# Deep Learning - Homework 1 - Group 32

Alexandre Pires - 92414, Diogo Fouto - 93705, João Fonseca - 92497

January 2022

## 1 Question 1

### 1.1

$$\begin{aligned}
\sigma(z) &= \frac{1}{1+e^{-z}} \\
\sigma'(z) &= \frac{1' \times (1+e^{-z}) - 1 \times (1+e^{-z})'}{(1+e^{-z})^2} = \\
&= \frac{e^{-z}}{1+e^{-z}} \times \frac{1}{1+e^{-z}} = \\
&= \frac{e^{-z}}{1+e^{-z}} \times \sigma(z) = \frac{e^{-z}+1-1}{1+e^{-z}} \times \sigma(z) = \\
&= \left(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right) \times \sigma(z) = \\
&= \sigma(z)(1-\sigma(z))
\end{aligned}$$

### 1.2

In order to see if a function $L$ is strictly convex, we need to show that for whatever $z \in D_L, L''(z) > 0$.

$$\begin{aligned}
L(z; y=+1) &= -log(\sigma(z)) \\
L'(z; y=+1) &= -log(\sigma(z))' = \frac{-\sigma'(z)}{\sigma(z)} = \\
&= -\frac{\sigma(z)(1-\sigma(z))}{\sigma(z)} = \sigma(z) - 1 \\
L''(z; y=+1) &= (\sigma(z)-1)' = \sigma(z)(1-\sigma(z))
\end{aligned}$$

Since $\sigma(z) > 0, \forall z \in \mathbb{R}, L'' > 0$ if and only if $1 - \sigma(z) > 0 \Leftrightarrow \sigma(z) < 1 \Leftrightarrow \frac{1}{1+e^{-z}} < 1 \Leftrightarrow e^{-z} > 0$
As this is always true for any value, one can conclude $L'' > 0, \forall z \in \mathbb{R}$, so L in strictly convex.

### 1.3

$$\begin{aligned}
\frac{\partial[\text{softmax}(z)]_j}{\partial z_k} &= \frac{\frac{\partial}{\partial z_k}(\exp(z_j)) \sum_{i=1}^N [\text{softmax}(z)]_i - \exp(z_j) \frac{\partial}{\partial z_k}(\sum_{i=1}^N [\text{softmax}(z)]_i)}{(\sum_{i=1}^N [\text{softmax}(z)]_i)^2} = \\
&= \frac{\frac{\partial}{\partial z_k}(\exp(z_j)) \sum_{i=1}^N [\text{softmax}(z)]_i - \exp(z_j + z_k)}{(\sum_{i=1}^N [\text{softmax}(z)]_i)^2}
\end{aligned}$$

This means that, for $k = j$:

$$\begin{aligned}
\frac{\partial[\text{softmax}(z)]_j}{\partial z_k} &= \frac{\exp(z_j)(\sum_{i=1}^N [\text{softmax}(z)]_i - \exp(z_j))}{(\sum_{i=1}^N [\text{softmax}(z)]_i)^2} = \\
&= [\text{softmax}(z)]_j (1 - [\text{softmax}(z)]_j)
\end{aligned}$$

And for $k \neq j$:

$$\begin{aligned}
\frac{\partial[\text{softmax}(z)]_j}{\partial z_k} &= -\frac{\exp(z_j)\exp(z_k)}{(\sum_{i=1}^N [\text{softmax}(z)]_i)^2} = \\
&= -[\text{softmax}(z)]_j [\text{softmax}(z)]_k
\end{aligned}$$

## 1.4

From the previous exercise, we know that

$$\frac{\partial}{\partial z_j} L(z; y = j) = -\frac{[\text{softmax}(z)]_j \ (1 - [\text{softmax}(z)]_j)}{[\text{softmax}(z)]_j} = [\text{softmax}(z)]_j - 1$$

Hence, the gradient will be

$$\nabla L(z) = \begin{bmatrix} [\text{softmax}(z)]_1 - 1 & \dots & [\text{softmax}(z)]_N - 1 \end{bmatrix}^T$$

With the knowledge of the previous exercise, we can calculate

$$\frac{\partial}{\partial z_j z_k} L(z; y = j) = \frac{\partial}{\partial z_k}([\text{softmax}(z)]_j - 1) = \begin{cases} [\text{softmax}(z)]_j(1 - [\text{softmax}(z)]_j) & \text{if } j = k \\ -[\text{softmax}(z)]_j[\text{softmax}(z)]_k & \text{otherwise} \end{cases}$$

Therefore, we define the Hessian matrix of $L$ as:

$$H_{jk} = \begin{cases} [\text{softmax}(z)]_j(1 - [\text{softmax}(z)]_j) & \text{if } j = k \\ -[\text{softmax}(z)]_j[\text{softmax}(z)]_k] & \text{otherwise} \end{cases}, \text{ for } j, k \in \{1, \dots, N\}$$

We can see that the diagonal entries of the Hessian are all positive. From that, and with $[\text{softmax}(z)]_i = s_i$, we can show that

$$\begin{aligned} &|s_i(1 - s_i)| \geq \sum_{j \neq i} |-s_i s_j| \Leftrightarrow \\ &\Leftrightarrow s_i(1 - s_i) \geq s_i \sum_{j \neq i} s_j \Leftrightarrow \\ &\Leftrightarrow 1 - s_i \geq \sum_{j=1}^{N} s_j - s_i \Leftrightarrow \\ &\Leftrightarrow 0 \geq 0, \end{aligned}$$

Which means that the matrix is diagonally dominant. Since the matrix is diagonally dominant and has real and positive diagonal entries, it follows that the Hessian is positive semi-definite, proving that our function $L$ is convex.

## 1.5

A function $f$ is convex if:

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y), \quad 0 < \lambda < 1$$

We want to show that the multinomial logistic loss function $g = (L \circ z)(x) = L(z(x))$ is convex.
We first prove $z$ is convex:

$$\begin{aligned} &z(\lambda x + (1 - \lambda)y) = \mathbf{W}\phi(\lambda x + (1 - \lambda)y) + \mathbf{B} \\ &= \mathbf{W}\phi(\lambda x) + \mathbf{W}\phi((1 - \lambda)y) + \mathbf{B} \\ &= \lambda \mathbf{W}\phi(x) + (1 - \lambda)\mathbf{W}\phi(y) + \lambda \mathbf{B} + (1 - \lambda)\mathbf{B} \\ &= \lambda z(x) + (1 - \lambda)z(y) \end{aligned}$$

And now, since we know from 1.4 that $L$ is convex, we prove $g$ is convex:

$$\begin{aligned} &g(\lambda x + (1 - \lambda)y) = L(z(\lambda x + (1 - \lambda)y)) \\ &= L(\lambda z(x) + (1 - \lambda)z(y)) \\ &\leq \lambda L(z(x)) + (1 - \lambda)L(z(y)) \\ &= \lambda g(x) + (1 - \lambda)g(y) \end{aligned}$$

Thus, the multinomial logistic loss function $L(z(x)) = L(\mathbf{W}\phi(x) + \mathbf{B})$ is convex with respect to ($\mathbf{W}$,$\mathbf{B}$), and therefore a local minimum is a global minimum.
Can we say the same for non-linear $z$? No. Let us recall that if $g(u) = Au + c$ and $f$ is convex in $\mathbb{R}^m$, then $(f \circ g)(u) = f(g(u)) = f(Au + c)$ is convex in $\mathbb{R}^m$. The contrapositive (logically equivalent) is that if $(f \circ g)(u)$ is not convex in $\mathbb{R}^m$, then $g(u) \neq Au + c$ (non-linear) or $f$ is not convex in $\mathbb{R}^m$. Therefore if $z$ is non-linear, it is possible that $L(z(x))$ is not convex.

# 2 Question 2

## 2.1

If we define $\mathbf{x} = \mathbf{z} - \mathbf{y}$, we can write the squared error loss function as $L(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{x}$, which is a perspective ($t = \frac{1}{2} > 0$) of the convex function $f(x) = x^T x$ (see [1] for proof of $f$'s convexity) and therefore also convex. Thus, we can conclude that $L(\mathbf{z}; \mathbf{y})$ is convex with respect to $\mathbf{z}$'s parameters: $\mathbf{W}$ and $\mathbf{B}$.

## 2.2

### 2.2.1 a)

The results for our linear regression model can be seen in 1 and 2, for the loss for each epoch and the distance to the analytical solution by each epoch, respectively. In this model, Figure 1 shows a clear improvement of the loss over the training set as the epoch number increases, but this improvement does not translate directly to an improvement over the test set. In fact, by epoch 70, the loss of the test set increases, which can be attributed to over-fitting the training set. Figure 2 reinforces the idea that the model does converge to the analytical solution, as we see the distance between the analytical and the predicted solutions get lower as the number of epochs increases.

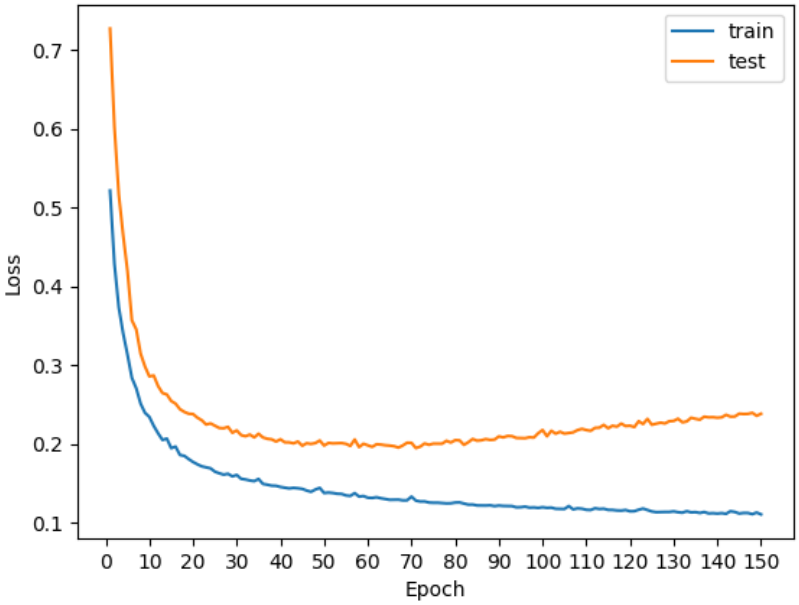


Figure 1: Linear Regression Model - Training and Test set loss in respect to the epoch

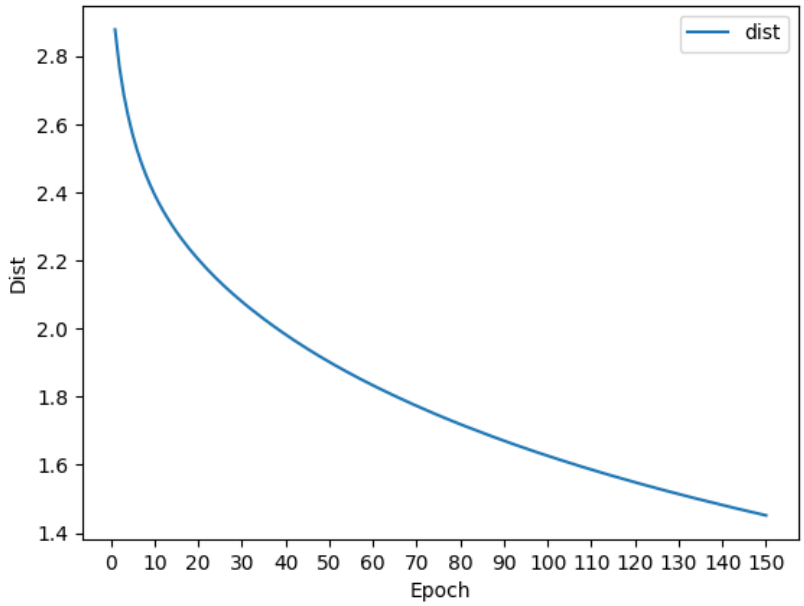


Figure 2: Linear Regression Model - Distance between analytical and predicted solution with respect to the epoch

### 2.2.2 b)

In the neural network model, the results for the loss for each epoch can be seen in Figure 3, and they show that the performance of the training and test set evolve similarly, where if the training performance in an epoch was good or bad, the test set results are also likely to follow the training's performance. However, there still is a difference between the training and test set, which while not drastic, it could also be an indicator of over fitting. Unlike the linear regression model, the results present a lack of stability in training, which could originate from having an inappropriate learning rate, or an unbalanced train/test split.

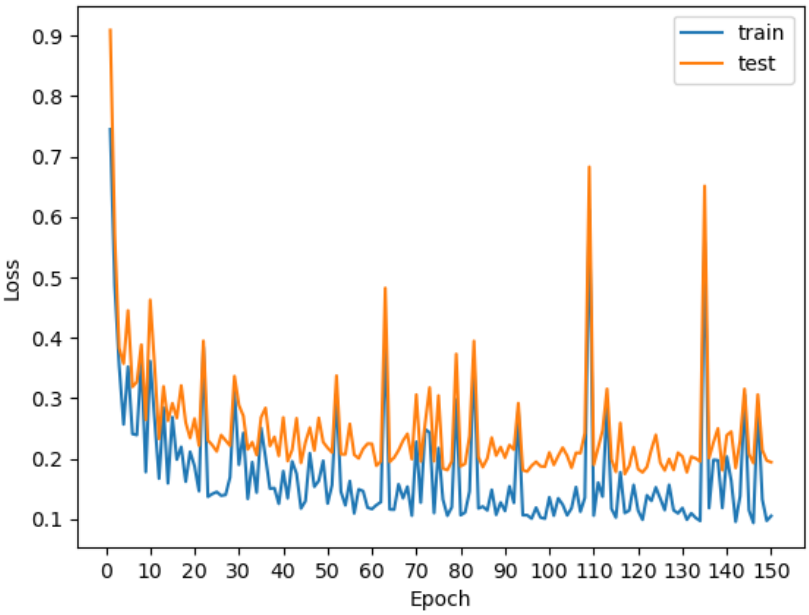


Figure 3: Neural Network Model - Training and Test set loss in respect to epoch

# 3 Question 3

## 3.1

### 3.1.1 a)

In our implementation of the Perceptron model, using the default parameters present in the skeleton code, the validation accuracy fluctuated sharply between 0.72 and 0.82 before falling to 0.70 after 20 epochs, with the test set results following the validation results very closely, although with slightly lower results. The accuracy of the model for each epoch can be seen in 4.

These oscillations could be attributed to a high learning rate, which do not always translate to an increase in accuracy, and the difference between the validation and the test set could be an indicator of over-fitting.

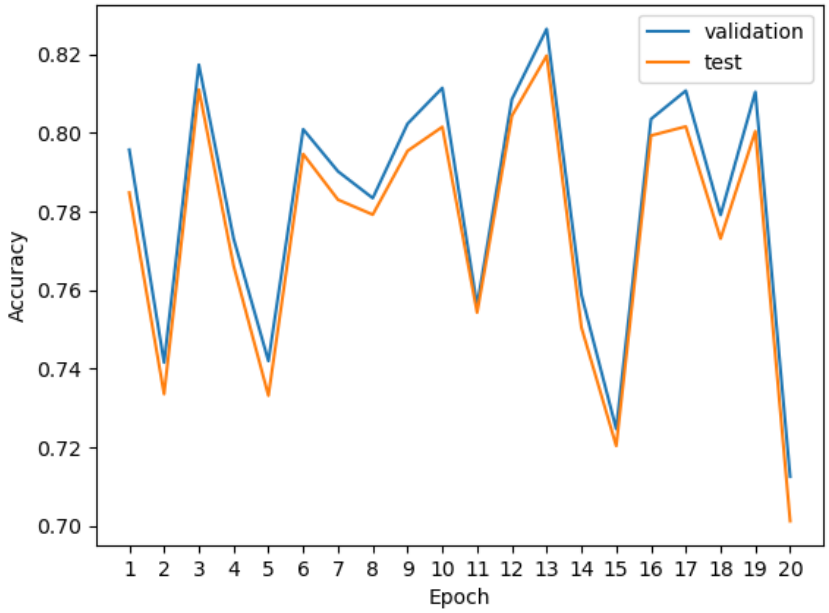


Figure 4: Perceptron - Training and Test set accuracy with respect to each epoch

### 3.1.2 b)

In our implementation of the Logistic Regression model, using the default parameters present in the skeleton code, the validation accuracy oscillated between 0.80 and 0.83, with the test set results showing a slightly worst result, oscillating between 0.78 and 0.82. The accuracy of the model for each epoch can be seen in 5.

Even so, there is a general, albeit slow tendency towards an increase in accuracy. These oscillations could be attributed to a high learning rate, and the difference between the validation and the test set could be an indicator of over-fitting.

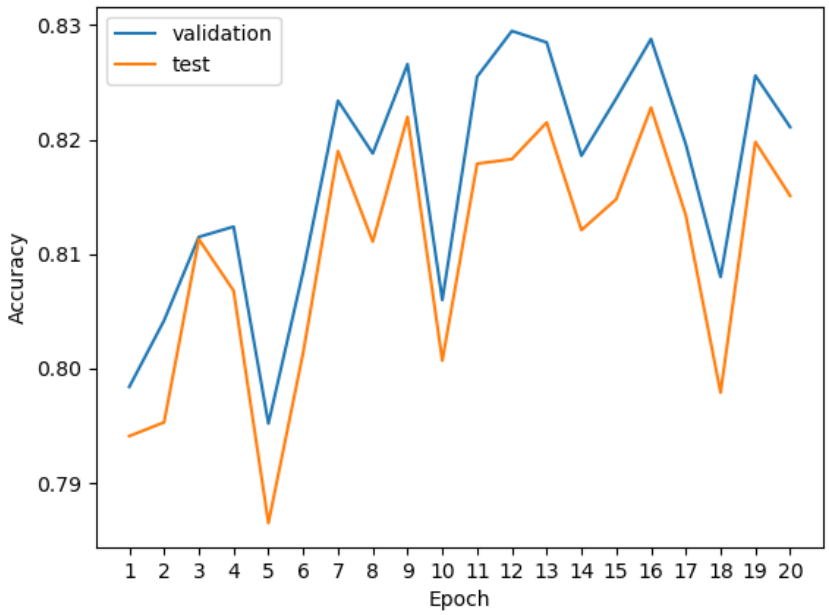


Figure 5: Logistic Regression - Training and Test set accuracy with respect to epoch

## 3.2

### 3.2.1 a)

The simple Perceptron can only solve linearly separable problems, or non-linearly separable problems that were transformed into linearly separable problems.

We can say that a Neural Network (NN) is more expressive than a single Perceptron because it was shown that NNs can approximate arbitrarily well any function. This result was first proven by Hornik et al. who state that a NN with a single hidden layer, with enough hidden units, and with linear output activation can approximate arbitrarily well any function. [2] Later, it was shown that by adding more hidden layers, the number of linear regions a NN can carve out grows exponentially with the number of hidden layers, for a fixed number of hidden units, using the ReLU activation function in each layer. [3] Both of these results assume that not all activation functions of the NN are linear, because NNs that use solely linear activation functions have the same expressive power as a Perceptron.

For this particular task, NNs will learn better internal representations of the different items of clothing, and will use non-linear decision boundaries to differentiate between them.

### 3.2.2 b)

This Multilayer Perceptron model had one hidden layer with 200 hidden units, and 10 output units (corresponding to the 10 different types of clothing in the dataset). The hidden layer used the ReLU as activation function, and the output layer used the softmax. It went through 20 training epochs, using a learning rate of 0.001. The initial weights were sampled from a Normal Distribution ($\mu = 0.1, \sigma^2 = 0.1$). We used Stochastic Gradient Descent without mini-batching to train the network. The chosen loss function was the cross-entropy loss.

As one can see, the network's accuracy had a sharp increase in the first 6 epochs, followed by a stabilization at around 0.86% accuracy, with hints of a general increasing trend nonetheless. Had we trained for more epochs, the accuracy would probably continue increasing until over-fitting started to occur.

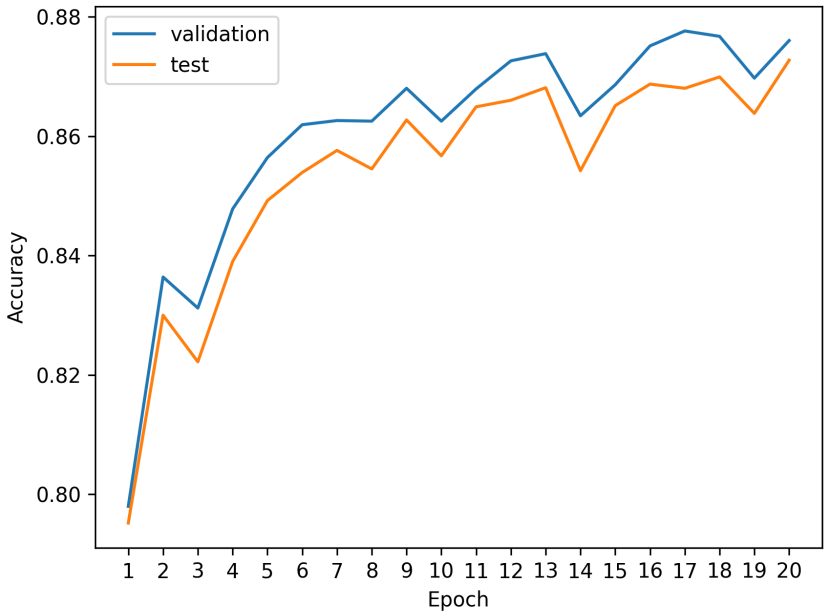


Figure 6: Multilayer Perceptron - Accuracy with respect to each training epoch (Learning Rate: 0.001)

# 4 Question 4

## 4.1

Our logistic regression model was trained three times, with different learning rates: 0.001, 0.01, 0.1. After analysing the results, it was clear that a learning rate of 0.001 presented the best results, which can be seen in Figure 7 and 8, for loss and validation set accuracy respectively. Other learning rates appeared to be too drastic, resulting in a great variation between each epoch, that did not always lead to a better result, so even if the loss was getting lower at each epoch for every learning rate, the validation set accuracy was not stable for a learning rate of either 0.1 and 0.001. In our best configuration, with the learning rate of 0.001, we see that validation set accuracy, while not entirely stable, does tend to increase at each epoch. This configuration reported a final test set accuracy of 0.8415.

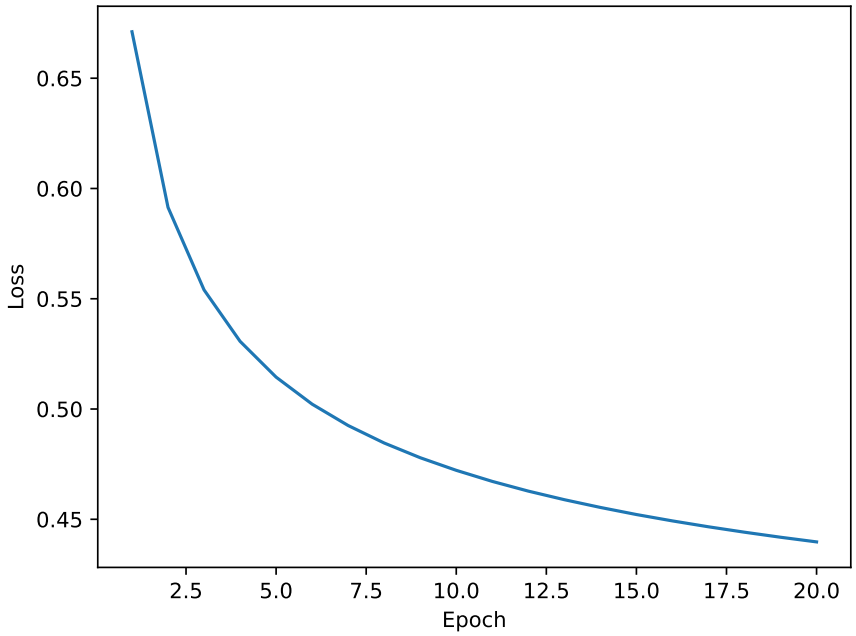


Figure 7: Pytorch Logistic Regression Model - Training loss in respect to the epoch - Learning Rate = 0.001

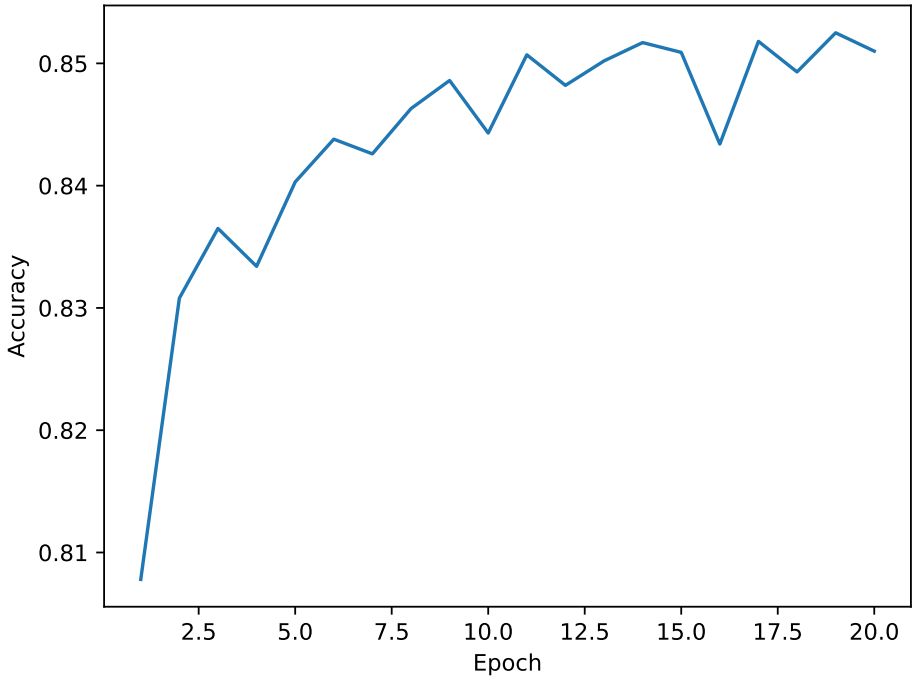


Figure 8: Pytorch Logistic Regression Model - Validation set accuracy with respect to the epoch - Learning Rate = 0.001

## 4.2

Our best Feed-Forward neural network with a single layer had the following parameters: Learning rate of 0.001, hidden-size of 200, drop-out probability of 0.3, ReLU activation function, and used SGD as an optimizer. The results for the loss function and validation accuracy for each epoch can be seen in figure 9 and 10, respectively. The model reported a final accuracy of 0.8776.

While this configuration is what we found to be most optimal, the search space was not exhaustively explored. The assignment asked that only one of the parameters could be changed at once from the default values, so some combinations were left untested. It is possible that many of these combinations had better results than the one produced here.

Here are the main conclusions of our exploration of the different parameters:

- Just like the previous Logistic Regression model, a learning rate of 0.01 or 0.1 proved unhelpful, as it caused too large of a change in the parameters, so the model never really improved.

- Changing the hidden size provided a clear improvement to the model, at the cost of a higher computation time.

- While not substantial, a dropout of 0.5 lead to lower results than with 0.3.

- The tanh activation function also provided slightly lower results than the default values, but with a substantial increase in computing time.

- The Adam optimizer proved considerably worst than using SGD. Not having learned Adam yet, we are unaware of the reason behind this.

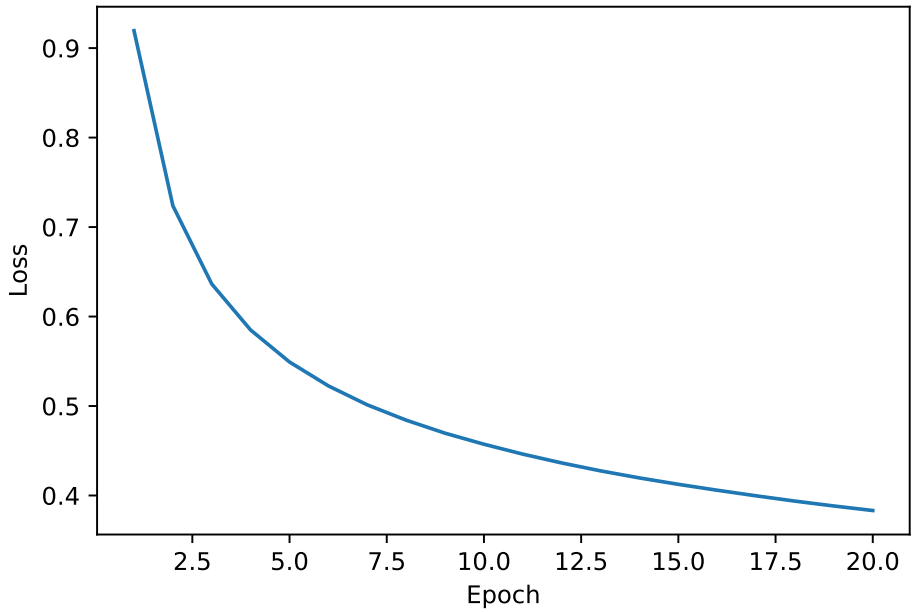


Figure 9: Pytorch Feed-Forward Neural Network Model - 1 Hidden Layer best configuration - Training loss in respect to the epoch

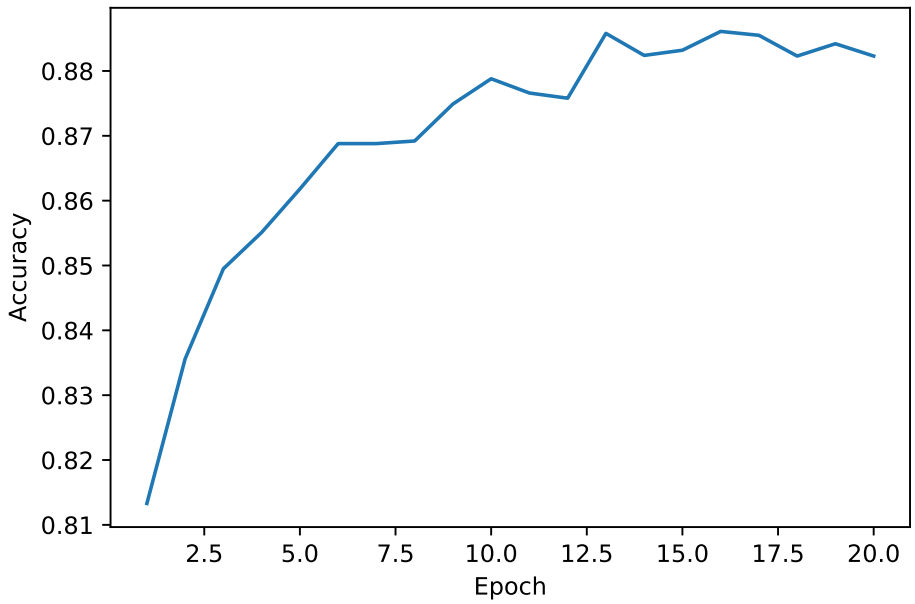


Figure 10: Pytorch Feed-Forward Neural Network Model - 1 Hidden Layer best configuration - Validation set accuracy with respect to the epoch

### 4.3

In both the 2 layer and the 3 layer feed-forward network models, the best configuration was also the one with the default parameters: Learning rate of 0.001, hidden-size of 200, drop-out probability of 0.3, ReLU activation function, and used SGD as an optimizer. Both the 2 layer and the 3 layer model outperformed the 1 layer model, with a final accuracy of 0.8785 and 0.8786, respectively.

The results for the 3 layer mode, which reported a higher accuracy than the 2 layer model, for the loss and validation set accuracy with respect to the epoch, can be found in figures 11 and 12. The model starts off worst than the 1 layer model, which can be due to the greater number of parameters that are yet to be adjusted, yet it quickly increases accuracy.

Finally, the same findings about the changes in parameters in the 1 layer model, referenced in 4.2, also hold true for the 2 and 3 layer models.

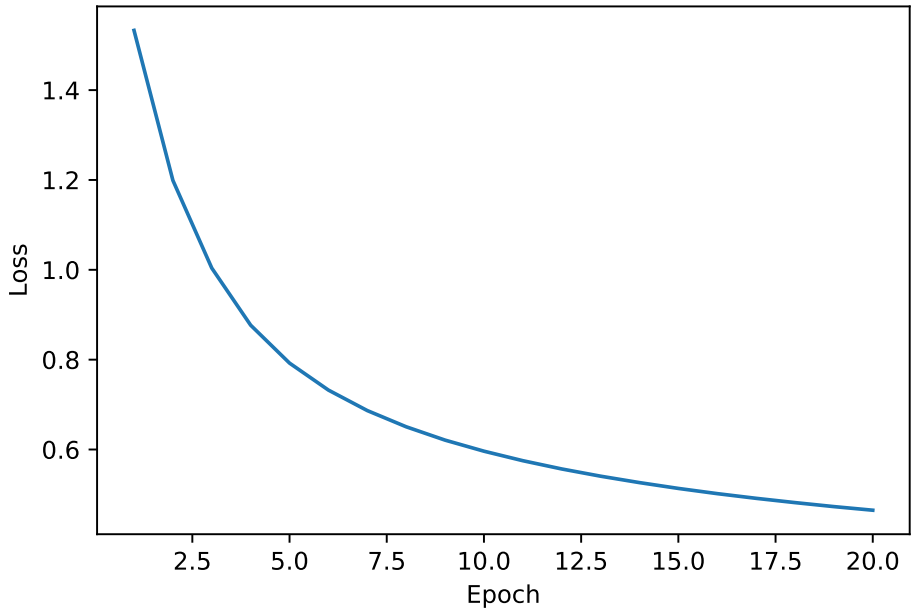


Figure 11: Pytorch Feed-Forward Neural Network Model - 3 Hidden Layers best configuration - Training loss in respect to the epoch

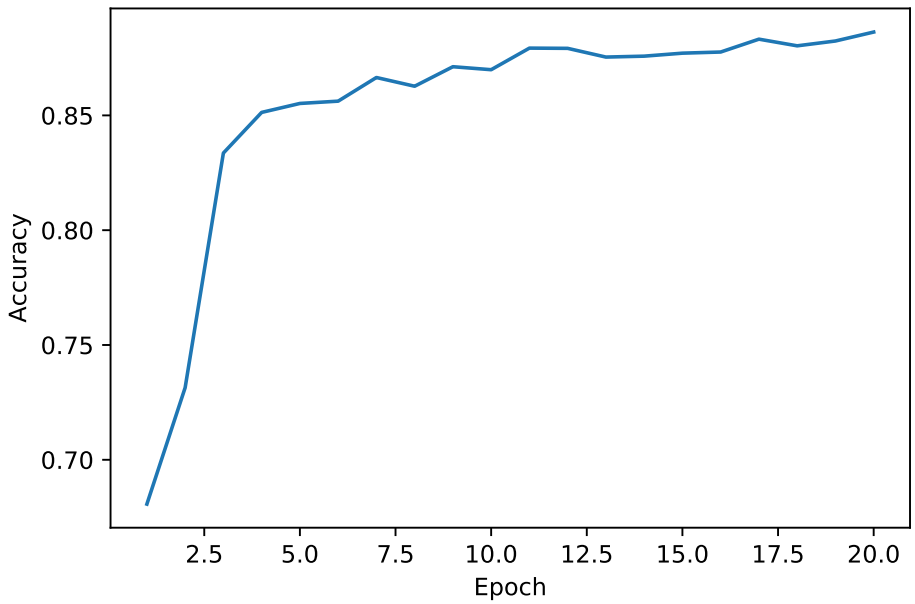


Figure 12: Pytorch Feed-Forward Neural Network Model - 3 Hidden Layers best configuration - Validation set accuracy in respect to the epoch

# References


[1] M. A. (https://math.stackexchange.com/users/22857/martin argerami), "Proof of convexity from definition ($x^t x$)." Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/259426 (version: 2021-01-26).

[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[3] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," *arXiv preprint arXiv:1402.1869*, 2014.