Deep Learning IST, 2021-22

Homework 1

Alexandre Rodrigues - student number: 89404
Luís António Lima da Silva - student number: 102406
Rui Jorge Breda Perdigoto - student number: 102381

Question 1

Exercise 1

Question 1

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

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

$$= -(1+e^{-z})^{2}\left(\frac{d}{dz}1 + \frac{d}{dz}e^{-z}\right) = -(1+e^{-z})^{2}\frac{d}{dz}e^{-z}$$

$$= -(1+e^{-z})^{2} e^{-z} \cdot \frac{d}{dz}(-z) = -(1+e^{-z})^{-2} \cdot \left(e^{-z} \cdot -\frac{d}{dz}(z)\right)$$

$$= -(1+e^{-z})^{-2} \cdot (e^{-z} \cdot -1) = (1+e^{-z})^{-2} \cdot e^{-z}$$

$$= \frac{e^{-z}}{(1+e^{-z}) \cdot (1+e^{-z})} = \frac{e^{-z}+1-1}{(1+e^{-z}) \cdot (1+e^{-z})} = \frac{1}{(1+e^{-z})} \cdot \frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}$$

$$= \underbrace{\frac{1}{1+e^{-z}}}_{\sigma(z)} \cdot 1 - \underbrace{\frac{1}{1+e^{-z}}}_{\sigma(z)} = \sigma(z) \cdot (1-\sigma(z))$$

Exercise 2

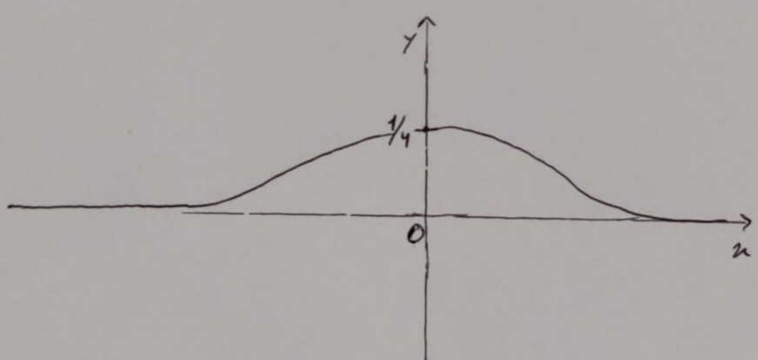
2) $l'(z; y=+1) = (-\log\sigma(z))' = (-\log(\frac{1}{1+e^{-z}}))' = (-(\log(1) - \log(1+e^{-z})))'$

$= (\log(1+e^{-z}))' = \frac{(1+e^{-z})'}{1+e^{-z}} = -\frac{e^{-z}}{1+e^{-z}}$

$l''(z; y=+1) = \left(-\frac{e^{-z}}{1+e^{-z}}\right)' = -\frac{(+e^{-z})' \cdot (1+e^{-z}) - (+e^{-z})(1+e^{-z})'}{(1+e^{-z})^2}$

$= -\frac{(-e^{-z}) \cdot (1+e^{-z}) - (+e^{-z}) \cdot (-e^{-z})}{(1+e^{-z})^2} = -\frac{-e^{-z} - (e^{-z})^2 + (e^{-z})^2}{(1+e^{-z})^2}$

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



This function is convex, because a twice-differentiable function of a single variable is convex if and only its second derivative is nonnegative on its entire domain.

Exercise 3

Question 1.3

$$\text{softmax} : \mathbb{R}^K \to \mathbb{R}^K$$

$$[\text{softmax}(z)]_j = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

Jacobian matrix

$$J_{jk} = \frac{\partial\, [\text{softmax}(z)]_j}{\partial z_k}$$

Two cases: $j = k$ and $j \neq k$

$j = k$:

$$\frac{\partial\, [\text{softmax}(z)]_j}{\partial z_k} = \frac{\partial}{\partial z_k} \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

$$= \frac{\partial}{\partial z_j} \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)} = \frac{\left(\frac{\partial}{\partial z_j} \exp(z_j)\right) \sum_{k=1}^{K} \exp(z_k) - \exp(z_j) \frac{\partial}{\partial z_j} \sum_{k=1}^{K} \dots}{\left(\sum_{k=1}^{K} \exp(z_k)\right)^2}$$

$$= \frac{\exp(z_j) \cdot \sum_{k=1}^{K} \exp(z_k) - \exp(z_j) \cdot \exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)^2} \leftarrow (\text{since } j = 1, \dots, K)$$

$$= \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)} \cdot \frac{\sum_{k=1}^{K} \exp(z_k) - \exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

$$= [\text{softmax}(z)]_j \left(1 - [\text{softmax}(z)]_j\right)$$

When $j \neq k$:

$$\frac{\partial}{\partial z_k}[softmax(z)]_j = \frac{\partial}{\partial z_k} \frac{exp(z_j)}{\sum_{k=1}^{K} exp(z_k)}$$

$$= \frac{\overbrace{\left(\frac{\partial}{\partial z_k} exp(z_j)\right)}^{0} \sum_{k=1}^{K} exp(z_k) - exp(z_j) \cdot \frac{\partial}{\partial z_k} \sum_{k=1}^{K} exp(z_k)}{\left(\sum_{k=1}^{K} exp(z_k)\right)^2}$$

$$= \frac{- exp(z_j) \cdot exp(z_k)}{\left(\sum_{k=1}^{K} exp(z_k)\right)^2}$$

$$= - \frac{exp(z_j)}{\sum_{k=1}^{K} exp(z_k)} \cdot \frac{exp(z_k)}{\sum_{k=1}^{K} exp(z_k)}$$

Note that when "$exp(z_k)$" is within a sum ($\sum_{k=1}^{K} exp(z_k)$), the index k refers to the index of the sum, whereas when "$exp(z_k)$" appears outside the sum (as a result of $\frac{\partial}{\partial z}$), the k refers to the index (j, k) in the jacobian matrix.

$$= -[softmax(z)]_j \cdot [softmax(z)]_k$$

So, the jacobian matrix of the softmax function is defined as follows:

$$J_{(j,k)} = \begin{cases} [softmax(z)]_j \cdot (1 - [softmax(z)]_j) & \text{if } j = k \\ -[softmax(z)]_j \cdot [softmax(z)]_k & \text{if } j \neq k \end{cases}$$

Exercise 4

4. $L(z; y=j) = -\log[\text{softmax}(z)]_j$

$$\frac{\partial}{\partial z_i} -\log[\text{softmax}(z)]_j$$

$$= -\frac{\partial}{\partial z_i} \log[\text{softmax}(z)]_j$$

$$= -\frac{1}{[\text{softmax}(z)]_j} \frac{\partial}{\partial z_i} [\text{softmax}(z)]_j$$

$$= -\frac{1}{[\text{softmax}(z)]_j} \cdot [\text{softmax}(z)]_j \cdot \left(1(i=j) - [\text{softmax}(z)]_i \right)$$

$$= -1(i=j) + [\text{softmax}(z)]_i$$

$$= [\text{softmax}(z)]_i - 1(i=j)$$

where the operation $1(i=j)$ is defined as : $1(i=j) = \begin{cases} 1 & , \text{if } i=j \\ 0 & \text{otherwise} \end{cases}$

The Hessian matrix is the matrix containing all the possible second derivatives.

$$H_{(i,j)} = \begin{cases} \frac{\partial}{\partial z_j} [\text{softmax}(z)]_i - 1 & \text{if } i=j \\ \frac{\partial}{\partial z_j} [\text{softmax}(z)]_i & \text{if } i \neq j \end{cases}$$

$$= \frac{\partial}{\partial z_j} [\text{softmax}(z)]_i \qquad \left(\text{since } \frac{\partial}{\partial z_j}(-1) = 0\right)$$

Notice that this matrix is the same as the Jacobian of the softmax transformation calculated in the previous exercise.

Hessian (loss function) = Jacobian (softmax)

So, the Hessian is

$$H_{(i,j)} = \begin{cases} [\mathrm{softmax}(z)]_i \cdot (1 - \mathrm{softmax}(z)]_i) & , i = j \\ - [\mathrm{softmax}(z)]_i \, [\mathrm{softmax}(z)]_j & , i \neq j \end{cases}$$

~~By definition~~, A twice differentiable function of several variables is convex iff its Hessian is positive semidefinite, i.e., $\forall v \in \mathbb{R}^n$, $v^T H v \geq 0$

For convenience, let's represent H as $H = \mathrm{diag}(\mathrm{softmax}(z)) - \mathrm{softmax}(z) \cdot \mathrm{softmax}(z)^T$

$$v^T H v = v^T \mathrm{diag}(\mathrm{softmax}(z)) v - v^T \mathrm{softmax}(z) \mathrm{softmax}(z)^T v$$

$$= \sum_{i=1}^{K} [\mathrm{softmax}(z)]_i v_i^2 - \left(\sum_{i=1}^{K} v_i [\mathrm{softmax}(z)]_i \right)^2$$

$$= \underbrace{\left(\sum_{i=1}^{K} [\mathrm{softmax}(z)]_i \right)}_{\text{due to the fact that this must sum to } 1.} \left(\sum_{i=1}^{K} [\mathrm{softmax}(z)]_i v_i^2 \right) - \left(\sum_{i=1}^{K} v_i [\mathrm{softmax}(z)]_i \right)^2$$

Based on the Cauchy-Schwartz inequality, we have that

$$\left(\sum_{i=1}^{K} [\mathrm{softmax}(z)]_i \right) \left(\sum_{i=1}^{K} [\mathrm{softmax}(z)]_i v_i^2 \right) - \left(\sum_{i=1}^{K} v_i \cdot [\mathrm{softmax}(z)]_i \right)^2 \geq 0$$

So, the Hessian of the multinomial logistic loss function is positive semidefinite, implying that the loss function is convex with respect to z.

Exercise 5

Question 1.5

As it was proved in the previous exercise, the multinomial logistic loss function is convex with respect to z. Since convexity of a function is invariant under the composition with an affine map, then $L(z; y=j) \circ z(w)$ is also convex w.r.t. the model parameters (w, b).

The convexity of a composition of functions is preserved under the following circumstances:

- if f and g are convex and g is non-decreasing over an univariate domain, then $h(x) = g(f(x))$ is convex.
- if f is concave and g is convex and non-decreasing over a univariate domain, then $h(x) = g(f(x))$ is convex.
- convexity is invariant under affine maps.

So, the logistic loss function can still be convex under specific circumstances when z is not a linear function of the model parameters.

Question 2

Exercise 1

Question 2.1

$$L(z;y) = \frac{1}{2}\|z-y\|_2^2 = \frac{1}{2}(z-y)^T(z-y)$$

$$= \frac{1}{2}\sum_{i=1}^{K}(z_i-y_i)^2 \quad , \quad |z| = K = |y|$$

Consider the ~~map~~ function $f : \mathbb{R}^K \to \mathbb{R}$ $\quad f(z) = \frac{1}{2}\sum_{i=1}^{K}(z_i-y_i)^2$

As an auxiliary result, we will first prove that f is convex w.r.t. z

$$\nabla f(z) = \begin{bmatrix} \frac{\partial f}{\partial z_1}(z) \\ \vdots \\ \frac{\partial f}{\partial z_K}(z) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial z_1}\frac{1}{2}\sum_{i=1}^{K}(z_i-y_i)^2 \\ \vdots \\ \frac{\partial}{\partial z_K}\frac{1}{2}\sum_{i=1}^{K}(z_i-y_i)^2 \end{bmatrix}$$

$$\frac{z_1^2 - 2z_1y_1 + y_1^2 + \dots + z_K^2 - 2z_Ky_K + y_K^2}{2}$$

$$\frac{\partial}{\partial z_i} = 2z_i - 2$$

$$= \begin{bmatrix} z_1 - 1 \\ \vdots \\ z_K - 1 \end{bmatrix}$$

In order to prove that f is convex, we must prove that it's hessian is positive semidefinite

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial z_1 \partial z_1} & \frac{\partial^2 f}{\partial z_1 \partial z_2} & \cdots & \frac{\partial^2 f}{\partial z_1 \partial z_K} \\ \frac{\partial f}{\partial z_2 \partial z_1} & \ddots & & \vdots \\ \vdots & & & \\ \frac{\partial f}{\partial z_K \partial z_1} & \cdots & & \frac{\partial^2 f}{\partial z_K^2} \end{bmatrix}$$

$$\frac{\partial f}{\partial z_i \partial z_j} = \frac{\partial}{\partial z_i}\frac{\partial}{\partial z_j}\frac{1}{2}\sum_{k=1}^{K}(z_k - y_k)^2$$

if $i = j$,

$$= \frac{\partial}{\partial z_i}\frac{1}{2}(2z_i - 2)$$

$$= \frac{\partial}{\partial z_i}(z_i - 1) = 1$$

$$\frac{\partial f}{\partial z_i \partial z_j} = \frac{\partial}{\partial z_i}\frac{1}{2}(2z_j - 2)$$

if $i \neq j$,

$$\frac{\partial}{\partial z_i}(z_j - 1) = 0$$

$$H(f) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

So, the Hessian matrix of f is the $K \times K$ identity matrix.

__Definition__. The real symmetric $N \times N$ matrix V is said to be positive semidefinite

if $$a^T V a \geq 0$$

for any real $N \times 1$ vector a.

$$\text{So,} \quad a^T H(f)\, a = a^T I a$$
$$= a^T a$$
$$= \sum_{i=1}^{K} a_i^2 \geq 0$$

So, the hessian matrix of f is positive semidefinite, hence f is convex.

Since the composition of a affine map $g: \mathbb{R}^n \to \mathbb{R}^m$ with a convex function $f: \mathbb{R}^m \to \mathbb{R}$ is convex, i.e., $(f \circ g)(u)$ is convex,

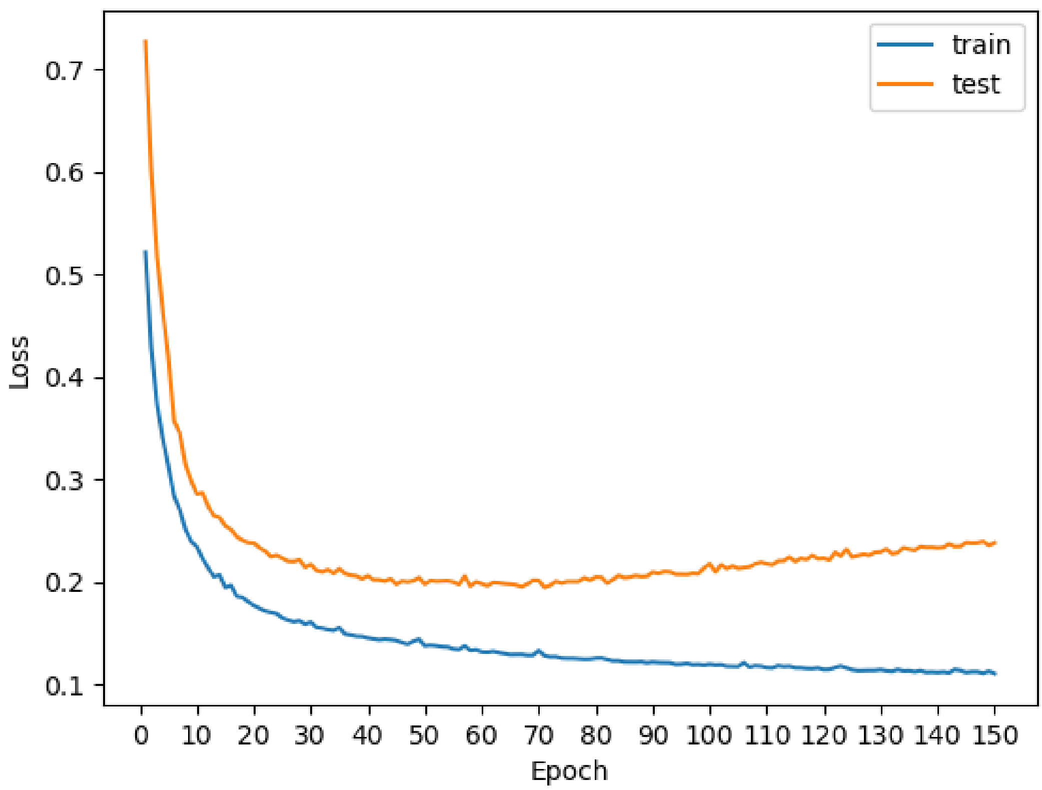
then the composition of $f = \frac{1}{2}\sum_{i=1}^{K}(\kappa_i - y_i)^2$ and $\kappa_i(w_i) = w_i\, \phi(x) + b$

$$(f \circ \cancel{g}\, \kappa_i)(w_i) = \frac{1}{2}\sum_{i=1}^{K}\left(w_i\, \phi(x) + b - y_i\right)^2$$ is convex with

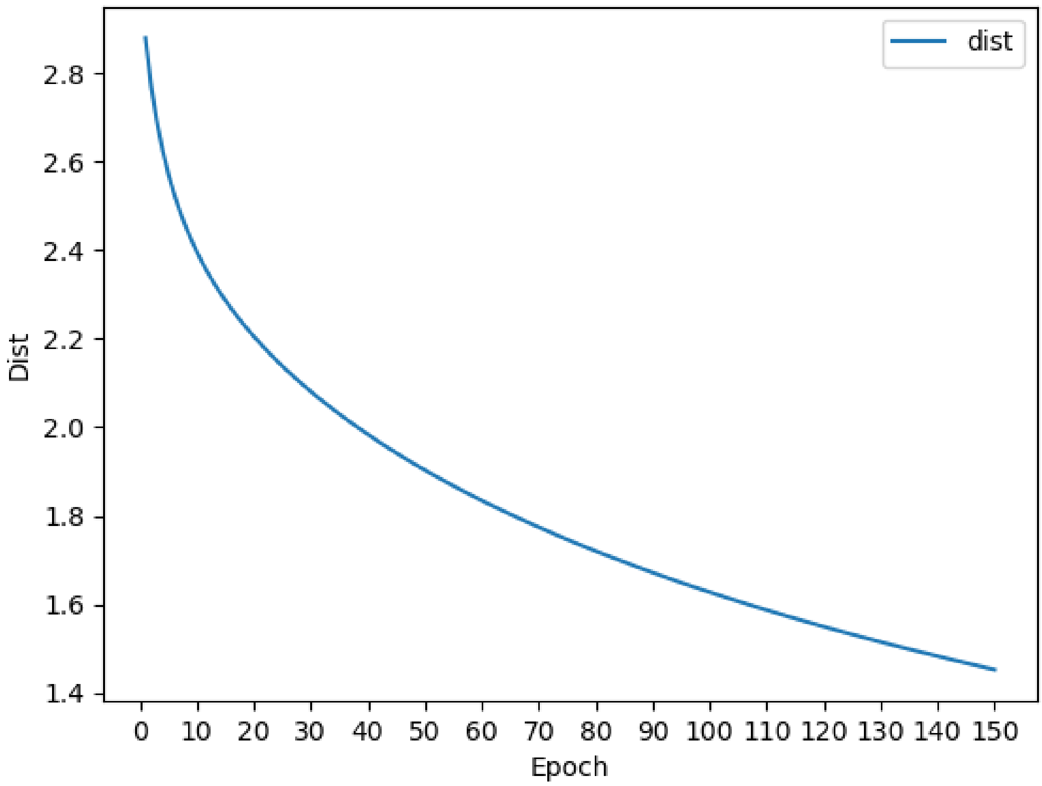
regards to w and b.

Exercise 2.a

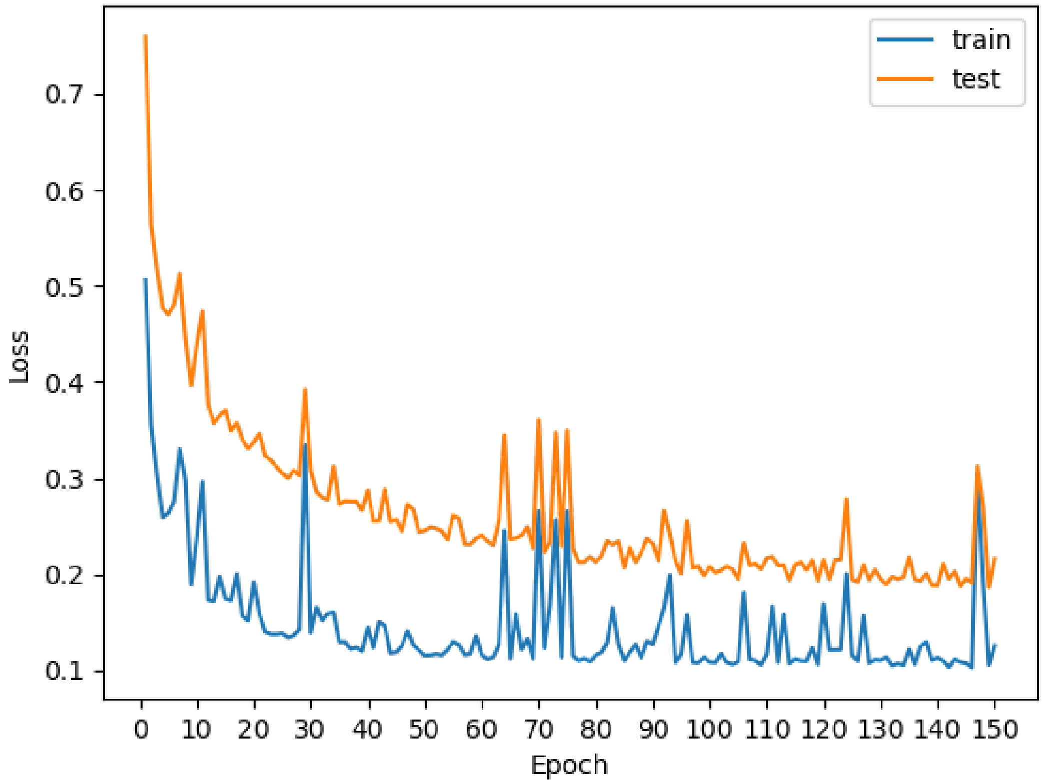
As the number of epochs increases the difference between the train and test sets' loss grows bigger. In the beginning both sets see a diminishing loss but at a certain point the train set's loss continues decreasing while the test set's loss increases. This is when our model starts to overfit. Overfitting is the reason why performance on the training and test sets differs as the number of epochs increases.



As we can see, the distance between our weight vector and the weight vector computed analytically decreases as the epochs increase.



Exercise 2.b

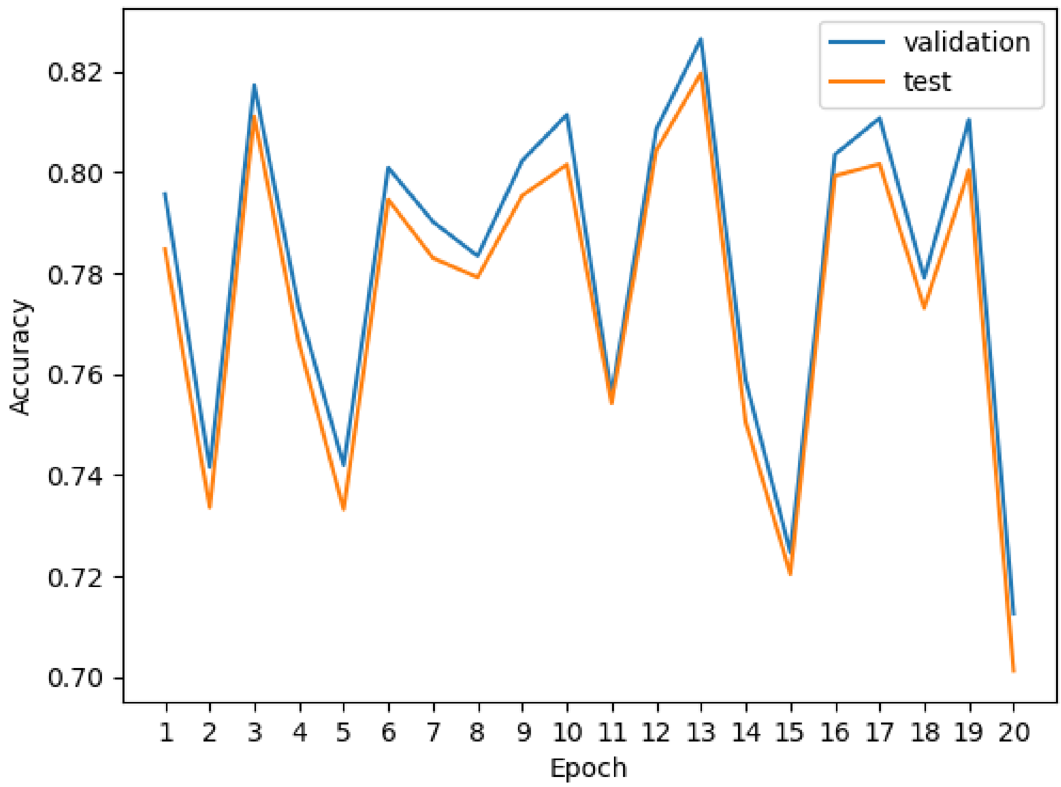


In comparison to the linear model, the observed difference between the performance in the training and test sets is that with a neural network we don't detect overfitting.

Question 3

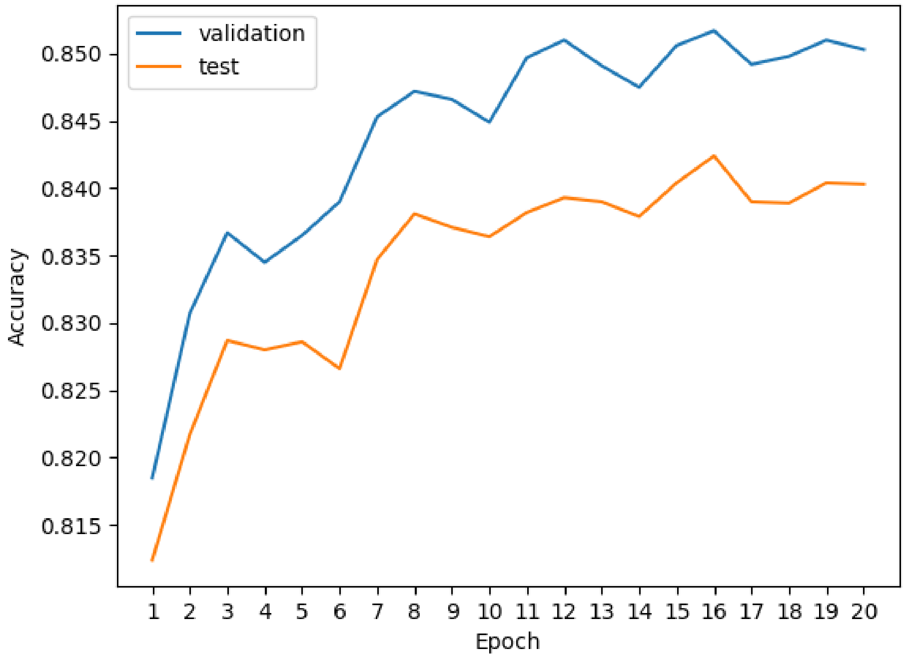
Exercise 1.a

Train with 20 epochs



Exercise 1.b

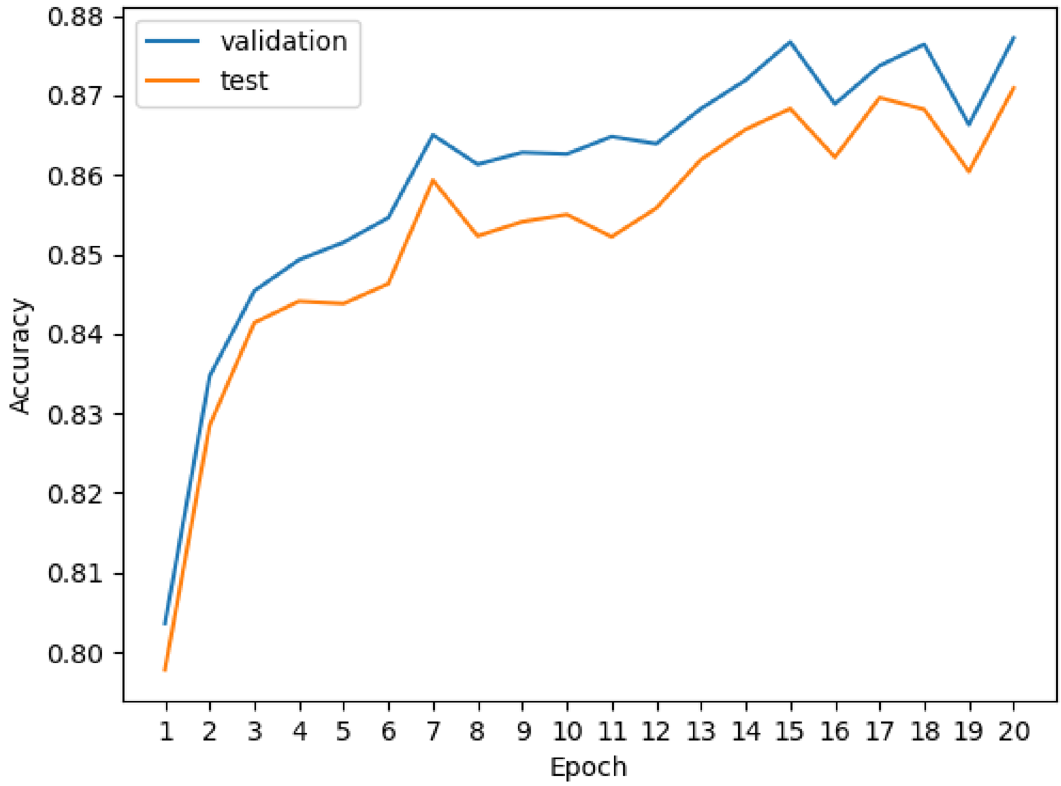
Train with 20 epochs



Exercise 2.a

The simple perceptron is a threshold-based feed-forward network that can only classify linearly separable cases with a binary target. Does not include hidden layers that allow the neural network to create a feature hierarchy.
Multi-layer perceptrons with non-linear activations are more expressive than the simple perceptron because they allow the model to create complex connections between the inputs of the Neural Network and outputs. They can solve the problems of linear activation functions because they allow backpropagation because they have a derived function that is related to the inputs and allow making a kind of "stack" of several layers of neurons constituted by hidden layers that are essential for learning complex datasets with high levels of accuracy. The limitation is that a Multi-layer perceptron with non-linear activations is different from any structure with one layer if the activation function is the same.

Exercise 2.b



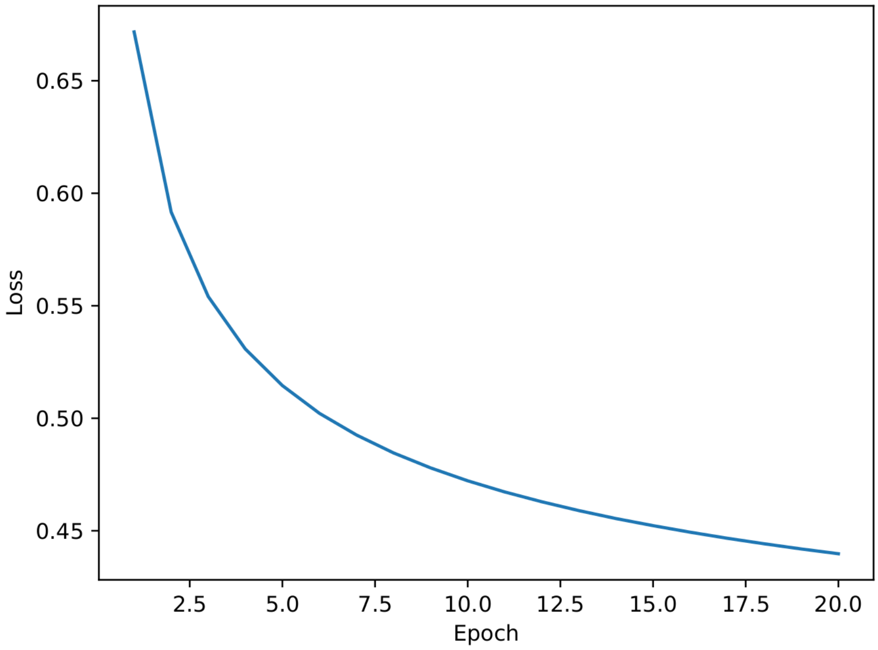
Question 4

Exercise 1

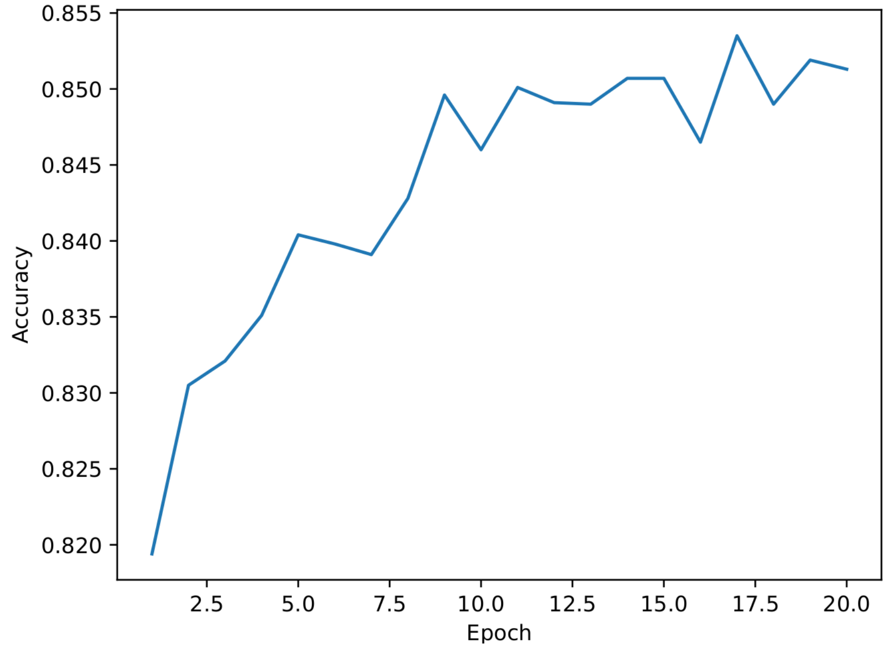
This exercise consisted of building a logistic regression model using pytorch and training it using the MNIST dataset with 20 epochs. Three different learning rates were used {0.001, 0.01 and 0.1} and their respective accuracies and losses were compared. As it can be observed, the configuration which had a better performance was the one using a learning rate of 0.001.

The final test accuracy with lr= 0.001: 0.8402

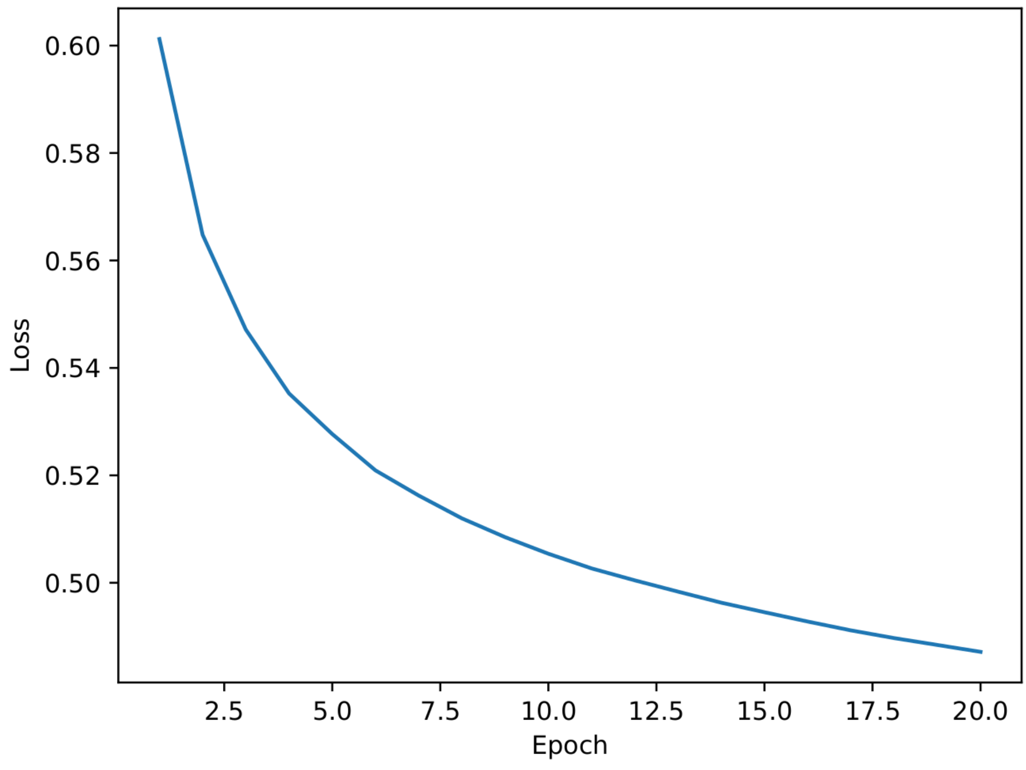
Training loss with lr = 0.001:



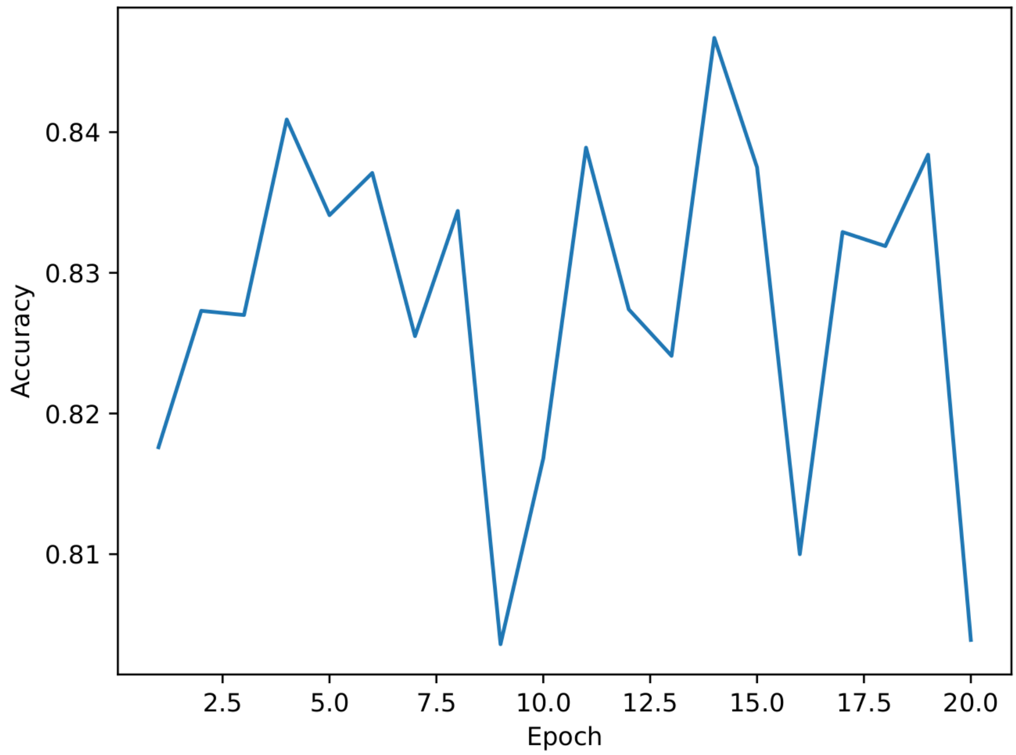
Validation accuracy with lr = 0.001:



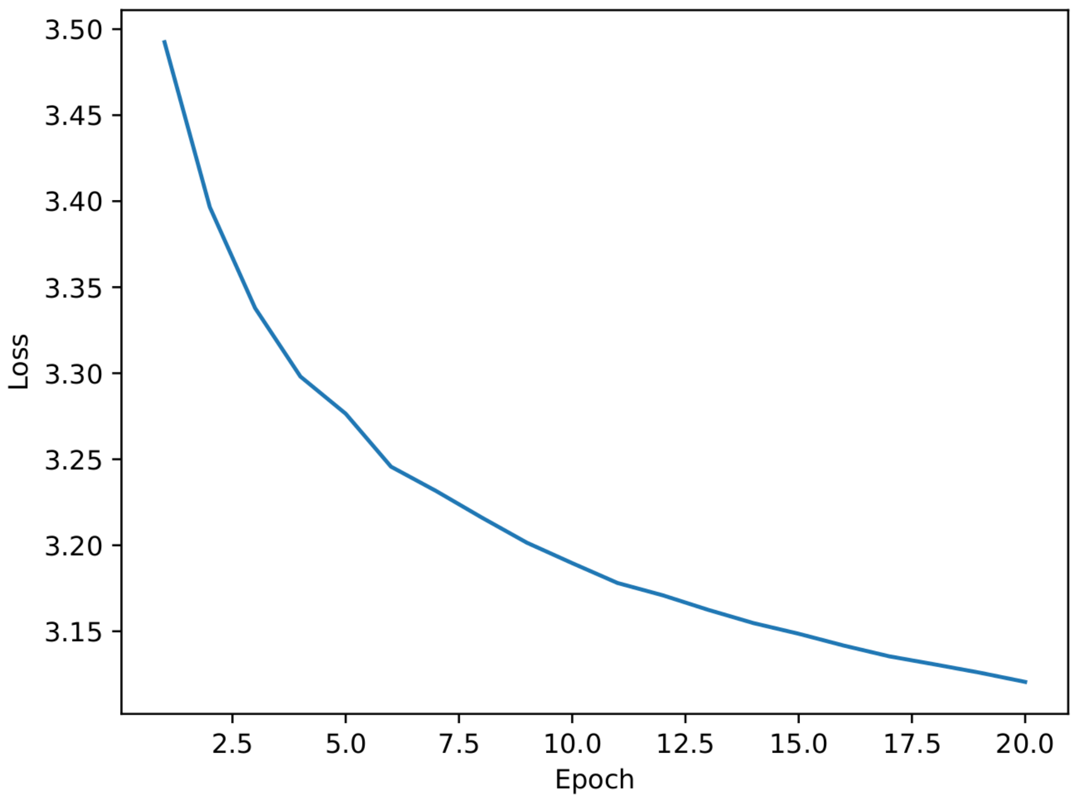
The final test accuracy with lr = 0.01: 0.7944
Training loss with lr = 0.01:



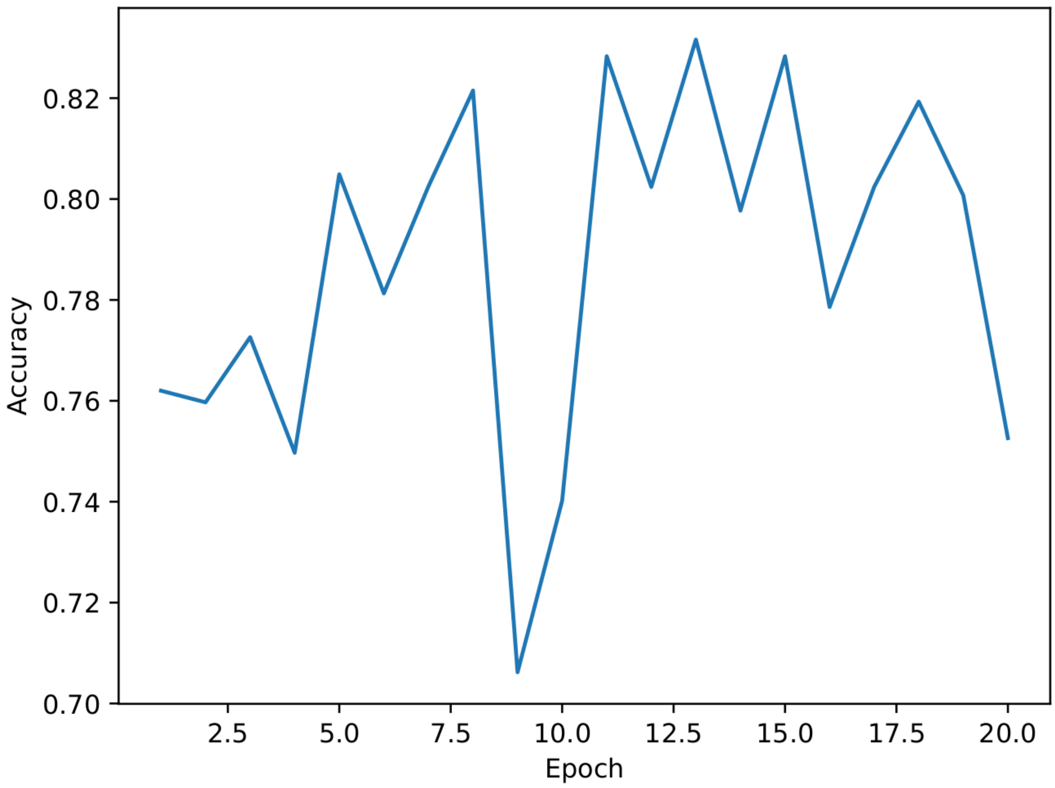
Validation accuracy with lr = 0.01:



The final test accuracy with lr = 0.1: 0.7500
Training loss with lr = 0.1:



Validation accuracy with lr = 0.1:

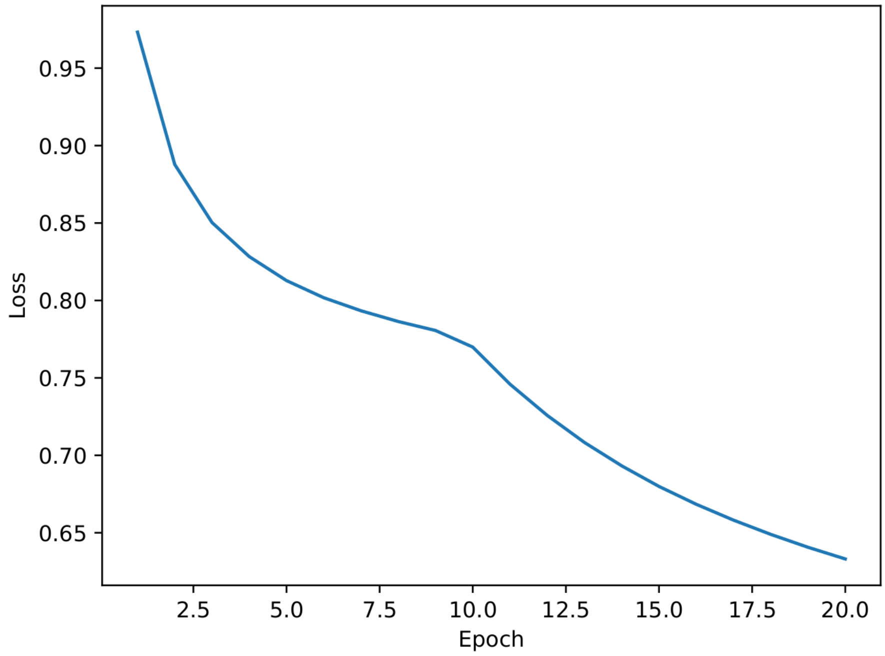


<u>Exercise 2</u>

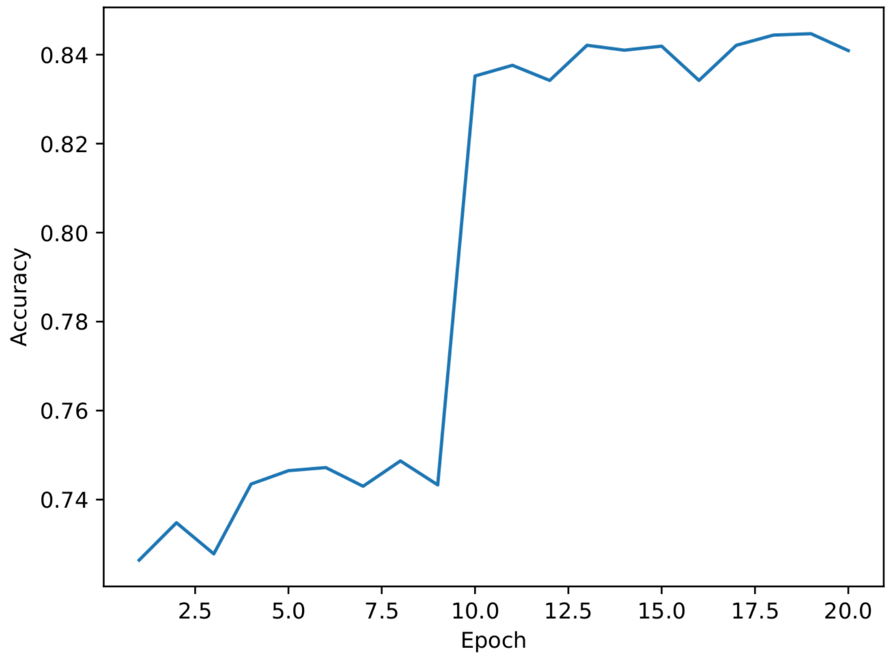
In this section we implemented a feed-forward neural network with one layer and compared the effects of different hyperparameters.

The final test accuracy with lr = 0.001: 0.8324

Training loss with lr = 0.001:

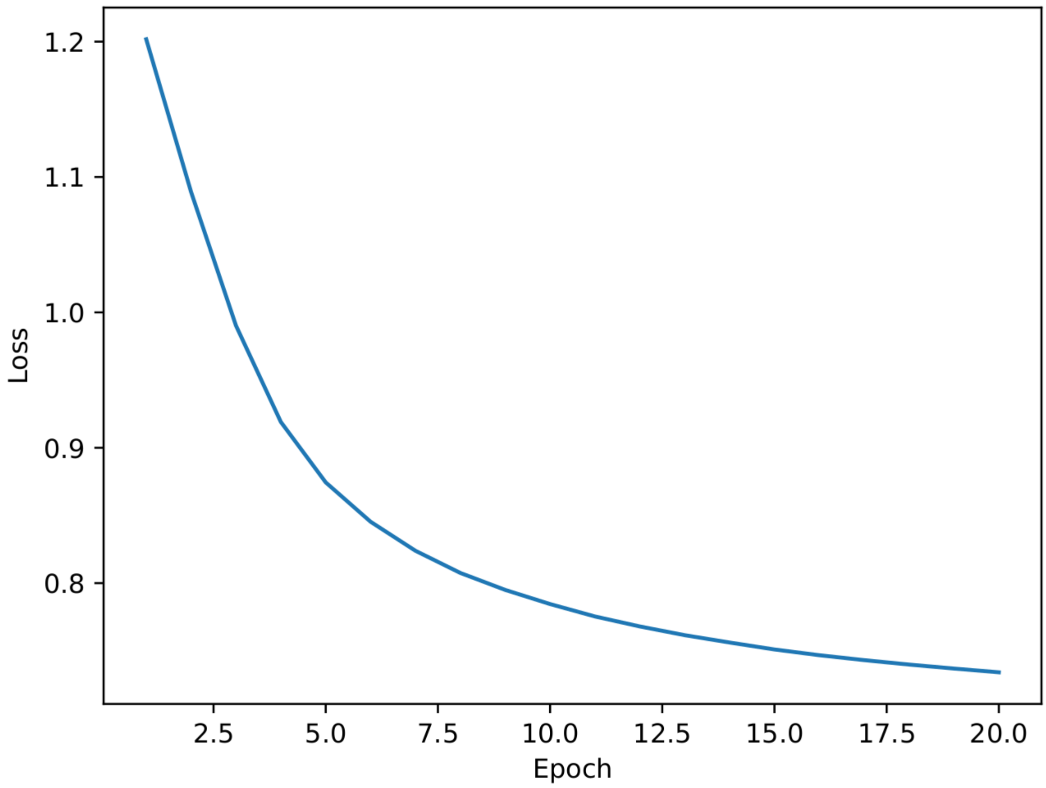


Validation accuracy with lr = 0.001:

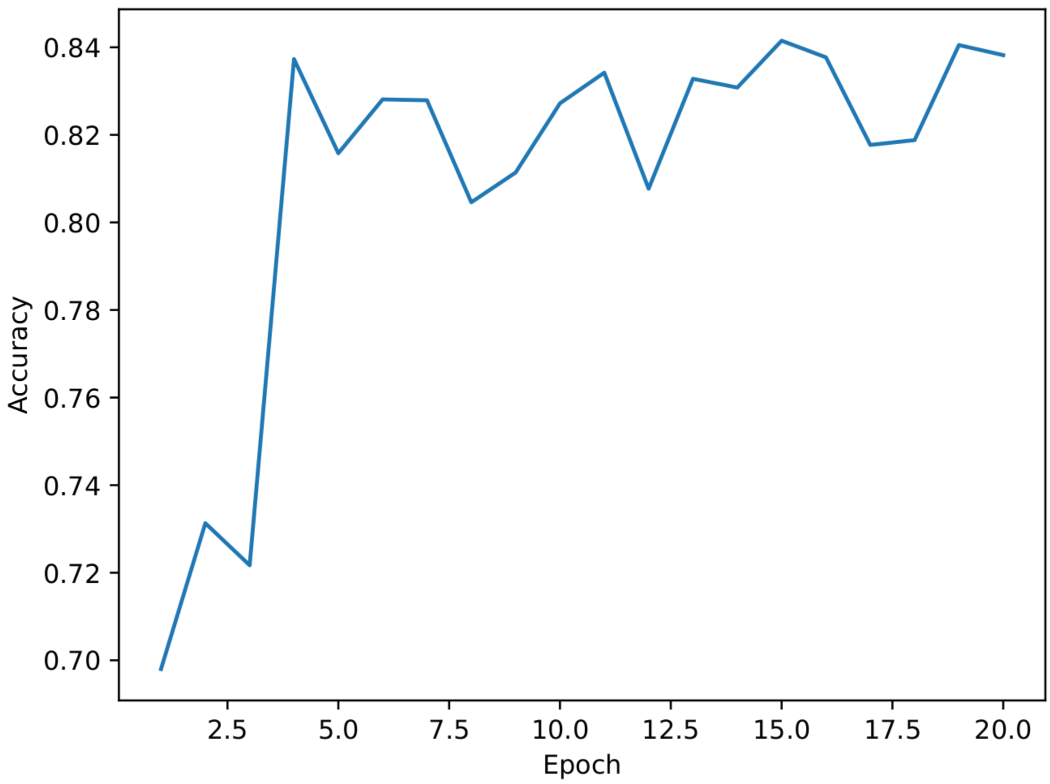


The final test accuracy with lr = 0.01: 0.8284

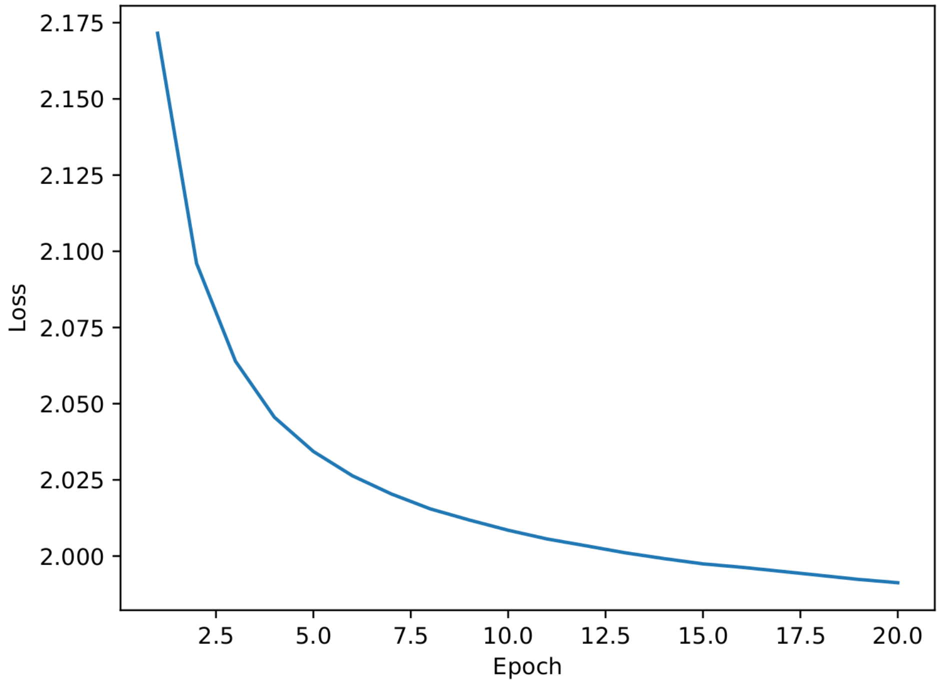
Training loss with lr = 0.01:



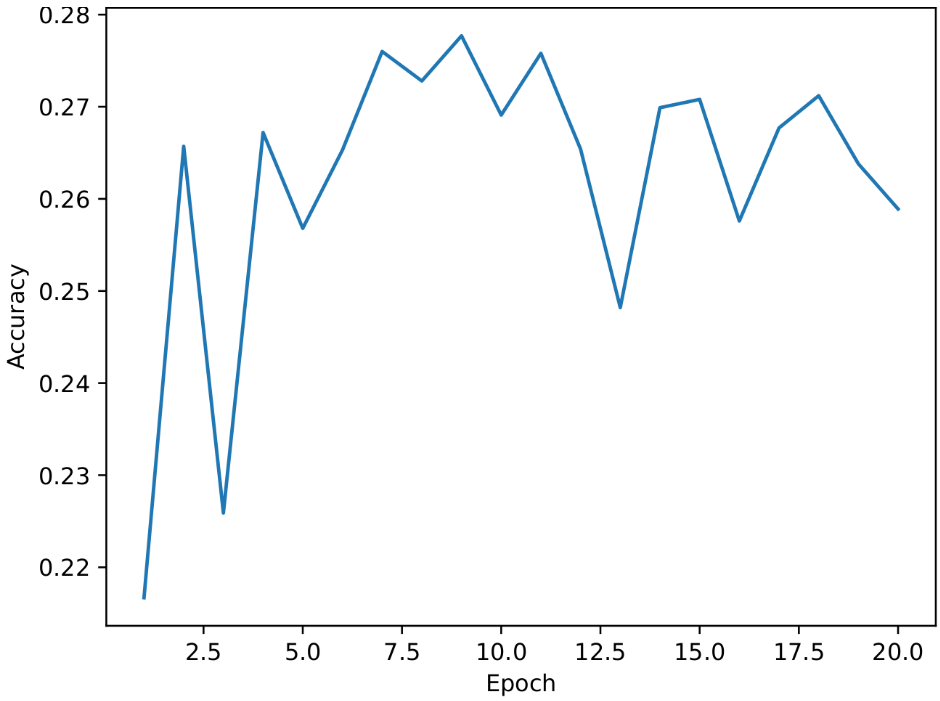
Validation accuracy with lr = 0.01:



The final test accuracy with lr = 0.1: 0.2552
Training loss with lr = 0.1:

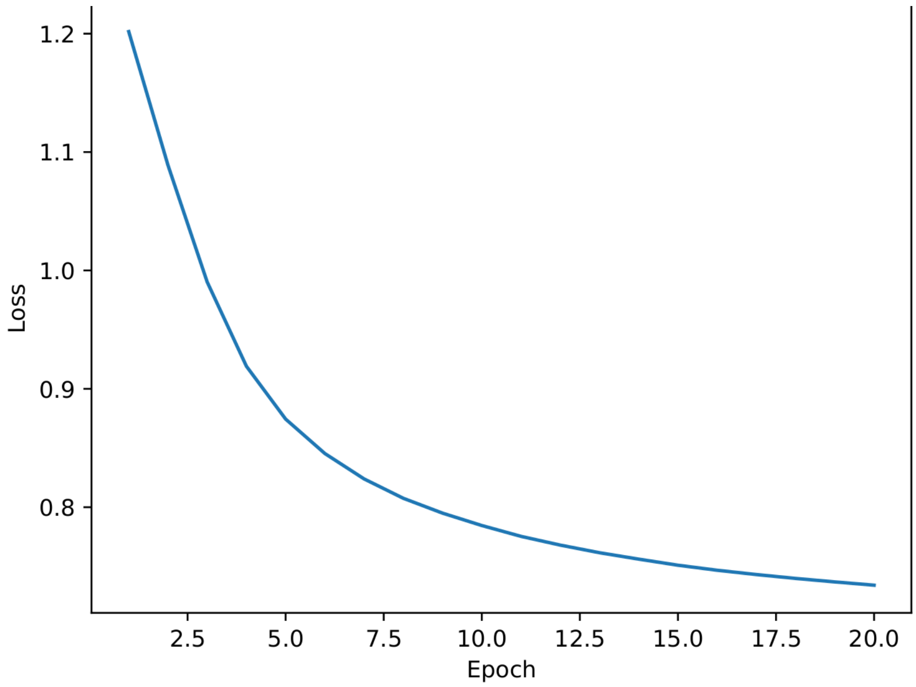


Validation accuracy with lr = 0.1:

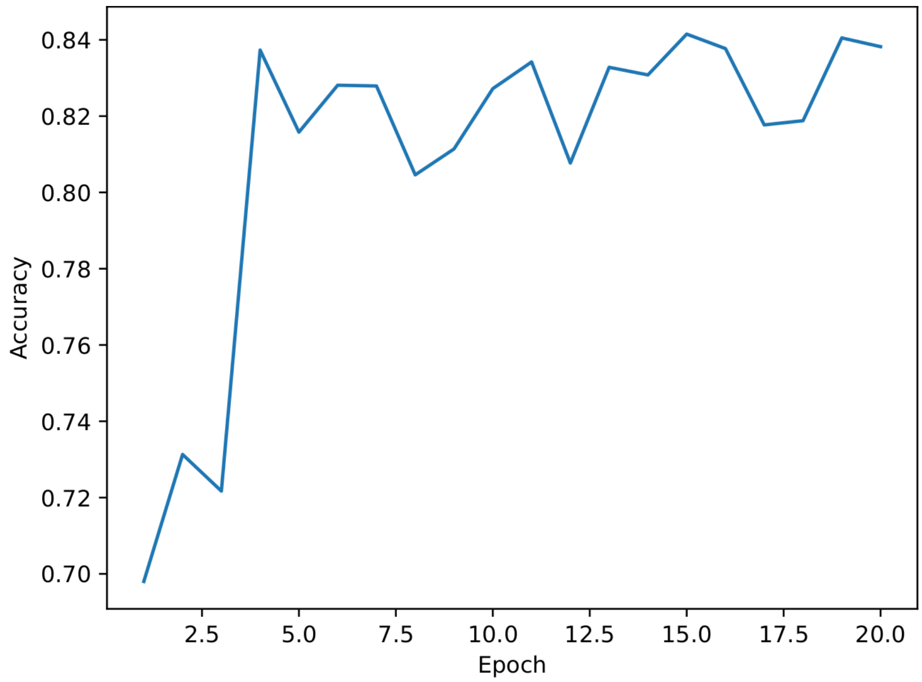


The final test accuracy with dropout probability = 0.3: 0.8284

Training loss with dropout probability = 0.3:

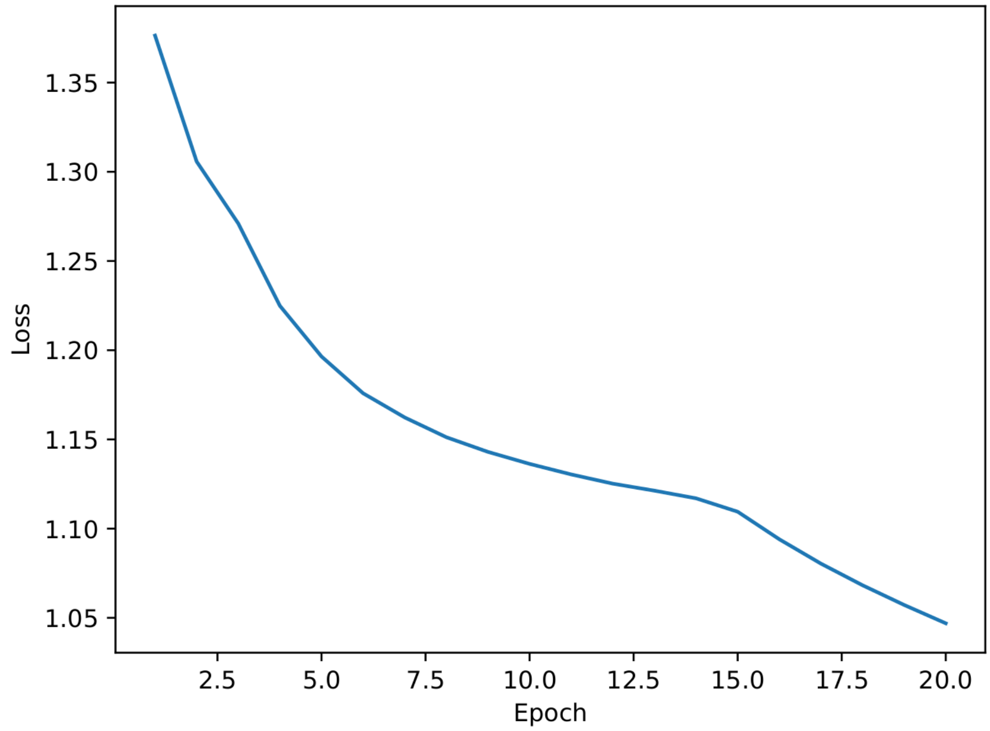


Validation accuracy with dropout probability = 0.3:

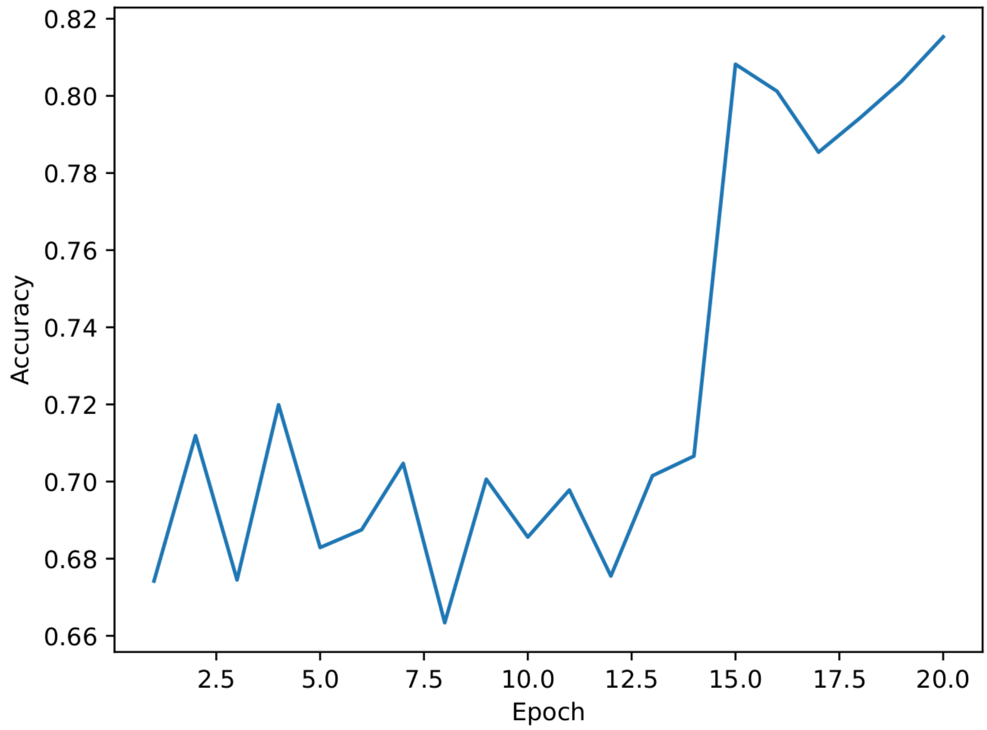


The final test accuracy with dropout probability = 0.5: 0.8114

Training loss with dropout probability = 0.5:

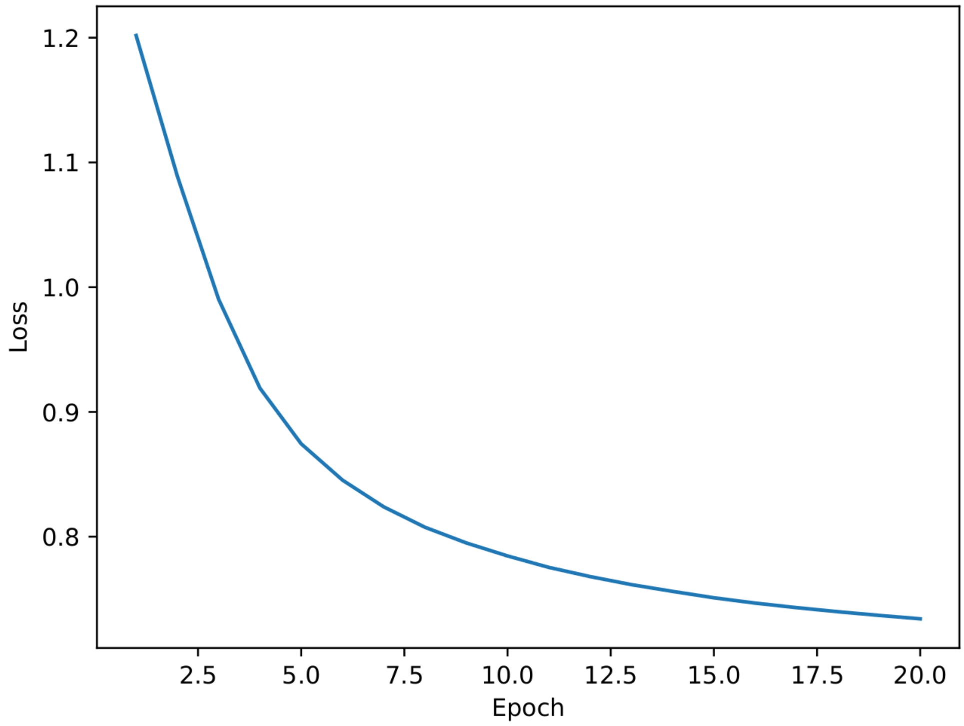


Validation accuracy with dropout probability = 0.5:

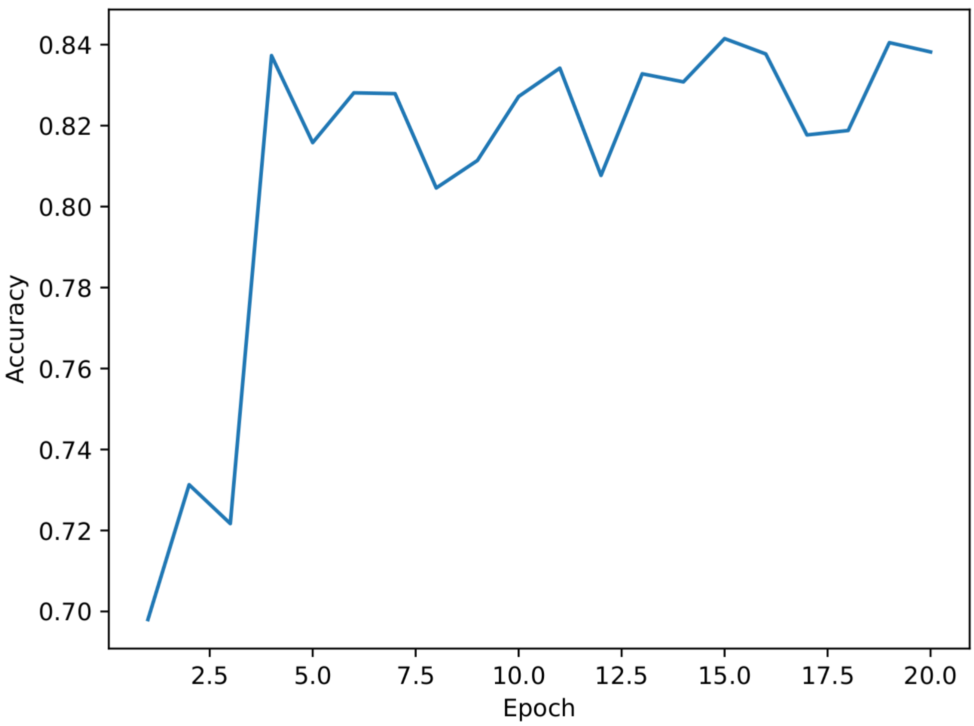


The final test accuracy with activation function ReLU: 0.8284

Training loss with activation function ReLU:

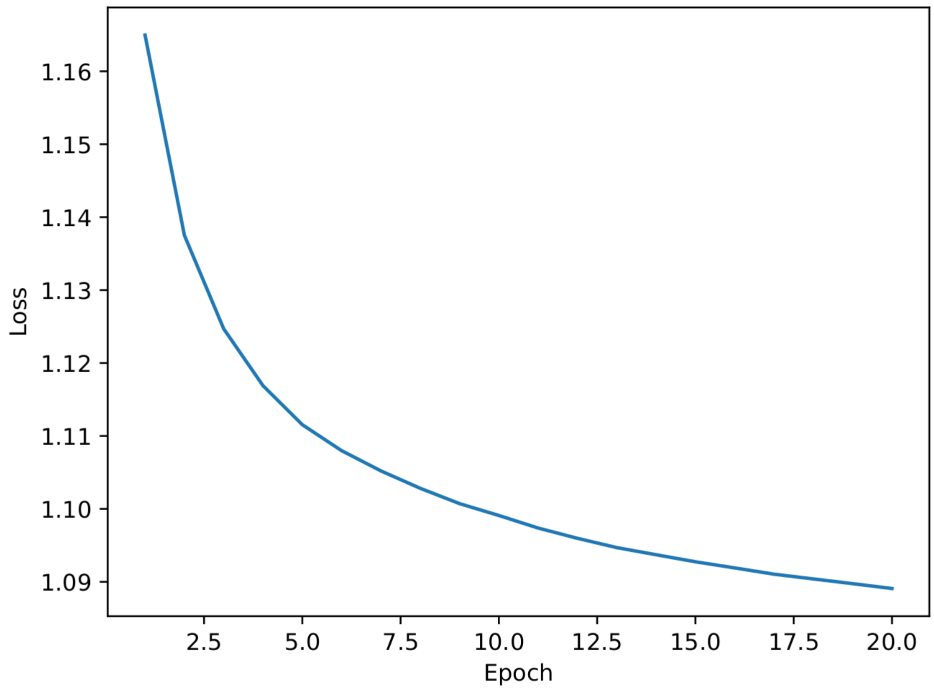


Validation accuracy with activation function ReLU:

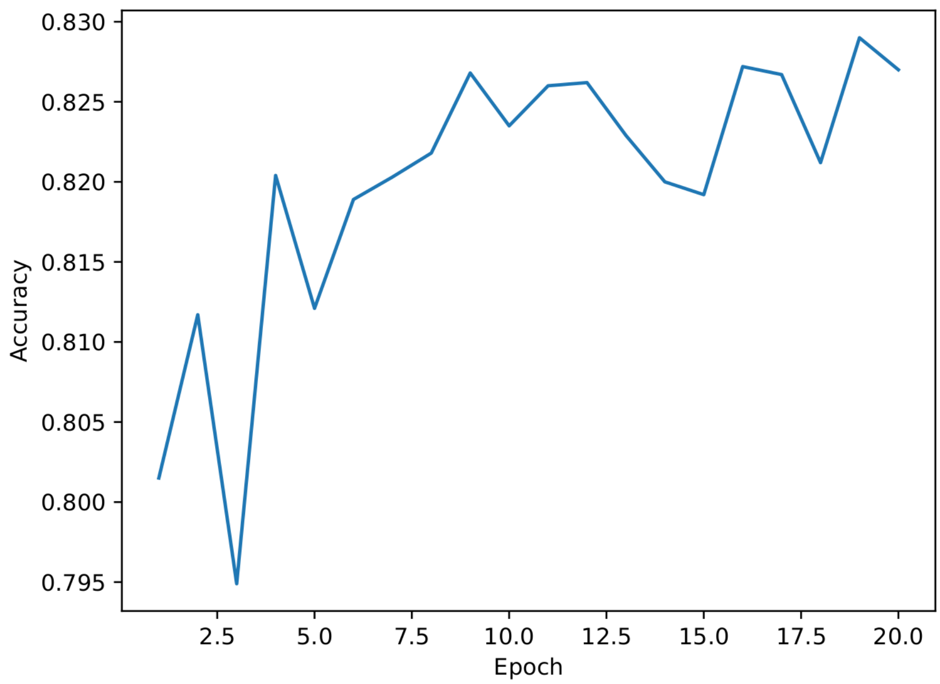


The final test accuracy with activation function Tanh: 0.8207

Training loss with activation function Tanh:

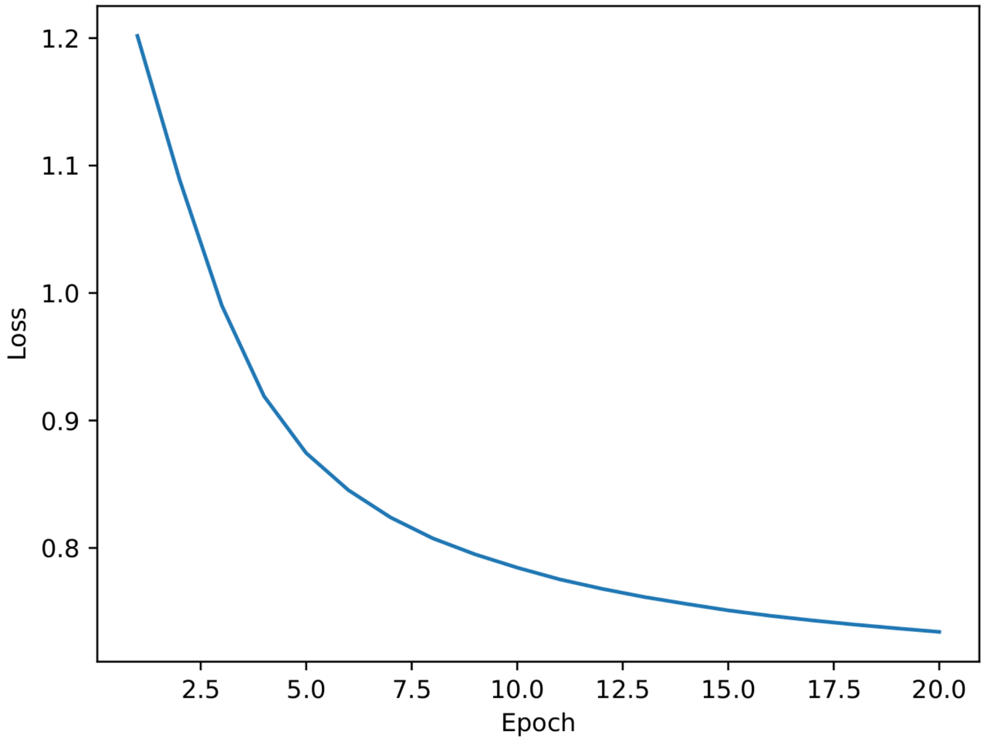


Validation accuracy with activation function Tanh:

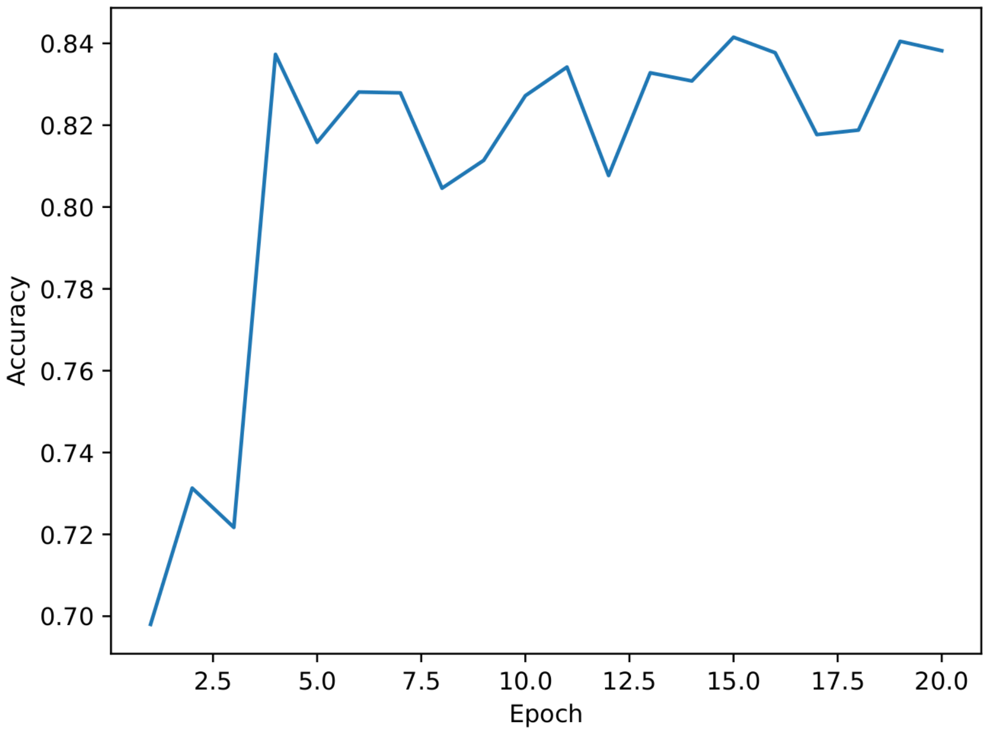


The final test accuracy with optimizer SGD: 0.8284

Training loss with optimizer SGD:

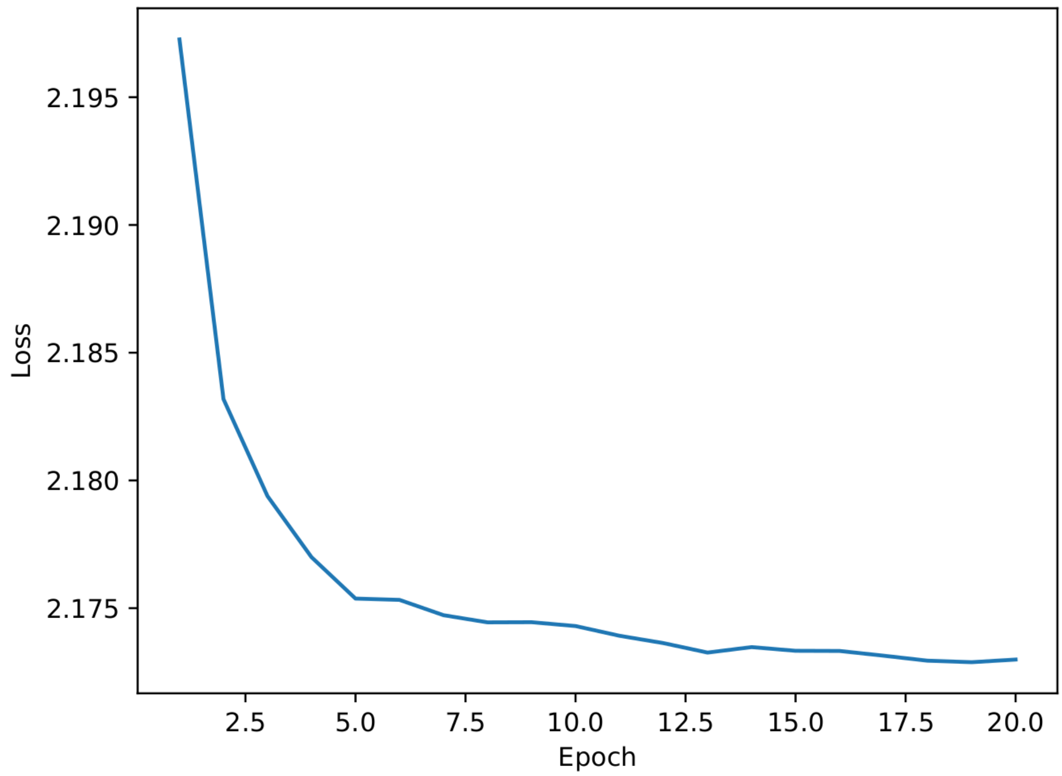


Validation accuracy with optimizer SGD:

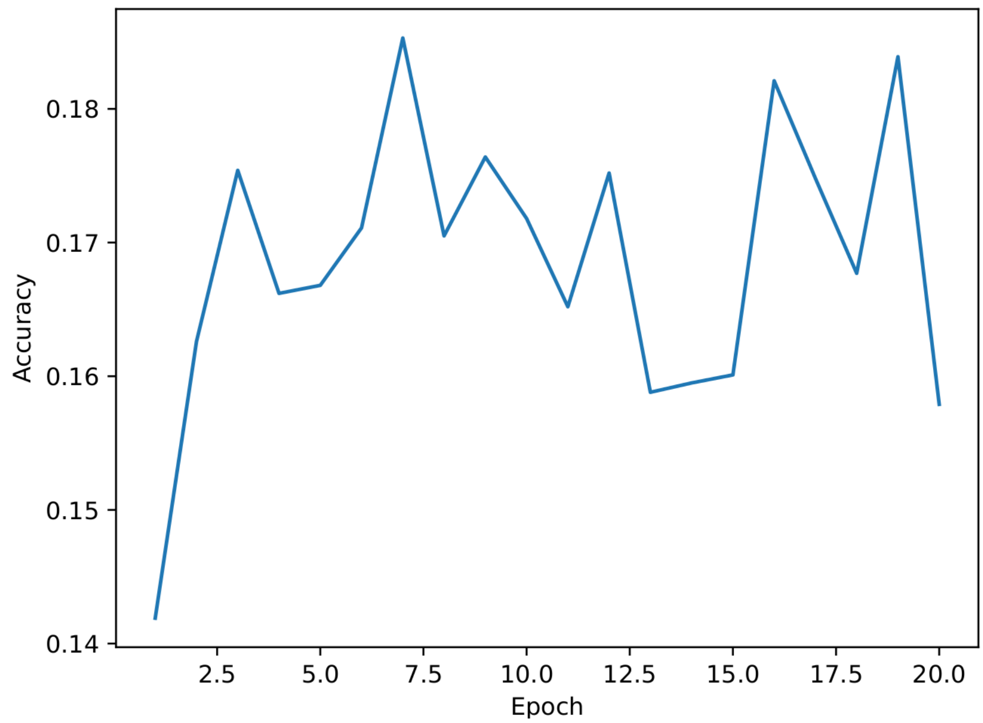


The final test accuracy with optimizer ADAM: 0.1579

Training loss with optimizer ADAM:



Validation accuracy with optimizer ADAM:

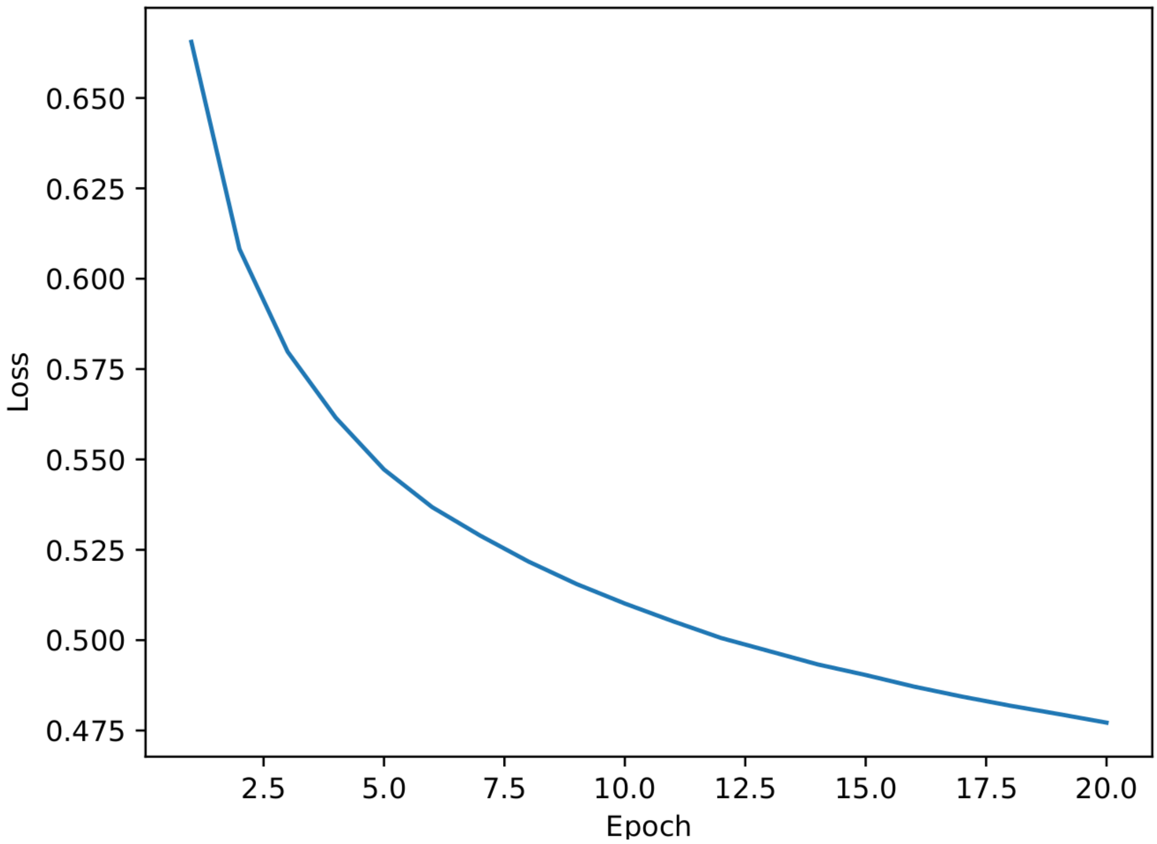


Taking into account all the test accuracies measured, it is clear that the highest one was observed when implementing the neural network with a learning rate of 0.001. Although most of them have similar performances, one could speculate that smaller "steps" during the optimization process leads to function approximators that better generalize to the test data.

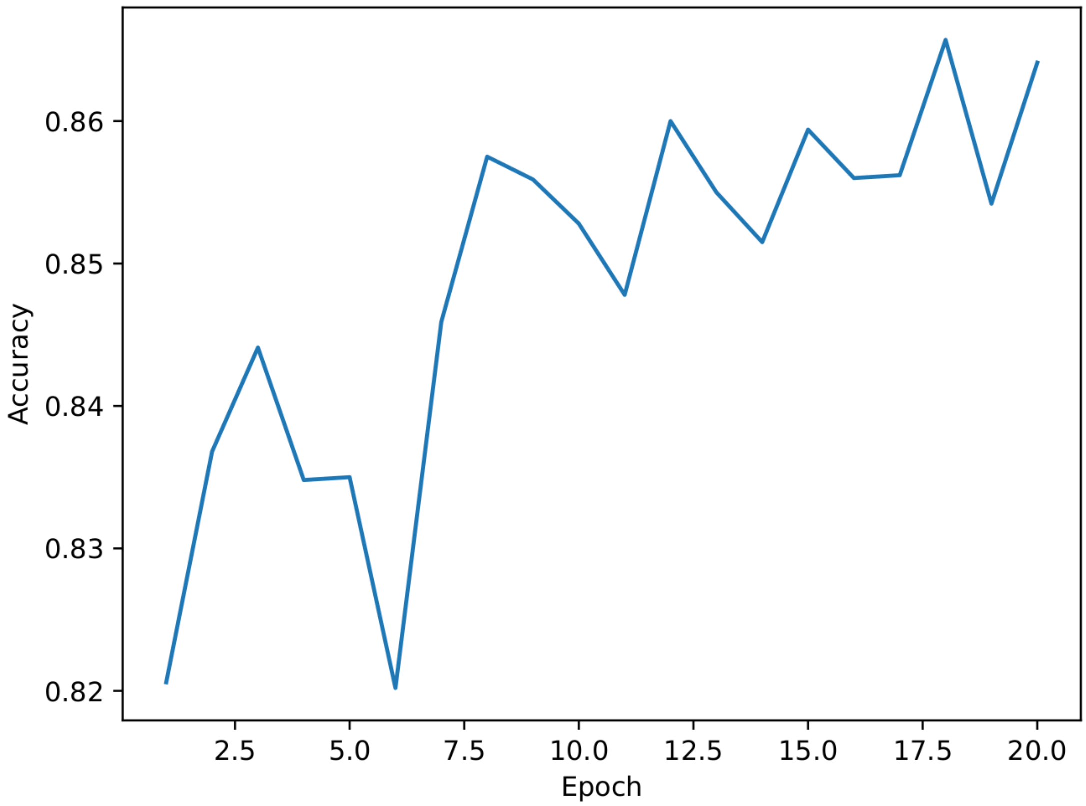
Exercise 3

In this final exercise, we implemented a 2-layer and a 3-layer neural network using the same hyperparameters as the previous exercise. As before, performances were plotted and tested.

The final test accuracy with 2 layers: 0.8576
Training loss with 2 layers:

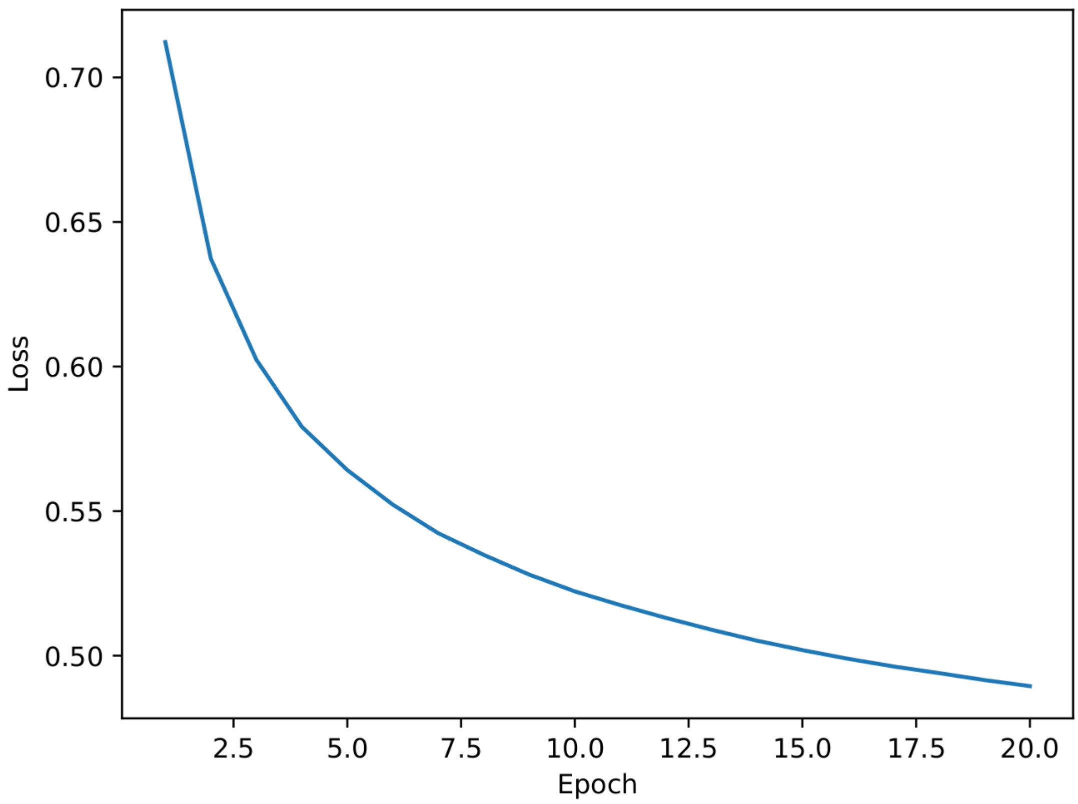


Validation accuracy with 2 layers:

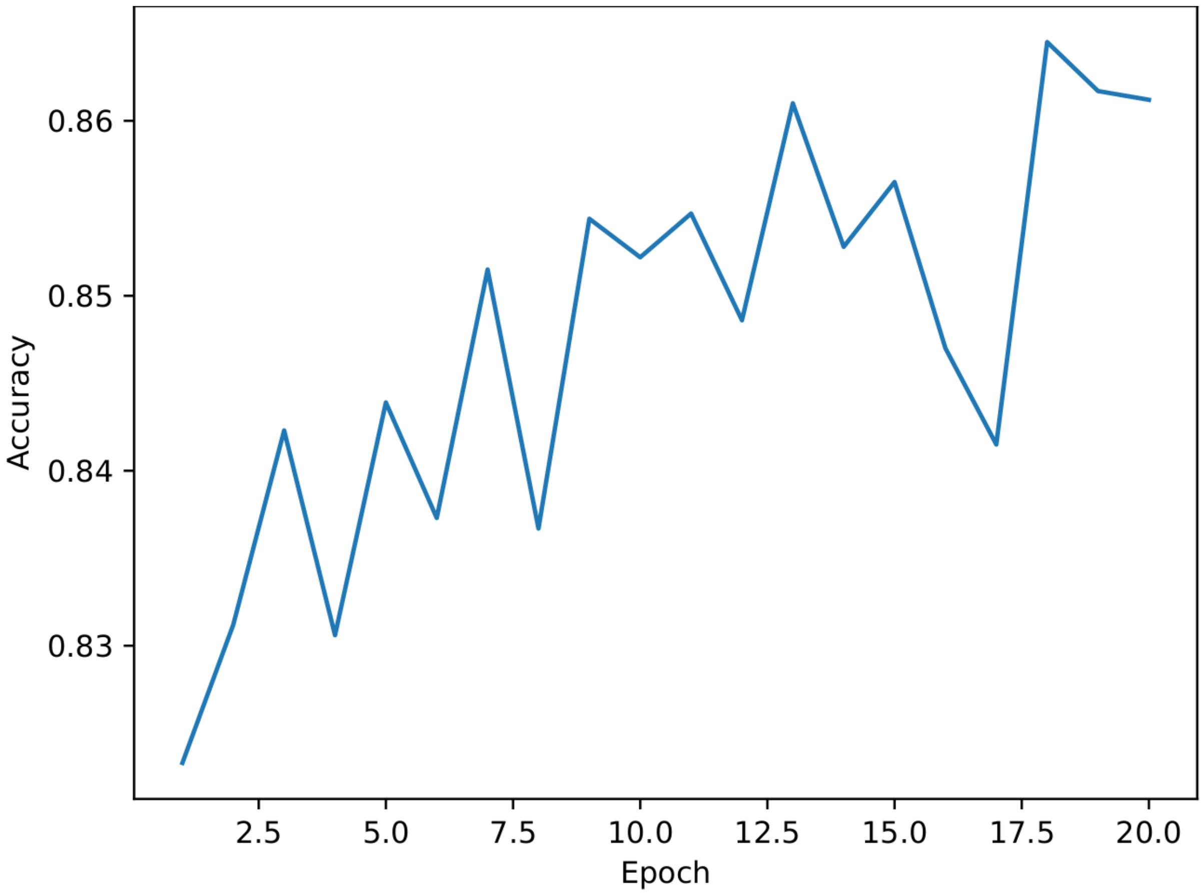


The final test accuracy with 3 layers: 0.8573

Training loss with 3 layers:



Validation accuracy with 3 layers:



As expected, the difference in the test accuracies of 1-layer and 2-layer perceptrons are noticeable, with the 3-layer perceptron having similar performance to the 2-layer, with slightly less test accuracy but generally higher valid accuracies during training.