum_{l=1}^{K} e^{z_l})^2}, j = k \\ \frac{-e^{z_j} \times e^{z_k}}{(\sum_{l=1}^{K} e^{z_l})^2}, j \neq k \end{cases}$$

$$J_{Softmax} = \begin{bmatrix} \frac{e^{z_1} \times \left(\sum_{l=1}^{K} e^{z_l} - e^{z_1}\right)}{(\sum_{l=1}^{K} e^{z_l})^2} & \cdots & -\frac{e^{z_1} \times e^{z_K}}{(\sum_{l=1}^{K} e^{z_l})^2} \\ \vdots & \ddots & \vdots \\ \frac{-e^{z_K} \times e^{z_1}}{(\sum_{l=1}^{K} e^{z_l})^2} & \cdots & \frac{e^{z_K} \times \left(\sum_{l=1}^{K} e^{z_l} - e^{z_K}\right)}{(\sum_{l=1}^{K} e^{z_l})^2} \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{e^{z_1}}{(\sum_{l=1}^{K} e^{z_l})} \times \left(\frac{\sum_{l=1}^{K} e^{z_l}}{(\sum_{l=1}^{K} e^{z_l})} - \frac{e^{z_1}}{(\sum_{l=1}^{K} e^{z_l})}\right) & \cdots & -\frac{e^{z_1}}{(\sum_{l=1}^{K} e^{z_l})} \times \frac{e^{z_K}}{(\sum_{l=1}^{K} e^{z_l})} \\ \vdots & \ddots & \vdots \\ -\frac{e^{z_K}}{(\sum_{l=1}^{K} e^{z_l})} \times \frac{e^{z_1}}{(\sum_{l=1}^{K} e^{z_l})} & \cdots & \frac{e^{z_K}}{(\sum_{l=1}^{K} e^{z_l})} \times \left(\frac{\sum_{l=1}^{K} e^{z_l}}{(\sum_{l=1}^{K} e^{z_l})} - \frac{e^{z_K}}{(\sum_{l=1}^{K} e^{z_l})}\right) \end{bmatrix} =$$

$$= \begin{bmatrix} s_1 \times (1 - s_1) & \cdots & -s_1 \times s_K \\ \vdots & \ddots & \vdots \\ -s_K \times s_1 & \cdots & s_K \times (1 - s_K) \end{bmatrix}$$

Exercise 1.4

Gradient:

$$L(z; y = j) = -\log[softmax(z)]_j \, , L : \mathbb{R}^K \to \mathbb{R},$$

we are only interested in the score z from class $y = j$.

$$\nabla_{L(z;y=j)} = \underset{(k\ =\ 1, \dots, K)}{\frac{\partial}{\partial z_k}} \big(L(z; y = j)\big) = - \begin{bmatrix} \frac{\partial s_j}{\partial z_1} \times \frac{1}{s_j} \\ \cdots \\ \frac{\partial s_j}{\partial z_K} \times \frac{1}{s_j} \end{bmatrix} = - \begin{bmatrix} \frac{-s_1 \times s_j}{s_j} \\ \cdots \\ \frac{s_j \times (1 - s_j)}{s_j} \\ \cdots \\ \frac{-s_K \times s_j}{s_j} \end{bmatrix} = \begin{bmatrix} s_1 \\ \cdots \\ s_j - 1 \\ \cdots \\ s_k \end{bmatrix}$$

Another way to write the gradient is:

$$\nabla_{L(z;y=j)} = \underset{(k\ =\ 1, \dots, K)}{\frac{\partial}{\partial z_k}} \big(L(z; y = j)\big) = \begin{cases} s_j - 1, & if\ k = j \\ s_k\ , & if\ k \neq j \end{cases}$$

The partial derivatives in the gradient can be found on the j-th line from the Jacobian Matrix calculated in the previous question. Computing the Hessian:

$$H_{L(z;y=j)} = \begin{bmatrix} \frac{\partial^2}{\partial {z_1}^2}\left(L(z;y=j)\right) & \frac{\partial^2}{\partial z_1 z_2}\left(L(z;y=j)\right) & \cdots & \frac{\partial^2}{\partial z_1 z_K}\left(L(z;y=j)\right) \\ \frac{\partial^2}{\partial z_2 z_1}\left(L(z;y=j)\right) & \frac{\partial^2}{\partial {z_2}^2}\left(L(z;y=j)\right) & \cdots & \frac{\partial^2}{\partial z_2 z_K}\left(L(z;y=j)\right) \\ \cdots & \cdots & \ddots & \cdots \\ \frac{\partial^2}{\partial z_K z_1}\left(L(z;y=j)\right) & \frac{\partial^2}{\partial z_K z_2}\left(L(z;y=j)\right) & \cdots & \frac{\partial^2}{\partial {z_K}^2}\left(L(z;y=j)\right) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial}{\partial z_1}(s_1) & \frac{\partial}{\partial z_1}(s_2) & \cdots & \frac{\partial}{\partial z_1}(s_j - 1) & \cdots & \frac{\partial}{\partial z_1}(s_k) \\ \frac{\partial}{\partial z_2}(s_1) & \frac{\partial}{\partial z_2}(s_2) & \cdots & \frac{\partial}{\partial z_2}(s_j - 1) & \cdots & \frac{\partial}{\partial z_2}(s_k) \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots \\ \frac{\partial}{\partial z_K}(s_1) & \frac{\partial}{\partial z_K}(s_2) & \cdots & \frac{\partial}{\partial z_K}(s_j) & \cdots & \frac{\partial}{\partial z_K}(s_k) \end{bmatrix} = {J_{Softmax}}^T = J_{Softmax},$$

since the matrix $J_{Softmax}$ is symmetric.

More specifically,

$$H_{L(z;y=j)} = \begin{bmatrix} s_1 \times (1 - s_1) & \cdots & -s_1 \times s_K \\ \vdots & \ddots & \vdots \\ -s_K \times s_1 & \cdots & s_K \times (1 - s_K) \end{bmatrix}$$

To show that the Loss function is convex with respect to z, we need to prove that the previous Hessian matrix is positive semidefinite.

Considering that this matrix is symmetric, to show that it is convex we only need to show its quadratic form is non-negative[1]:

$$\boldsymbol{x^T J_{Softmax} x} = \sum_{j=1}^{K} \sum_{i=1}^{K} x_i\, x_j\, J_{Softmax_{ij}}, \text{ for every vector } \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_K \end{bmatrix}, \in \mathbb{R}^K.$$

In fact,

$$\sum_{j=1}^{K} \sum_{i=1}^{K} x_i x_j J_{Softmax_{ij}} = s_1 s_2 \times (x_1 - x_2)^2 + s_1 s_3 \times (x_1 - x_3)^2 + \cdots + s_1 s_K \times (x_1 - x_K)^2 + s_2 s_3 \times (x_2 - x_3)^2 + \cdots + s_2 s_K \times (x_2 - x_K)^2 + \cdots + s_{K-1} s_K \times (x_{K-1} - x_K)^2 \geq 0, \forall\, \boldsymbol{x} \in \mathbb{R}^K$$

1 https://www.schmidheiny.name/teaching/matrixalgebra2up.pdf

So, we conclude that the Loss function is convex with respect to z.

Exercise 1.5

Using the provided hint, if we define:

$$\boldsymbol{g}\big(\boldsymbol{\phi}(\boldsymbol{x})\big) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}, \text{ and}$$

$$f(z) = -\log[softmax(z)]_j, \text{ then}$$

$$(f \circ \boldsymbol{g})\big(\phi(x)\big) = f(\boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}) = -\log[softmax(\boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b})]_j$$

This composition is a convex function in its parameters $(\boldsymbol{W}, \boldsymbol{b})$ since $\boldsymbol{g}$ is an affine map (linear transformation plus a translation) of $\boldsymbol{W}$ and $\boldsymbol{b}$ and f is convex for every $z \in \mathbb{R}^K$, as it was shown in the previous question.

Now, let's consider a case where function $\boldsymbol{g}$ is a non-linear function of $(\boldsymbol{W}, \boldsymbol{b})$**.** For example, if we consider:

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_1 w_2 & 0 \\ w_1 & w_2 \end{bmatrix} \begin{bmatrix} x \\ x^2 \end{bmatrix},$$

When $x = 1$, we can plot the graph of $-\log[softmax(\boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b})]_j$ as a function parameters $w_1$and w_2, which is clearly non-convex (see the yellow line in next figure).

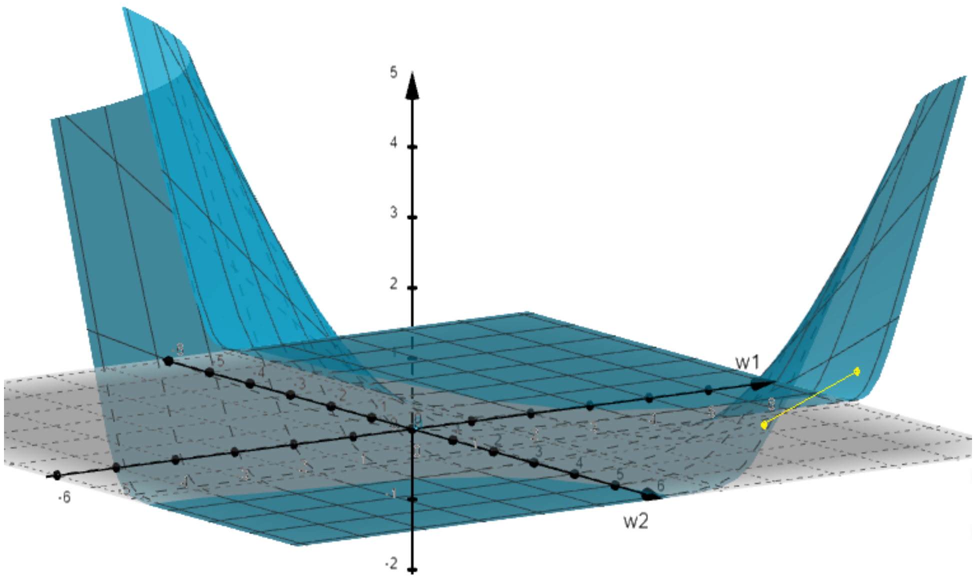


Fig 1

We have shown with this counter example that the composition is not convex in a certain region of its domain, when $\boldsymbol{z}$ is a non-linear function of its parameters.

Question 2

Exercise 2.1

First, we can add a term $\phi_0 = 1$ that captures the bias term, b. In this case $b = w_0$, thus showing that the loss is convex with respect to this extended W is the same as showing that it is convex with respect to (W,b) without this constant feature included.

We have $z = W * \frac{1}{2}\left(z^T z - 2z^T y + y^T y\right)(x)$, with $\phi_0 = 1$ and $b = w_0$.

$$L(z; y) = \frac{1}{2}(z - y)^T * (z - y) = \frac{1}{2}(z^T z - 2z^T y + y^T y)$$

Computing the Gradient,

$$\frac{\partial}{\partial W}\big(L(z;y)\big) = \frac{\partial}{\partial z}\left(\frac{1}{2}\big(z^T z - 2z^T y + y^T y\big)\right) * \frac{\partial z}{\partial W} = \frac{1}{2} * (2 * z - 2 * y) * \frac{\partial z}{\partial W} = (z-y) * \frac{\partial z}{\partial W}$$
$$= (z-y) * \phi^T$$

Computing the Hessian,

$$\frac{\partial^2}{\partial W^2}\big(L(z;y)\big) = \frac{\partial}{\partial W}\big((z-y) * \phi^T\big) = \frac{\partial}{\partial W}(z * \phi^T) = I \otimes (\phi\phi^T),$$

where I is the identity matrix and $\otimes$ is the Kronecker product.

The Hessian is a 4th order entity that can be represented in a matrix form as a $K^2 \times K^2$ matrix, comprised by blocks of symmetric matrices $\emptyset\emptyset^T$ along the "diagonal" multiplied by 1 and zero blocks outside the "diagonal":

$$I \otimes (\emptyset\emptyset^T) = \begin{bmatrix} 1 * \phi\phi^T & \cdots & 0 * \phi\phi^T \\ \vdots & \ddots & \vdots \\ 0 * \phi\phi^T & \cdots & 1 * \phi\phi^T \end{bmatrix} = \begin{bmatrix} \phi\phi^T & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \phi\phi^T \end{bmatrix}$$

If $L(z;y)$ is convex with respect to W, then the Hessian matrix has to be positive semidefinite (PSD).

By definition[2], a symmetric matrix is PSD if all its eigenvalues are non-negative.

Each diagonal block of the Hessian Matrix is constituted by the Matrix $\emptyset\emptyset^T$, which is PSD (every matrix times each transpose is PSD), and therefore it has non-negative eigenvalues.

Since the eigenvalues of a block matrix is constituted by the eigenvalues of each block[3], we can conclude that the eigenvalues of the Hessian are non-negative, and therefore the Hessian is PSD. Ultimately, we conclude that L(z;y) is convex.

2 http://maecourses.ucsd.edu/~mdeolive/mae280a/lecture11.pdf

3 https://math.stackexchange.com/questions/1307998/how-to-find-the-eigenvalues-of-a-block-diagonal-matrix

Exercise 2.2 a)

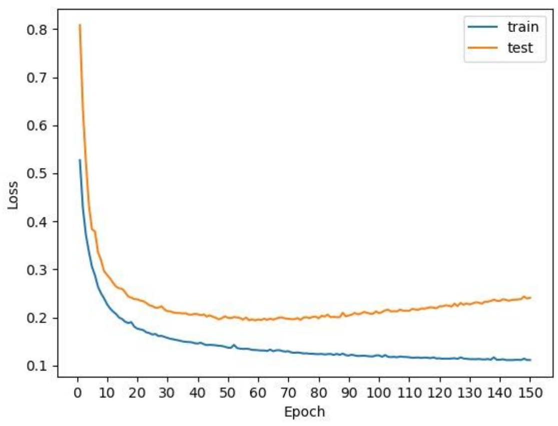


Fig 2

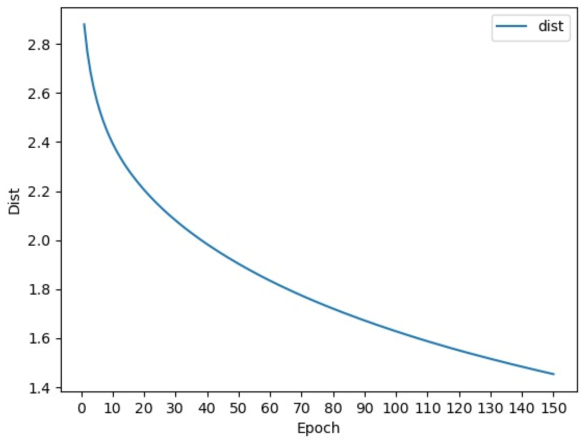


Fig 3

First, we can see on Fig. 2 that the loss on the test and train decrease, and the solutions obtained converge to a global minimum (or high-quality local minimum) around epoch 60. After that particular epoch, the model starts overfitting the training data, and loses its generalization capacity, since the loss on the test increases.

We can also verify that the numerical solution is converging to the analytical solution, since the distance between the weight vectors is decreasing with the number of epochs, as seen in Fig 3. Ultimately, we conclude that the vectors that are close to the analytical solution vector aren't a good solution for this problem, since they are located in the region of Fig. 2 where the model is in overfitting. In sum, if we had to pick this model, we would train it with 60 epochs.

Exercise 2.2 b)

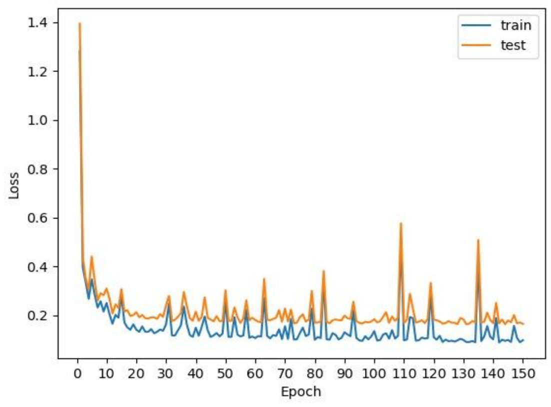


Fig 4

For the Neural Network (see Fig. 4), we can observe that there is convergence to global minima (or a high-quality local minimum), since the loss of model decreases with the number of epochs, until it settles around 0.1 and 0.2 for the train and test, respectively.

Nonetheless, we can observe some “spikes” in the loss function of both the train and test set. This is, most likely, a consequence of using SGD as an optimizer. Since SGD calculates gradients using just one data point, this implies more time until convergence is achieved. One certain epochs, the model probably computed the gradients with certain “noisy”/”hard-to-learn” points, that lead the model to exit the low-loss zones, where it was located.

If we had to pick this model, training it with 150 epochs could be a relevant option, but it would also be worth to compute the gradients using mini-batch, for example. Since we would be computing gradients using more data points, this could help in the convergence of the model and eliminate the “spikes”.

If we compare these results of the Neural Network (NN) with the linear model, as the number of epochs increases, the NN settles around 0.1 and 0.2 for the train and test, respectively. We previously saw that the linear model’s loss decreased in the train (below 0.1), but it increased in the test (more than 0.2), for a larger number of epochs. Nonetheless, both models have similar values of loss in both train and test sets, when they are trained for 60 epochs. For this reason, maybe the linear model could be the optimal solution, since its performance is similar to the NN, but it’s a much simpler model, especially from the computational point of view.

Question 3

Exercise 3.1 a)

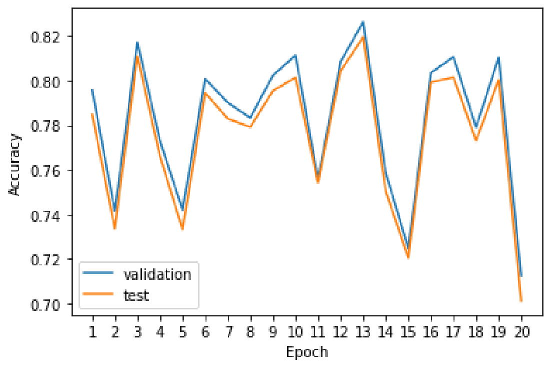


Fig 5

We can observe on Fig. 5 that the perceptron, even though it’s a simple linear model, produced reasonable good results in terms of accuracy. As we can see, for the 20th epoch, it’s always higher than 0.7 and smaller than 0.82.

Nonetheless, we can observe that its accuracy oscillates. Since the perceptron oscillates when the data is not linearly separable, this is most likely the reason why this phenomenon occurred.

Due to this reason, it's hard to pick a certain number of epochs to train the model, so other models (potentially more complex) should be considered.

Exercise 3.1 b)

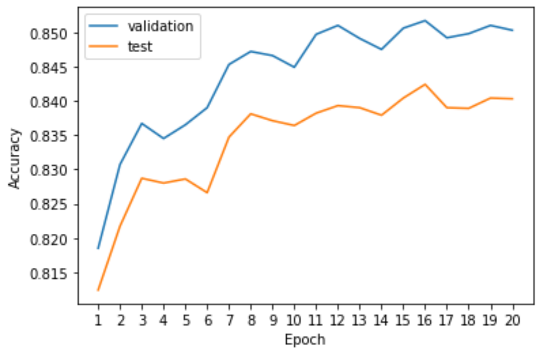


Fig 6

In this case, we are using logistic regression which is, once again, a linear model. Even though it's a simple model, we can observe that it produces better results than the perceptron (see Fig.6).

Here, even though there is some oscillation, this one is much smoother, as the number of epochs grows. Ultimately, any model obtained using 15 to 20 epochs seems a viable option. We can see that the accuracy on the validation is good (0.85) when compared with a model trained with a lower number of epochs.

Exercise 3.2 a)

Multi-layer perceptrons with non-linear activation functions are more expressive than a simple perceptron because of the presence of this non-linearity, which allows the models to explain the target variable in such a way that it varies in a non-linear way, with respect to the explanatory variables. For this classification problem, we allow the model to have a non-linear decision boundary, which is more expressive than a hyperplane (only useful for problems where the data is linearly separable). With this element in the architecture, it is possible to overcome problems where the data is not linearly separable, like the classic "XOR" problem.

Using a linear function as an activation function will not work when approaching non-linearly separable data, since the final output will essentially be a linear combination of the inputs of the network. In fact, no matter how complex the architecture is, if it only contains linear activation functions, then the network is effective only one layer deep.

Exercise 3.2 b)

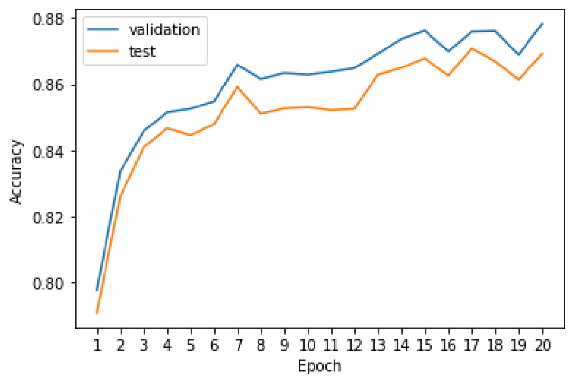


Fig 7

For the MLP, the accuracy results are a little bit more satisfying (see Fig. 7). Just like in the logistic regression model, there are some oscillations in the accuracy of the validation (and test set), but after the model is trained with more than 15 epochs, we can see that the accuracy on the validation is above 0.86 (roughly).

In this model, the accuracy results are just a little bit better when compared with the previous ones. In particular, the model's capacity to produce more complex decision boundaries, due to the non-linarites present in the architecture, could be the reason why the accuracy increased.

For this reason, this model seems to be the ideal pick, among the other models explored in this exercise.

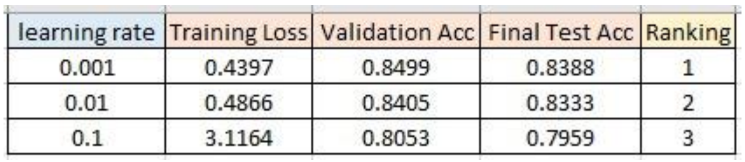
Exercise 4

In order present the results in a more simple, legible and effective way, we will report a table that synthesizes the results obtained for questions 4.1 and 4.2. The detailed accuracy/loss plots of these questions will be included in the appendix.

Exercise 4.1

For this classification problem, we trained a logistic regression model with three different learning rates (see Table 1). Our main conclusion here is the importance of choosing an adequate learning rate. The images for this exercise (see the appendix) show that larger learning rates (0.01 and 0.1) produced oscillations in the validation accuracy and higher training loss values, when compared to the model trained with a smaller learning rate. In fact, the smaller learning rate (0.001) produced a better accuracy on the validation set and lower loss on the training.

Although its accuracy also oscillates, this effect is less severe, and a model with this learning rate and trained over 20 epochs seems to be the best option among these three models.

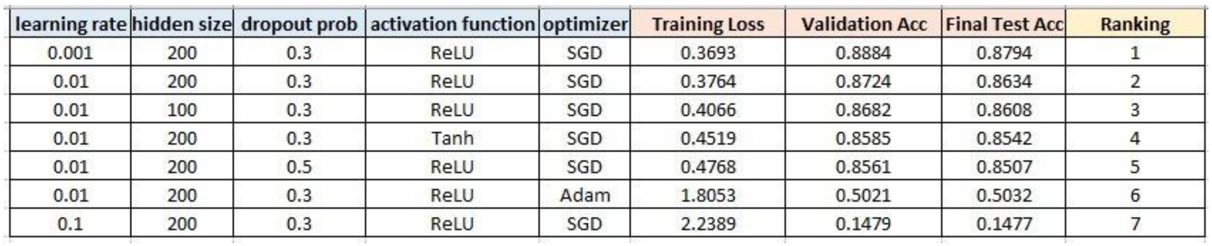


learning rate	Training Loss	Validation Acc	Final Test Acc	Ranking
0.001	0.4397	0.8499	0.8388	1
0.01	0.4866	0.8405	0.8333	2
0.1	3.1164	0.8053	0.7959	3

Table 1

Exercise 4.2

For this exercise, we trained a Neural Network with certain default parameters (second line of Table 2) and then we tuned its hyperparameters (learning rate, hidden size, dropout probability, activation function and optimizer). Every time we changed one of the default values of the five hyperparameters, the others were left in their default configuration. All the configurations that we used to train the model can be found in Table 2. All the plots for these configurations can be found in the appendix.



learning rate	hidden size	dropout prob	activation function	optimizer	Training Loss	Validation Acc	Final Test Acc	Ranking
0.001	200	0.3	ReLU	SGD	0.3693	0.8884	0.8794	1
0.01	200	0.3	ReLU	SGD	0.3764	0.8724	0.8634	2
0.01	100	0.3	ReLU	SGD	0.4066	0.8682	0.8608	3
0.01	200	0.3	Tanh	SGD	0.4519	0.8585	0.8542	4
0.01	200	0.5	ReLU	SGD	0.4768	0.8561	0.8507	5
0.01	200	0.3	ReLU	Adam	1.8053	0.5021	0.5032	6
0.1	200	0.3	ReLU	SGD	2.2389	0.1479	0.1477	7

Table 2

Once again, we can see the importance of picking an adequate learning rate (0.001). The model with the smaller learning rate is the one that produced the best results (lower training loss and higher validation accuracy). On the other hand, a larger learning rate (0.1) produced the worst results. This larger learning rate, most likely, made the model diverge and prevented it from converging to good quality local minima.

Other thing important to notice is that, sometimes, complex architectures are needed to obtain higher quality solutions. In fact, when we reduced the hidden layer size to 100 units, the model lost some of its complexity, which ultimately leads to worse results on both presented metrics.

The same comment applies for the dropout probability. Increasing it from 0.3 to 0.5 probably turned to zero even more elements of the network which, most likely, made the model under-learn some important patterns in the data, which translated in worse performance in both metrics.

It was also interesting to verify that *Adam* – a very popular optimizer – performed much worse than *SGD*. Also, *ReLU* outperformed *Tanh*, although the differences in training loss and validation accuracy aren't significant.

This leads us to the conclusion that hyperparameter tuning is very important task to perform when training a neural network. After this exercise, if we had to pick one model, we would pick the model ranked 1, seen on table 2.

Exercise 4.3

One Layer: **Two Layers:** **Three Layers:**

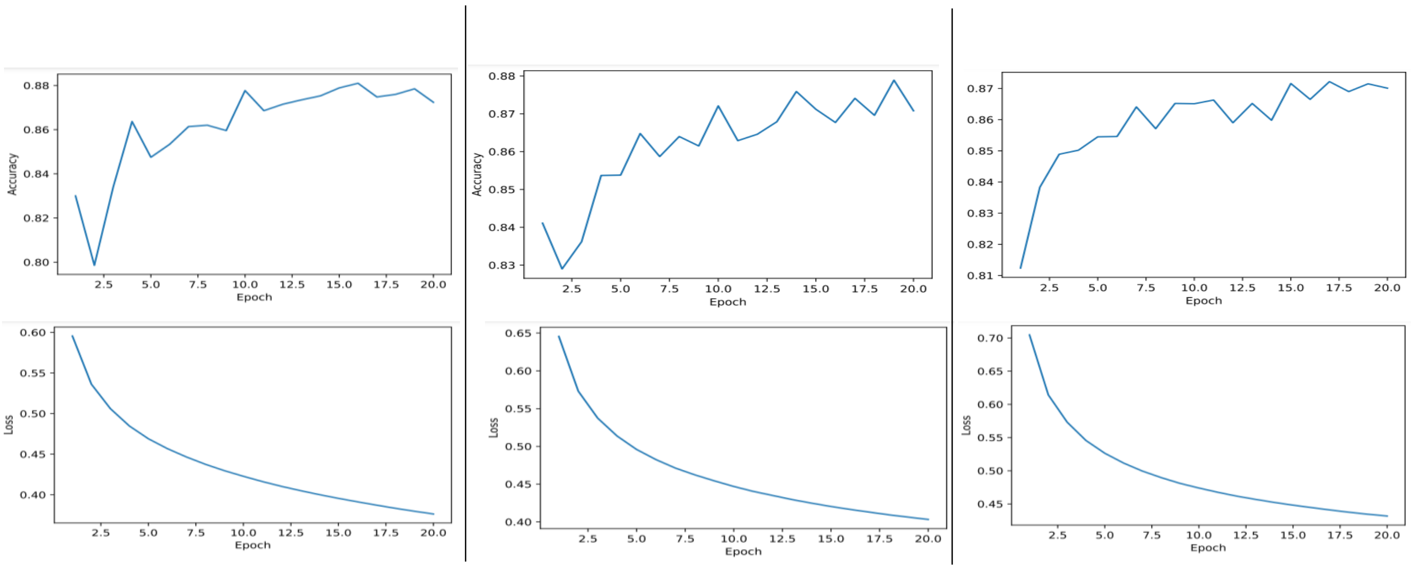


Table 3

On the previous exercise we saw that the complexity of the hidden size was important to produce better results. Here, we can verify that the number of layers (for the same default settings) does not produce better results. We can see that more layers lead to higher training loss and lower validation accuracy (specially the model with three layers). The more complex models, most likely, learned "too well" the data points from the training set, which made them loss some generalization capacity.

Nonetheless, even though it was not required to tune the hyperparameters from the models with more than one layer, it is important to mention that tunning them could lead to better results, so it is worth trying it in future works.

Conclusion

These four regression and classification problems represented great opportunities for us to have a more hands-on approach of the theoretical concepts and models presented in the classes, and allowed us to experiment and go further, understanding better the applications and meaning of the whole training – validation – test process that is a fundamental part of supervised learning.

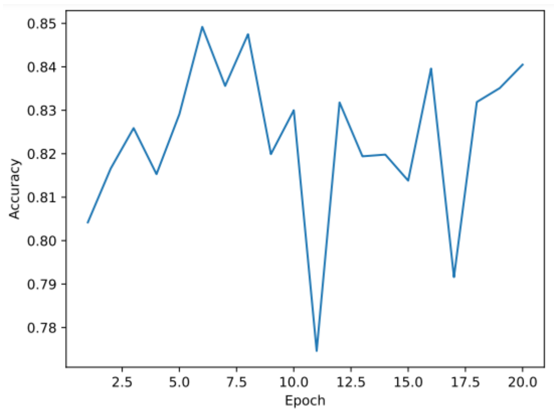
Furthermore, these exercises generated a lot of discussion, because we were left in a situation where we could think by ourselves and come up with distinct suggestions and approaches – always having in mind the final goal of optimizing the error value.

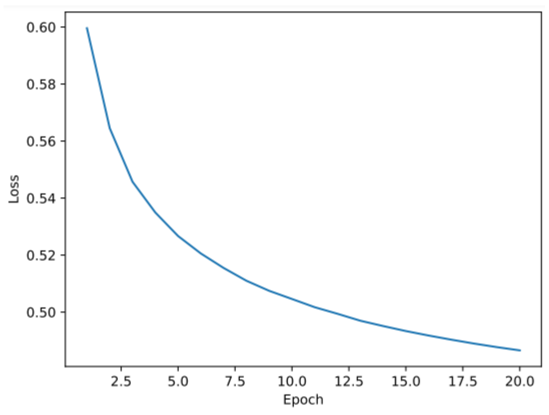
Overall, these exercises were great learning experiences and provided knowledge that we will take for the rest of the trimester and eventually apply in our professional lives.

Appendix

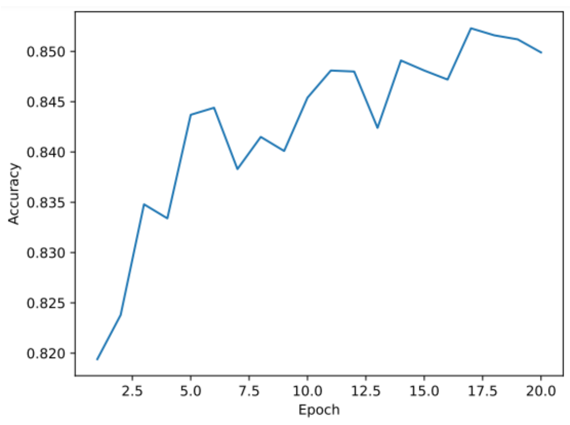
4.1

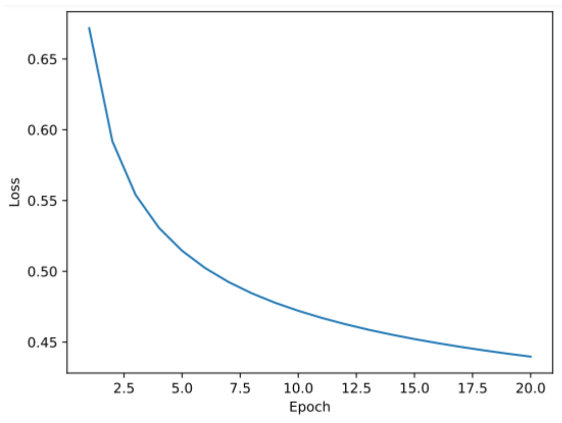
Lr = 0.01:



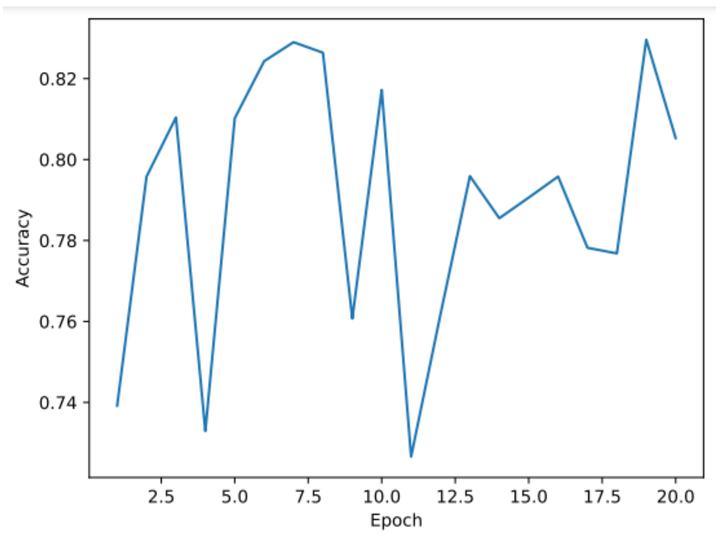


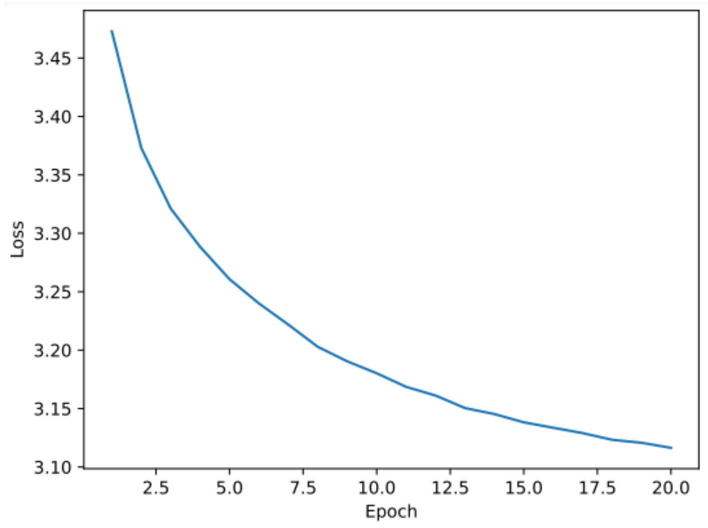
L.r = 0.001:





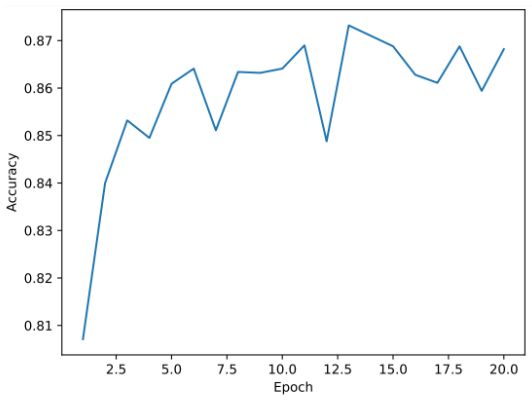
L.r = 0.1:

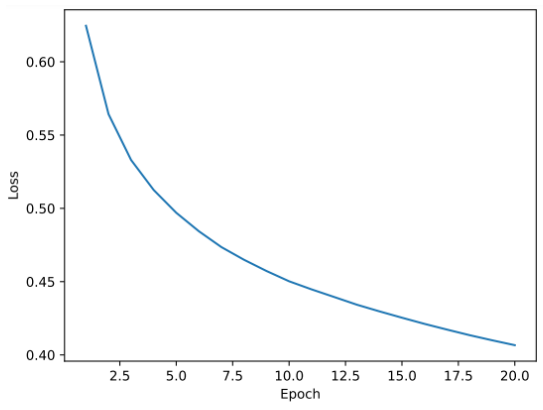




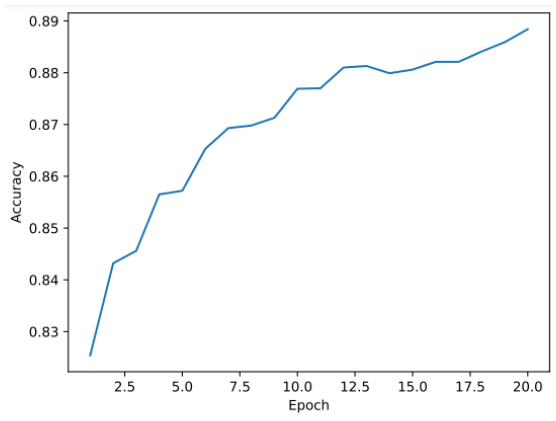
4.2

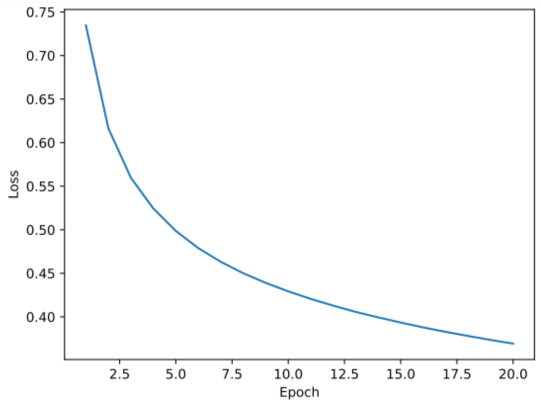
l.r.=0.01, h.s.=100, d.p.=0.3, ReLU, SGD:



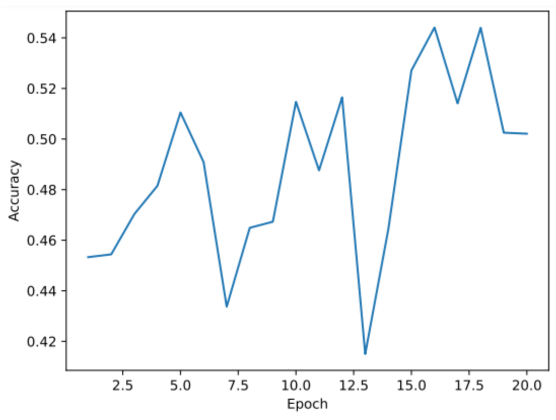


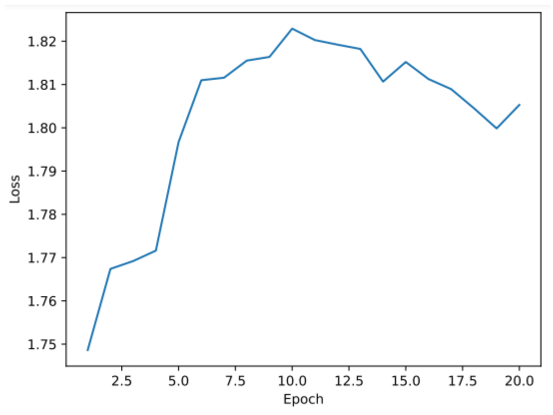
l.r.=0.001, h.s.=200, d.p.=0.3, ReLU, SGD:



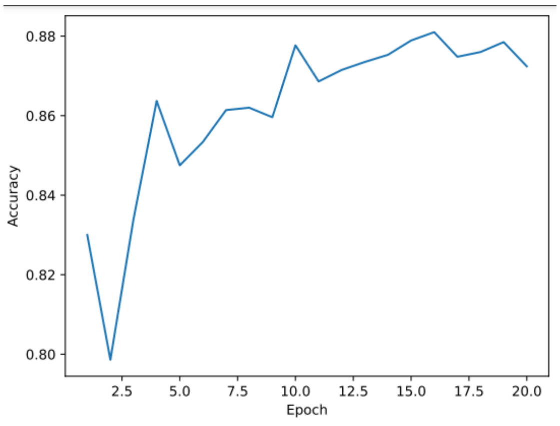


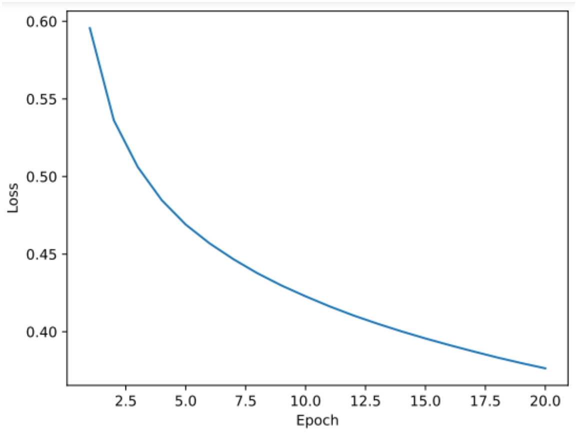
l.r.=0.01, h.s.=200, d.p.=0.3, ReLU, Adam:



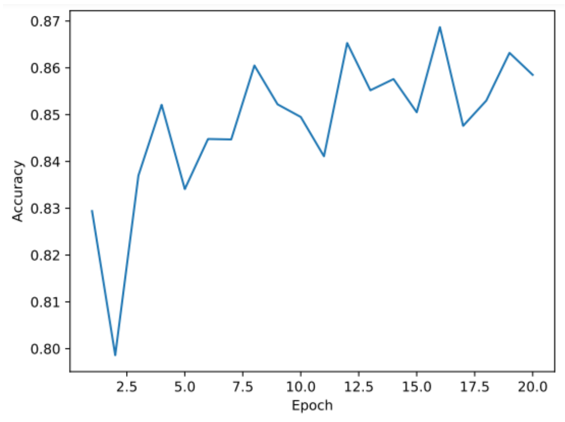


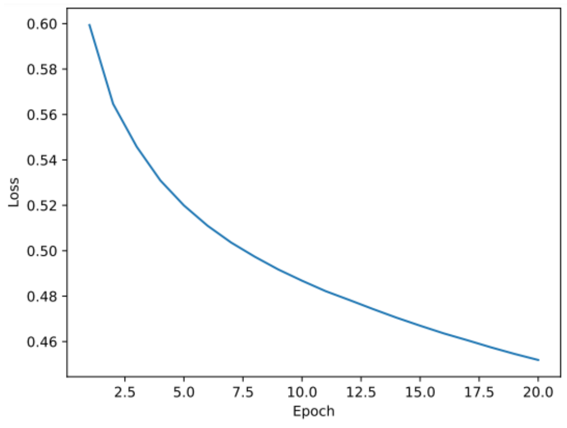
l.r.=0.01, h.s.=200, d.p.=0.3, ReLU, SGD:



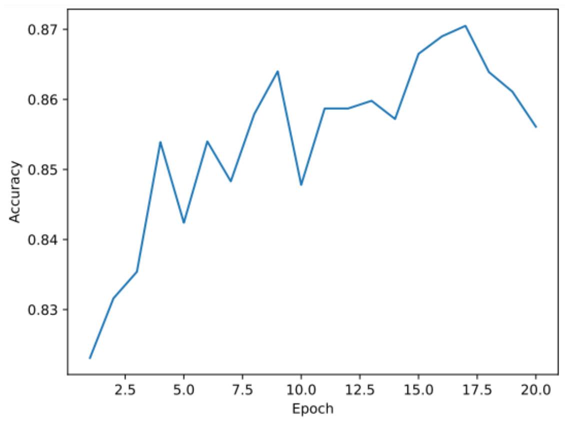


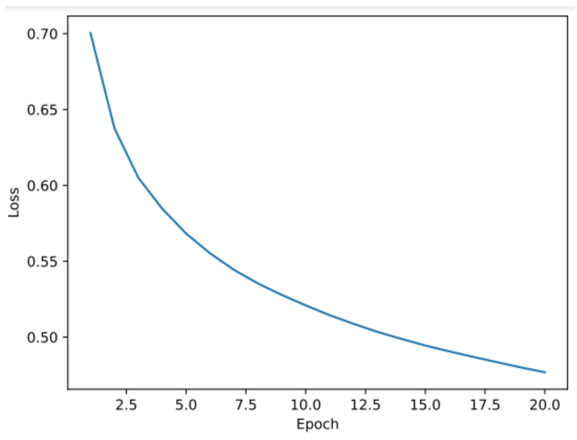
l.r.=0.01, h.s.=200, d.p.=0.3, Tanh, SGD:



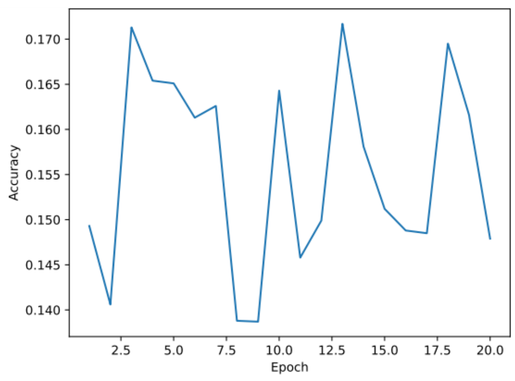
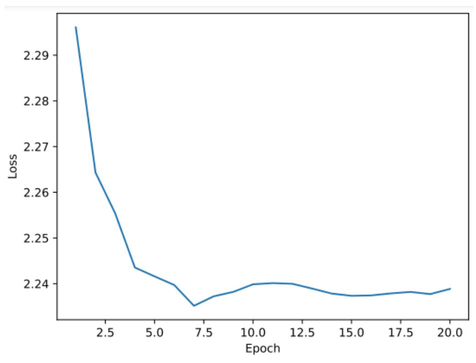


l.r.=0.01, h.s.=200, d.p.=0.5, ReLU, SGD:





l.r.=0.1, h.s.=200, d.p.=0.3, ReLU, SGD:


Accuracy
Epoch
0.170
0.165
0.160
0.155
0.150
0.145
0.140
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Loss
Epoch
2.29
2.28
2.27
2.26
2.25
2.24
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0