MEIC 23/24

Deep Learning

Homework 1

Both students contributed to answer the homework questions with the same commitment. The code development and theory questions have been produced by all members of the group.

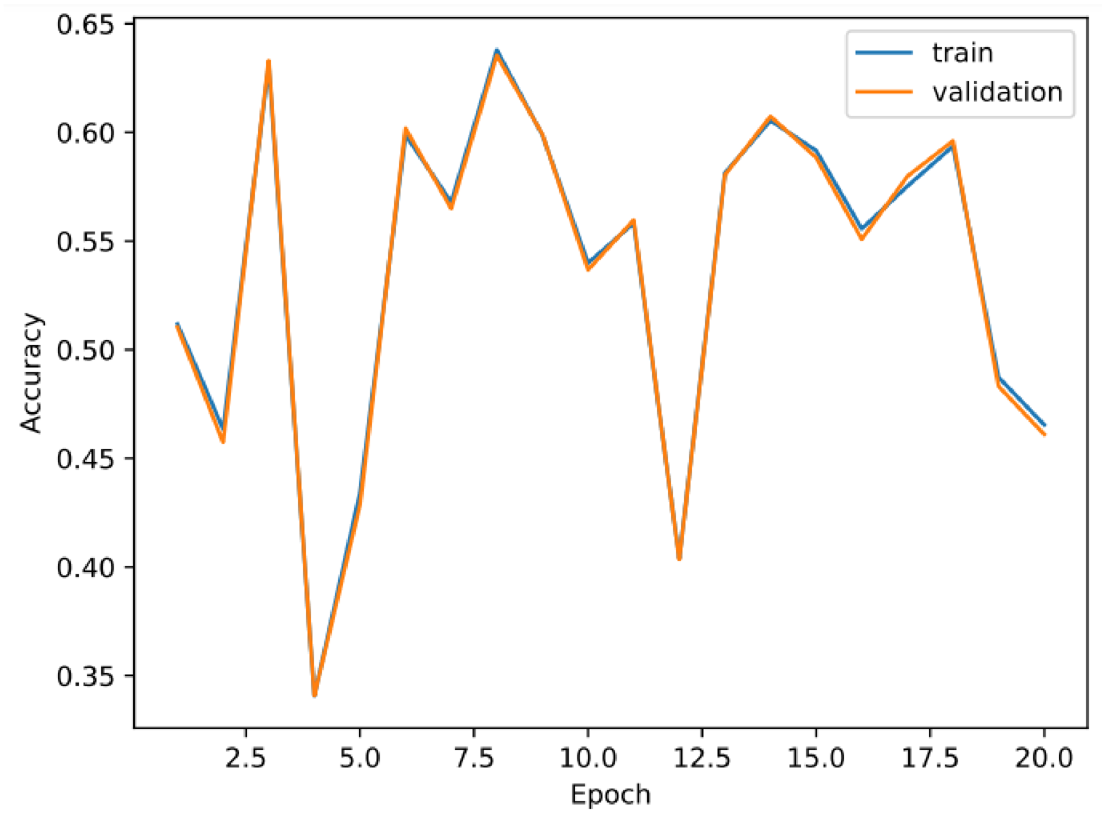
Group 65

99286 Miguel Mano

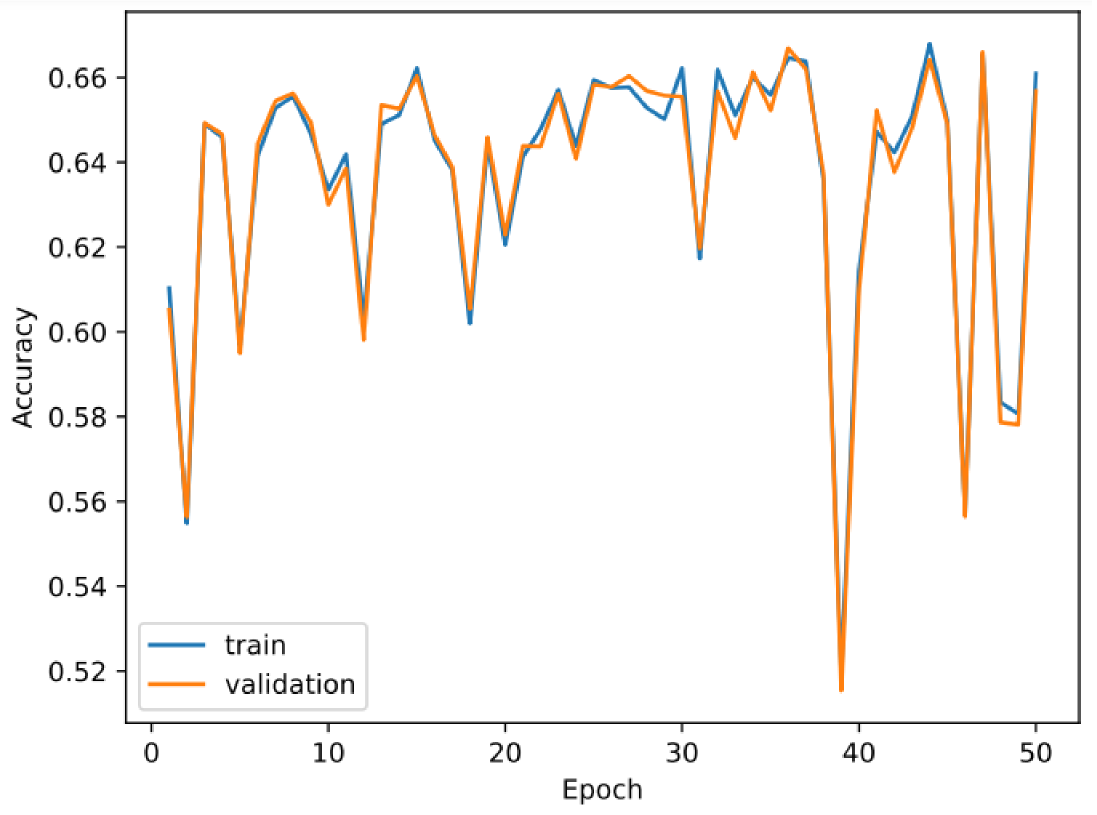
99300 Pedro Rodrigues

Question 1

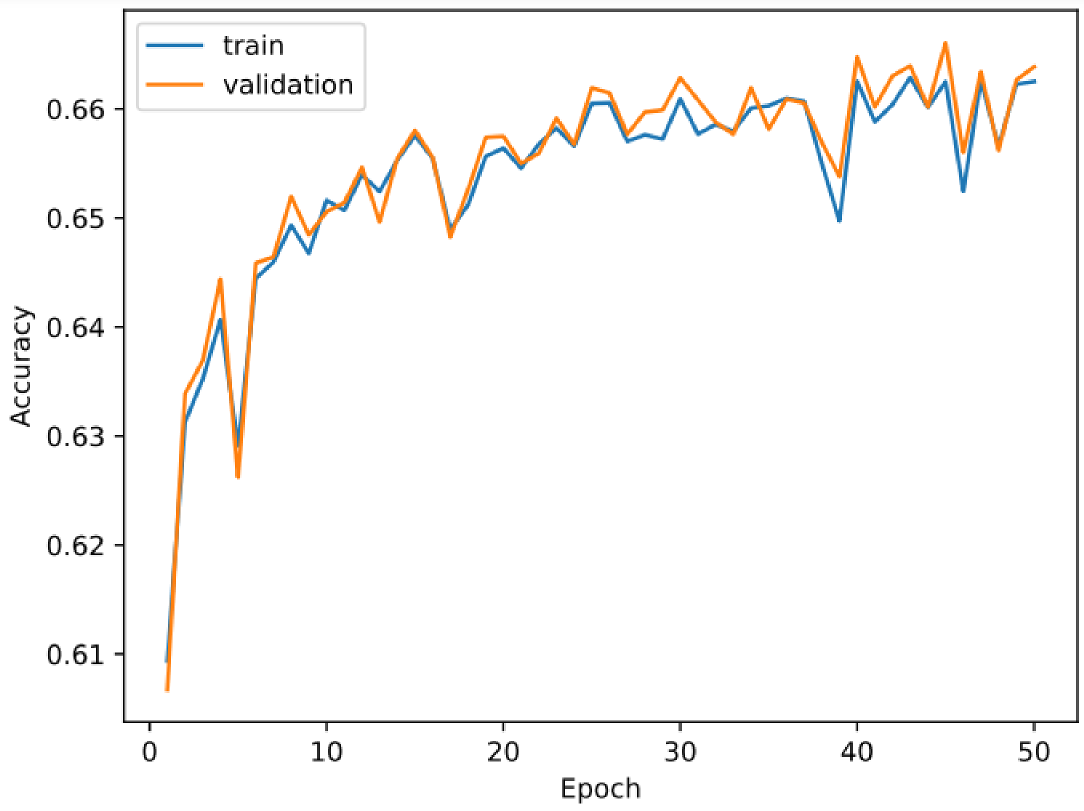
1.a) *Final Test Accuracy* - **0.3422**



1.b) *Final Test Accuracy for LR of 0.01* – **0.5784**



Final Test Accuracy for LR of 0.001 – **0.5936**



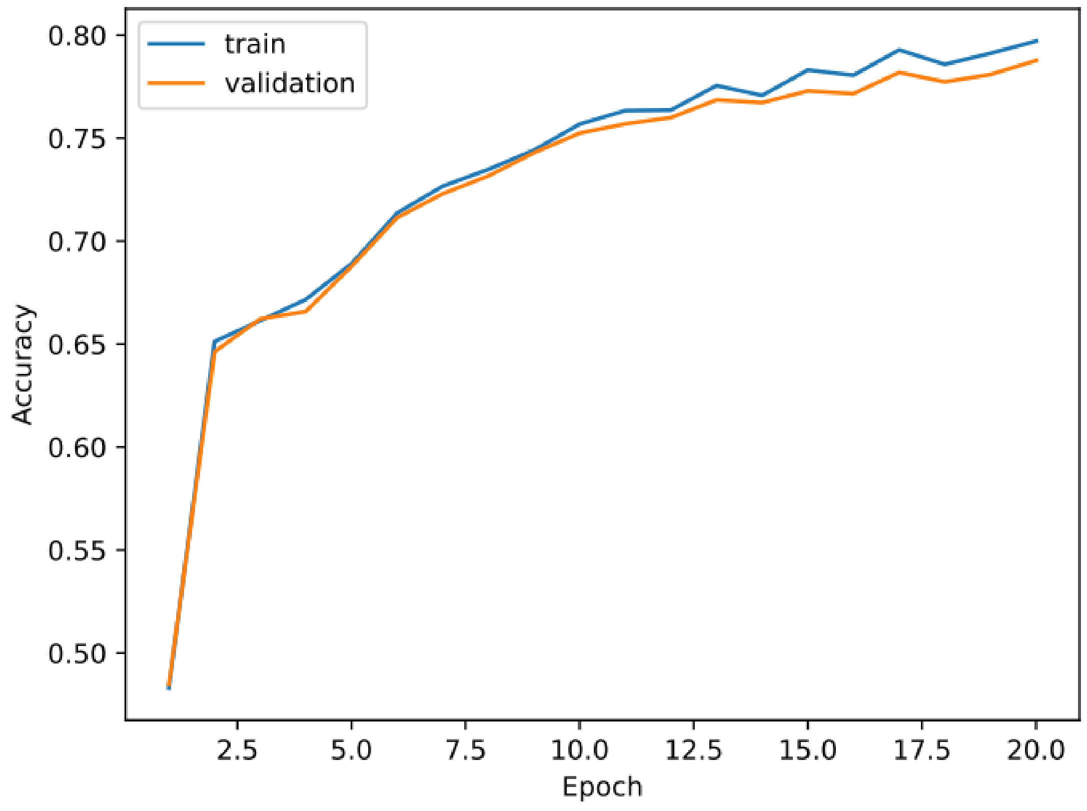
2.a)

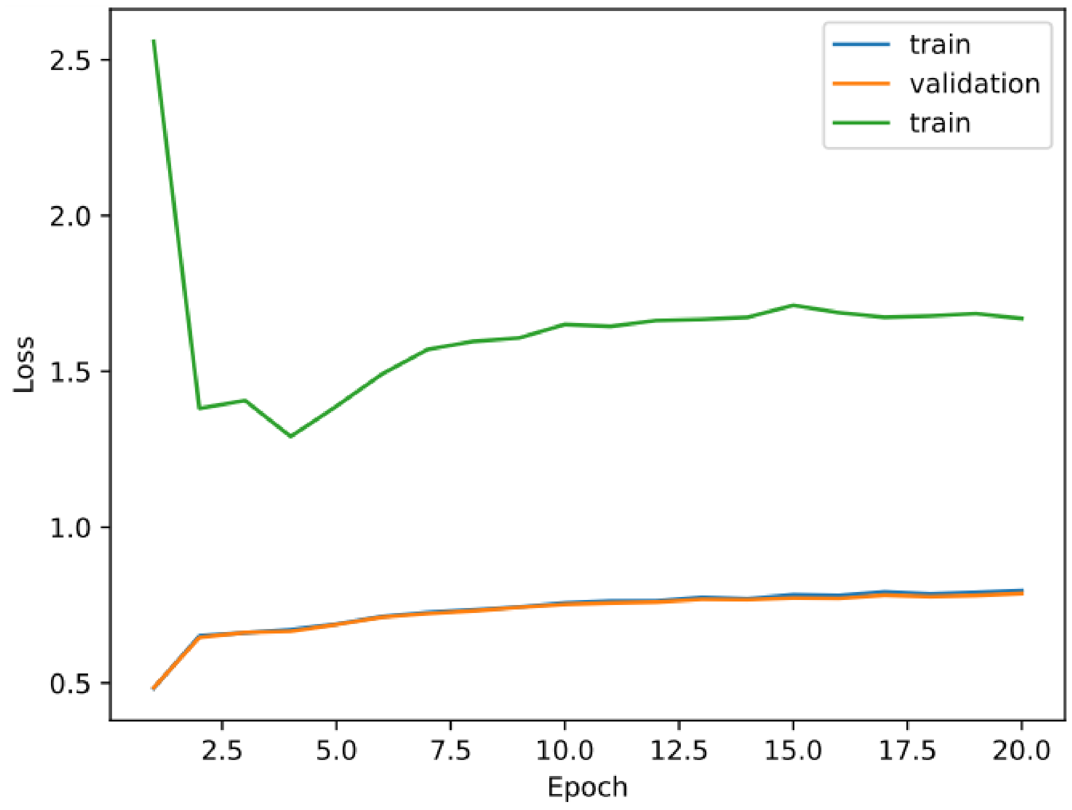
Both claims are true. Logistic regression, being a type of linear model, primarily captures linear relationships between input and output variables. In the other hand, the multi-layer perceptron (MLP) employs Rectified Linear Unit (**ReLU**) activations within its hidden layers, enabling the model to comprehend non-linear relationships, facilitating the learning of intricate patterns. This capability makes MLP more adept at recognizing complex patterns, particularly beneficial in tasks like image recognition.

Regarding optimization, logistic regression often constitutes a convex optimization problem due to the convergence of gradient descent to a single minimum point. In contrast, MLP, with its multiple layers and **ReLU** activations, poses a more challenging scenario in finding a global minimum. In addition to this, the MLP with **ReLU** function training process is sensitive to hyperparameter initialization and allows to eliminate the linear dependence on the biases and weights.

Therefore, training a logistic regression model is easier compared to the complexities involved in MLP training.

2.b) *Final Test Accuracy* – **0.7656**





Question 2

1. Need to run these commands, one by one:

- *python3 hw1-q2.py -epochs 20 -batch_size 16 -learning_rate 0.001 logistic_regression*

Final Test Accuracy for LR 0.001 – **0.6503**

- *python3 hw1-q2.py -epochs 20 -batch_size 16 -learning_rate 0.01 logistic_regression*

Final Test Accuracy for LR 0.01 – **0.6200**

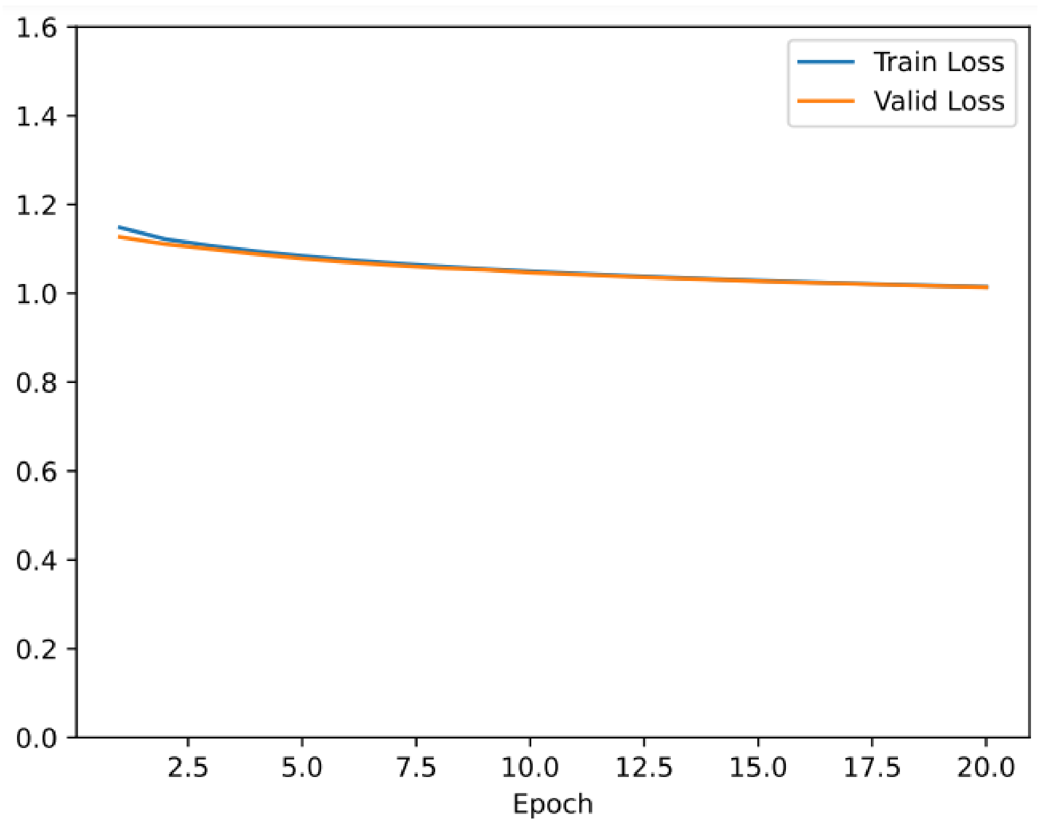
- *python3 hw1-q2.py -epochs 20 -batch_size 16 -learning_rate 0.1 logistic_regression*

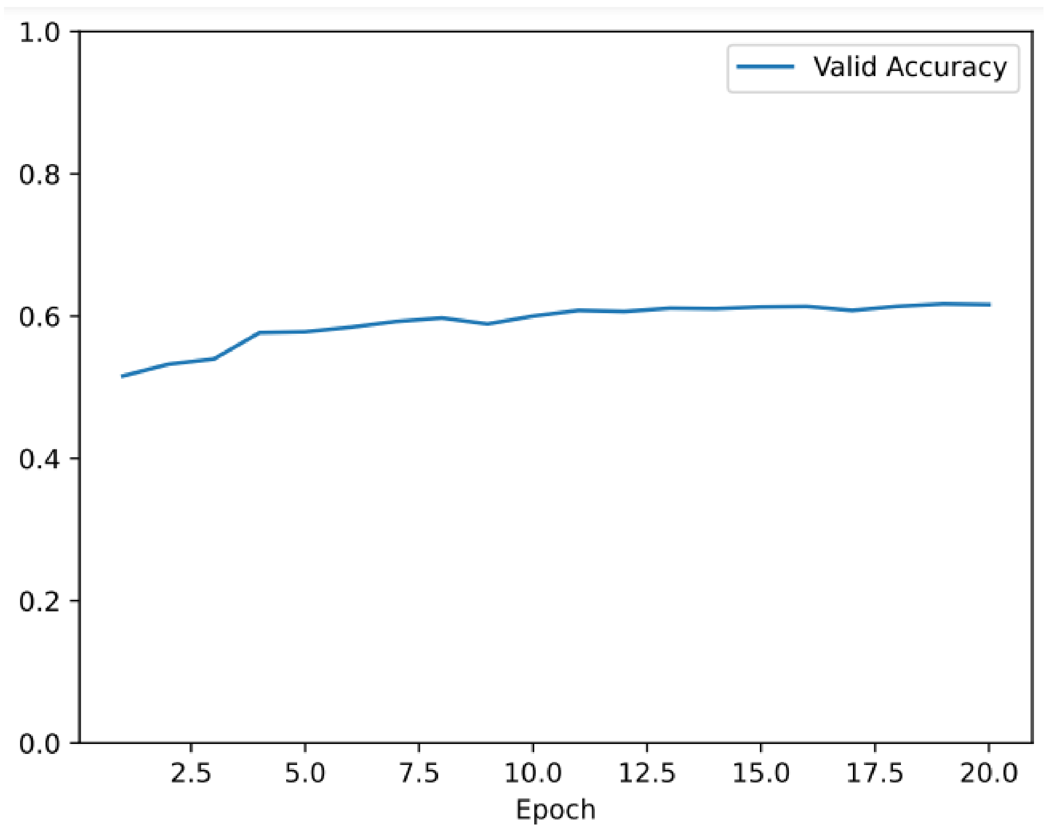
Final Test Accuracy for LR 0.1 – **0.5577**

In terms of final validation accuracy, the model with best learning rate configuration for this task is **0.001**. – Final Accuracy Value of **0.6503**

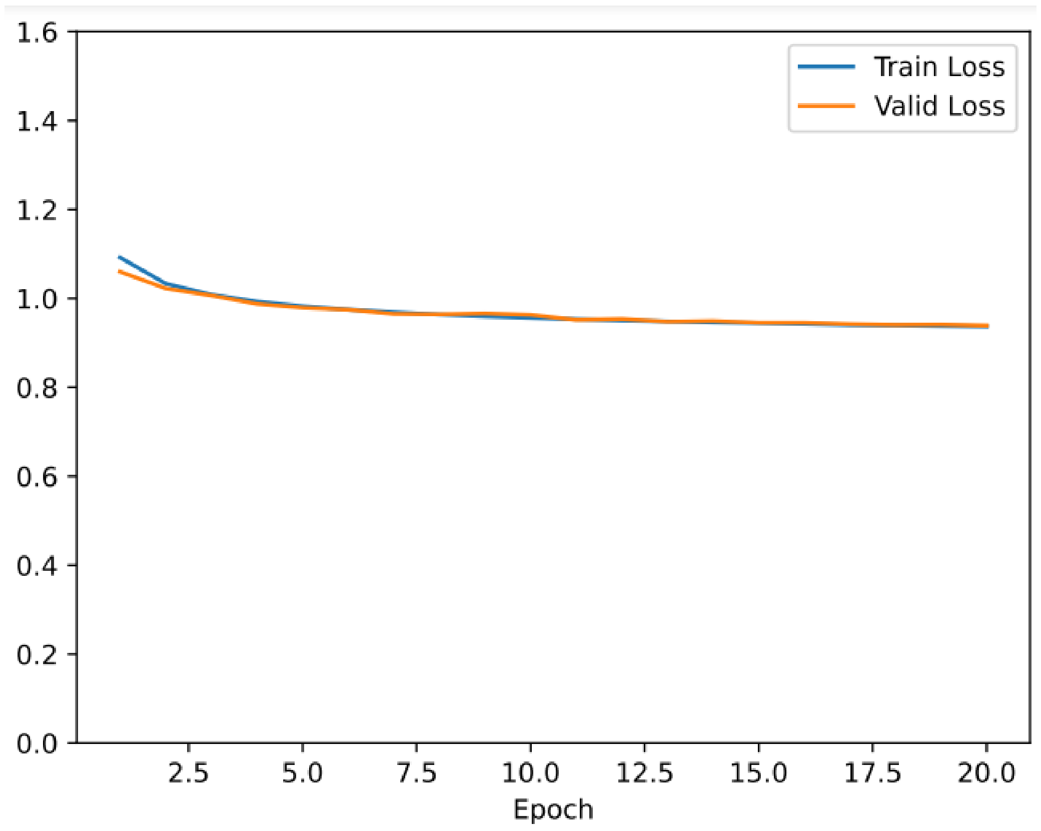
In the following images, you can observe 3 (Training Loss + Final Validation Accuracy) graphs for learning rate of 0.001, 0.01 and 0.1, respectively.

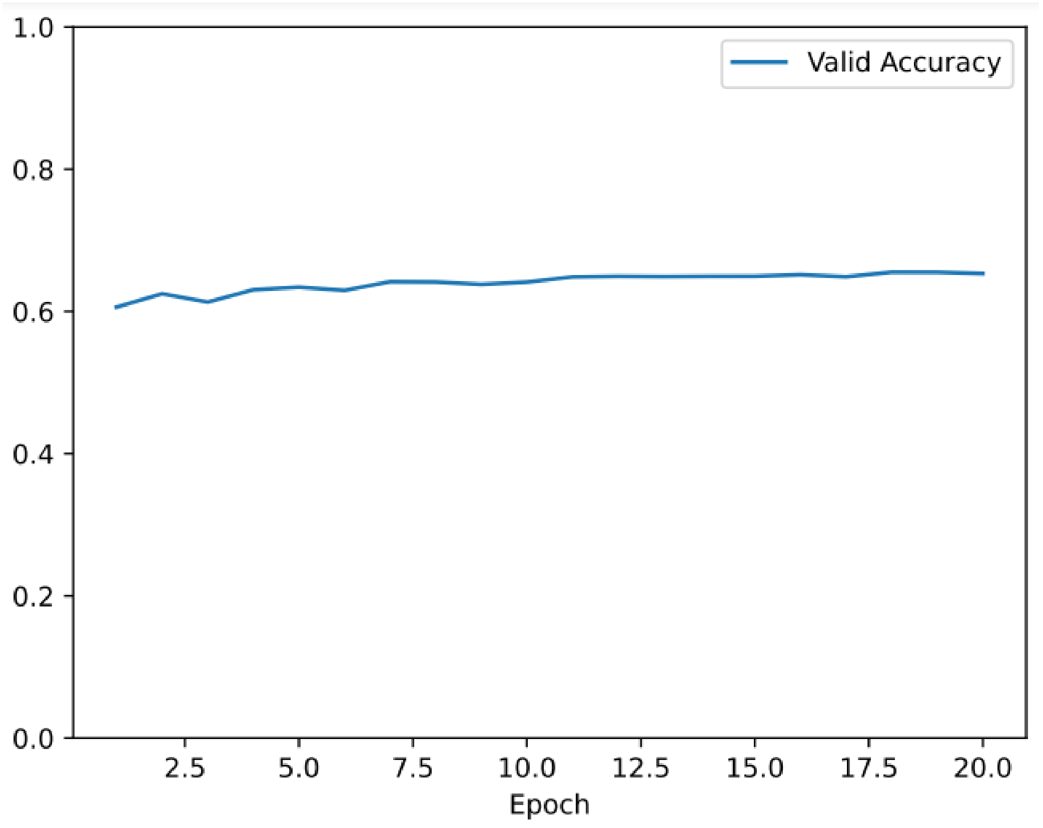
Learning Rate of 0.001



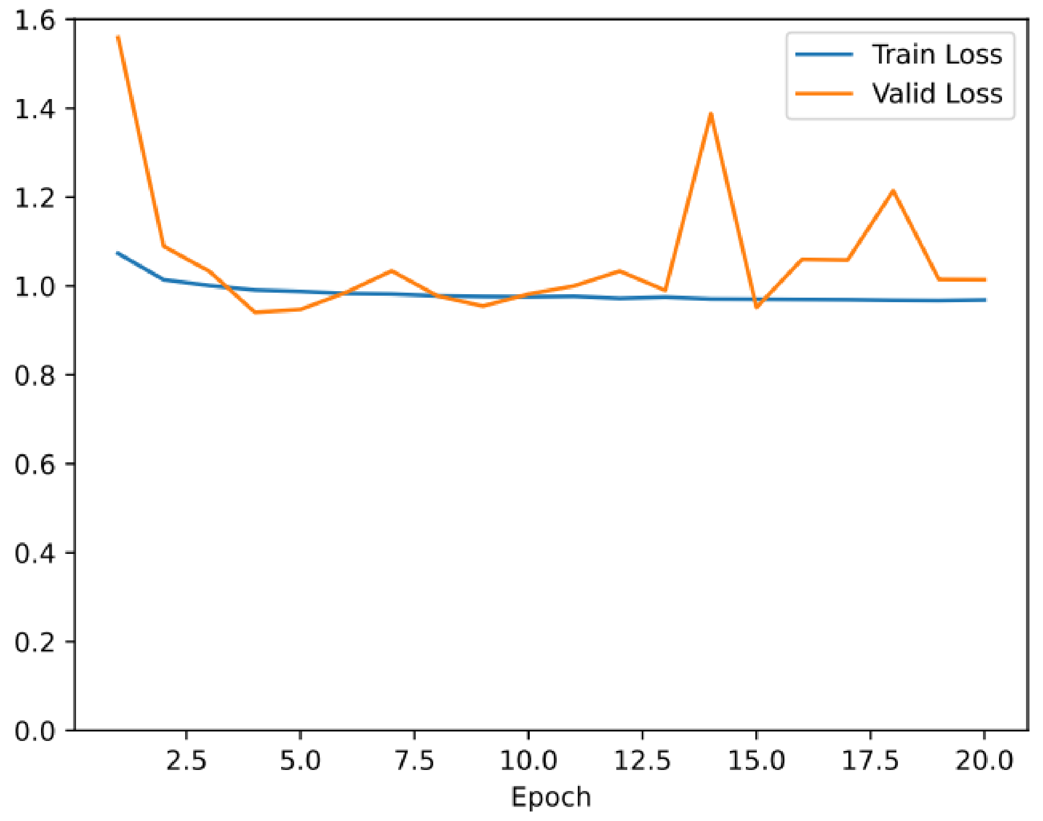


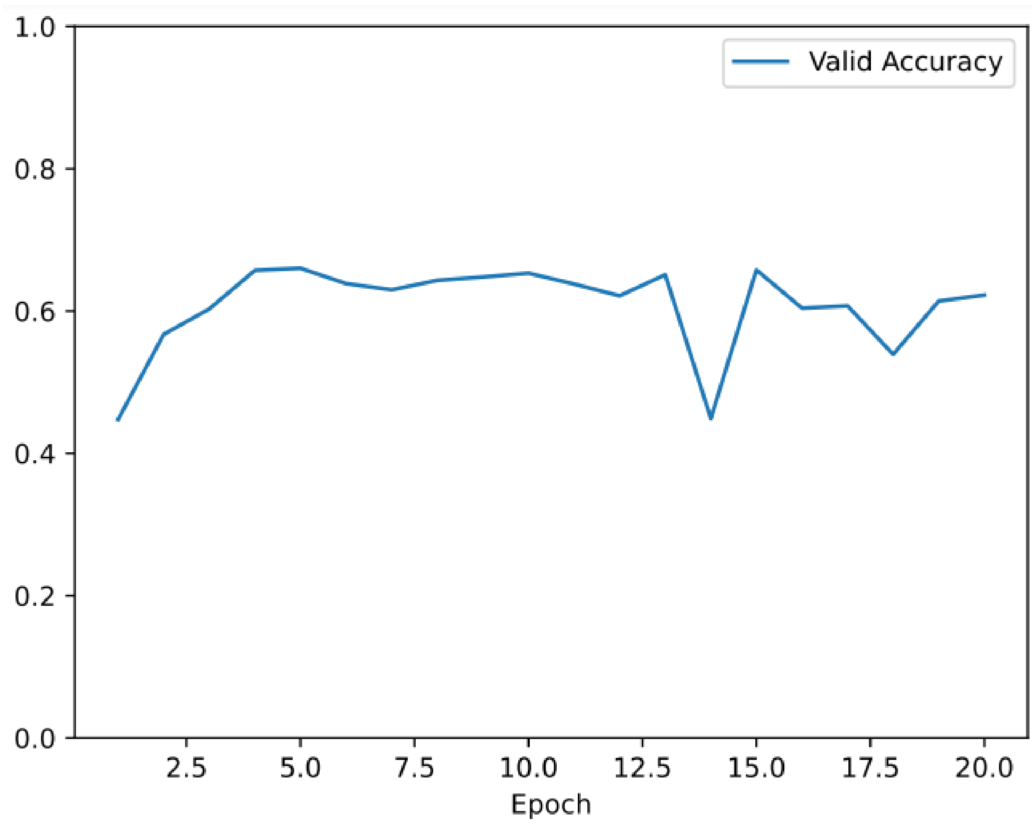
Learning Rate of 0.01





Learning Rate of 0.1



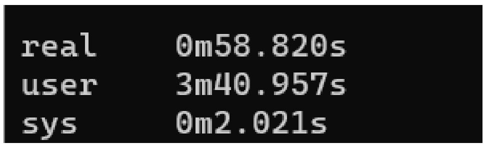


2. a)

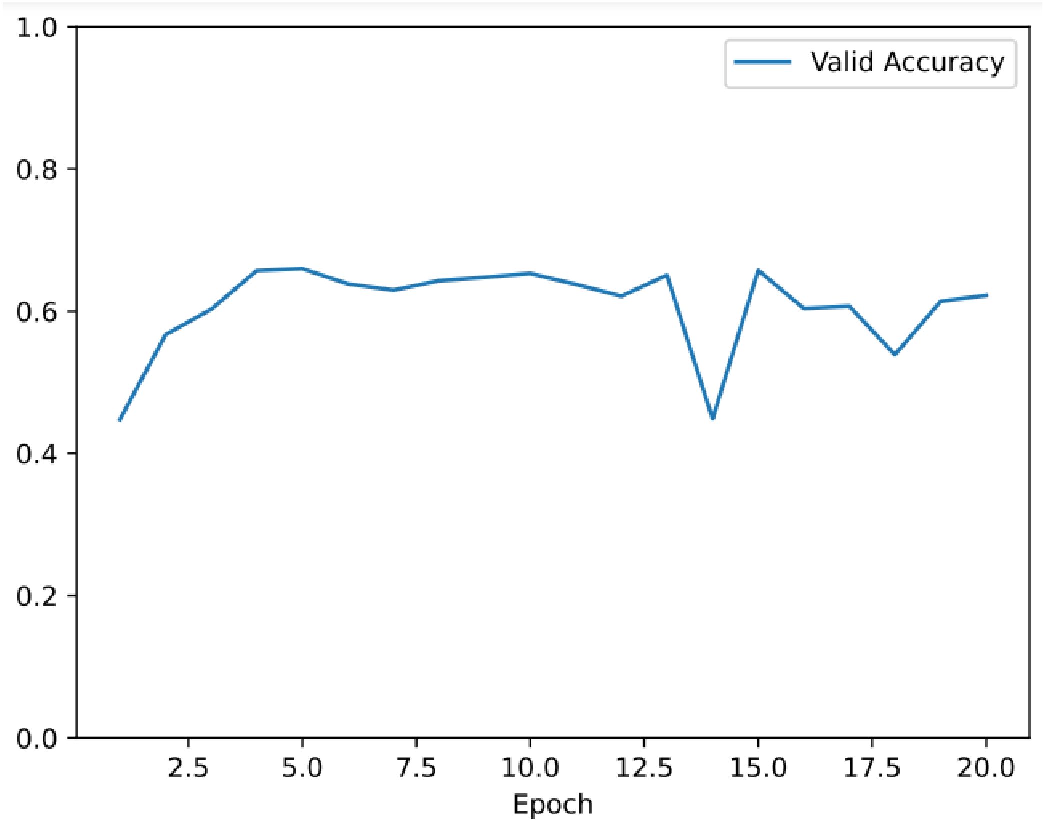
python3 hw1-q2.py -epochs 20 -batch_size 16 -learning_rate 0.1 -hidden_size 200 -layers 2 -dropout 0.0 -activation relu -optimizer sgd logistic_regression

Final Test Accuracy for batch size of 16 – **0.5577**

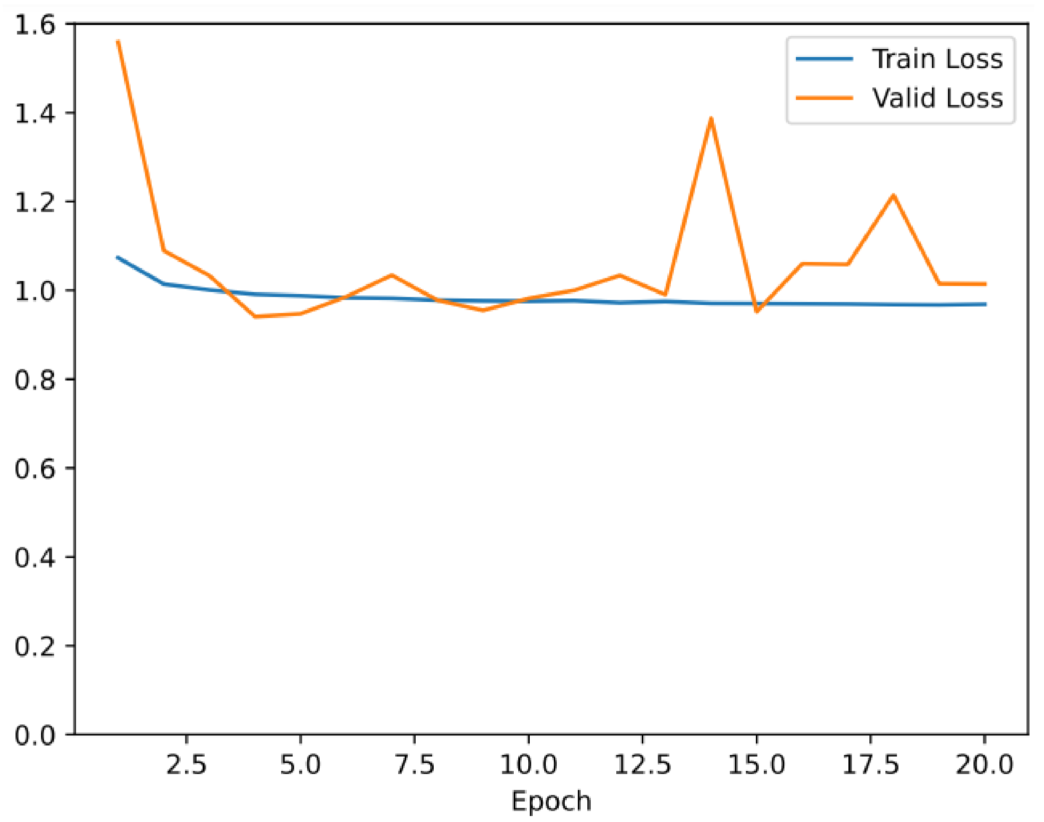
Time



```
real    0m58.820s
user    3m40.957s
sys     0m2.021s
```



Valid Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epoch

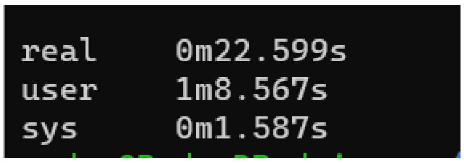


Train Loss
Valid Loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epoch

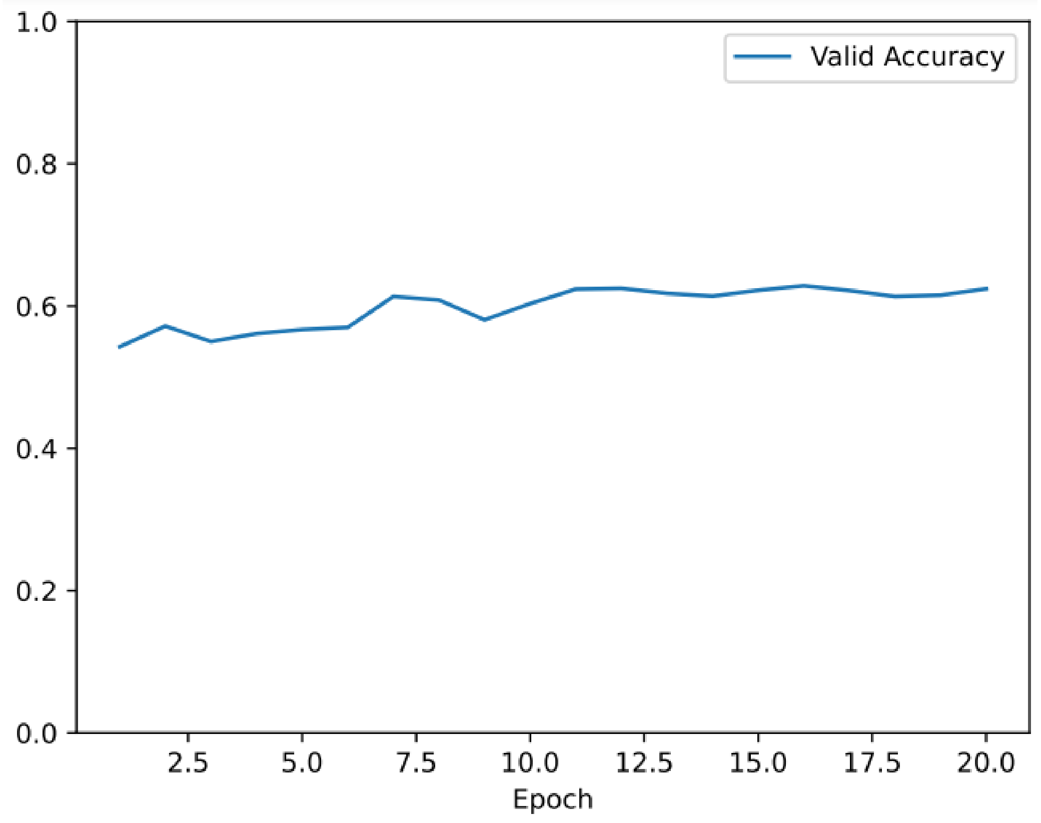
python3 hw1-q2.py -epochs 20 -batch_size 1024 -learning_rate 0.1 -hidden_size 200 -layers 2 -dropout 0.0 -activation relu -optimizer sgd logistic_regression

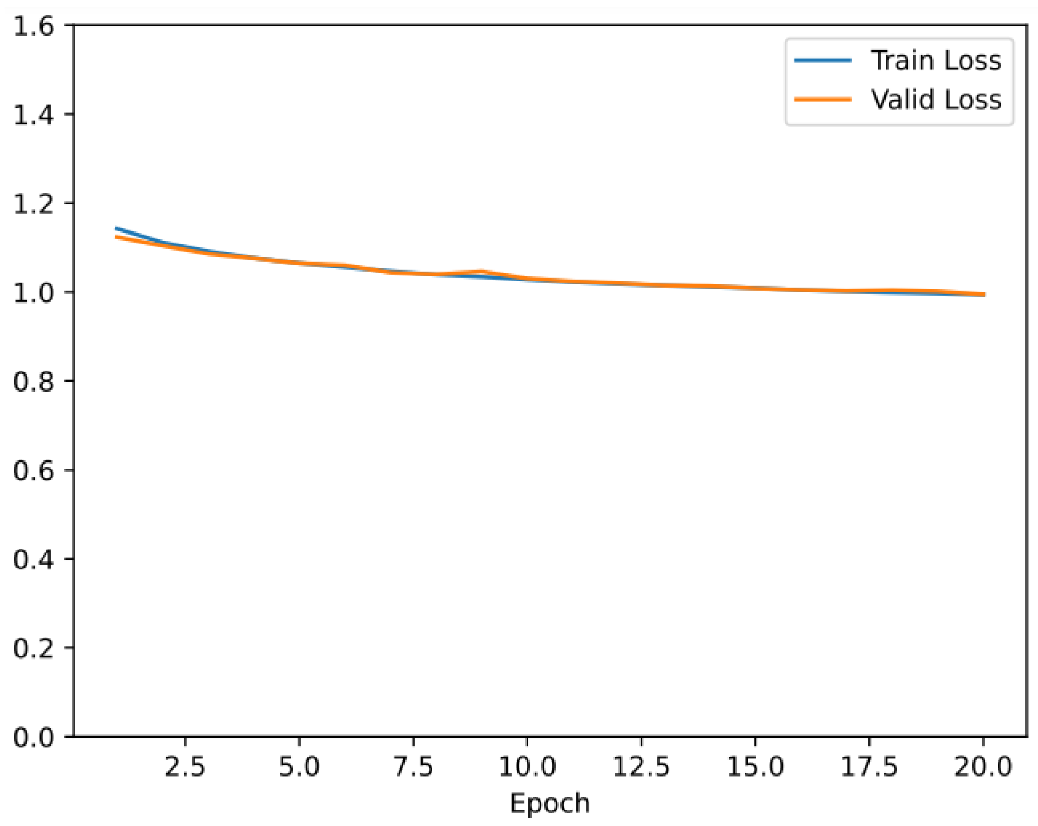
Final Test Accuracy for batch size of 1024 – **0.6446**

Time



```
real    0m22.599s
user    1m8.567s
sys     0m1.587s
```





The model with a larger batch size (1024) performs better in terms of test accuracy and running time compared to the smaller batch size model (16).

There are some points important to highlight:

- Smaller batch sizes (16) require more frequent weight updates, resulting in longer training times per epoch due to more iterations per epoch.
- Larger batch sizes might allow for faster convergence since they provide more stable estimates of the gradient direction.

2. b)

python3 hw1-q2.py -epochs 20 -batch_size 16 -learning_rate 0.001 -hidden_size 200 -layers 2 -dropout 0.0 -activation relu -optimizer sgd logistic_regression

Final Test Accuracy for LR of 0.001 – **0.6503**

python3 hw1-q2.py -epochs 20 -batch_size 16 -learning_rate 0.01 -hidden_size 200 -layers 2 -dropout 0.0 -activation relu -optimizer sgd logistic_regression

Final Test Accuracy for LR of 0.01 – **0.6200**

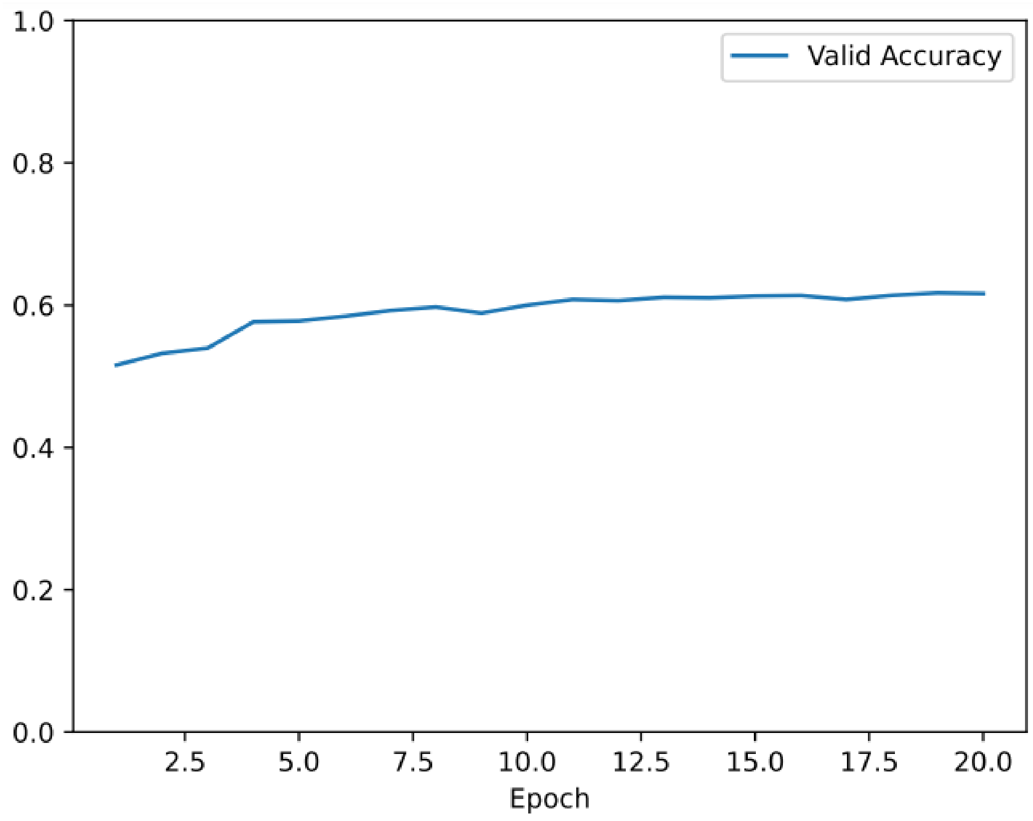
python3 hw1-q2.py -epochs 20 -batch_size 16 -learning_rate 0.1 -hidden_size 200 -layers 2 -dropout 0.0 -activation relu -optimizer sgd logistic_regression

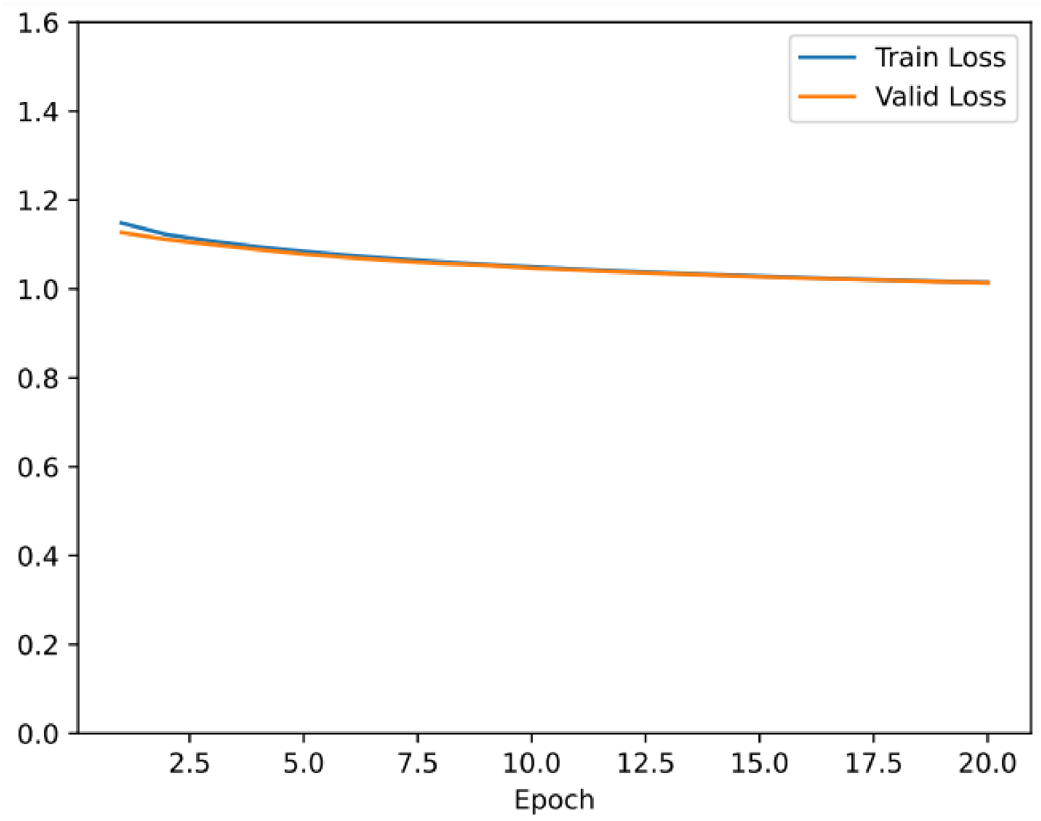
Final Test Accuracy for LR of 0.1 – **0.5577** *python3 hw1-q2.py -epochs 20 -batch_size 16 -learning_rate 1.0 -hidden_size 200 -layers 2 -dropout 0.0 -activation relu -optimizer sgd logistic_regression*

Final Test Accuracy for LR of 1.0 – **0.2930**

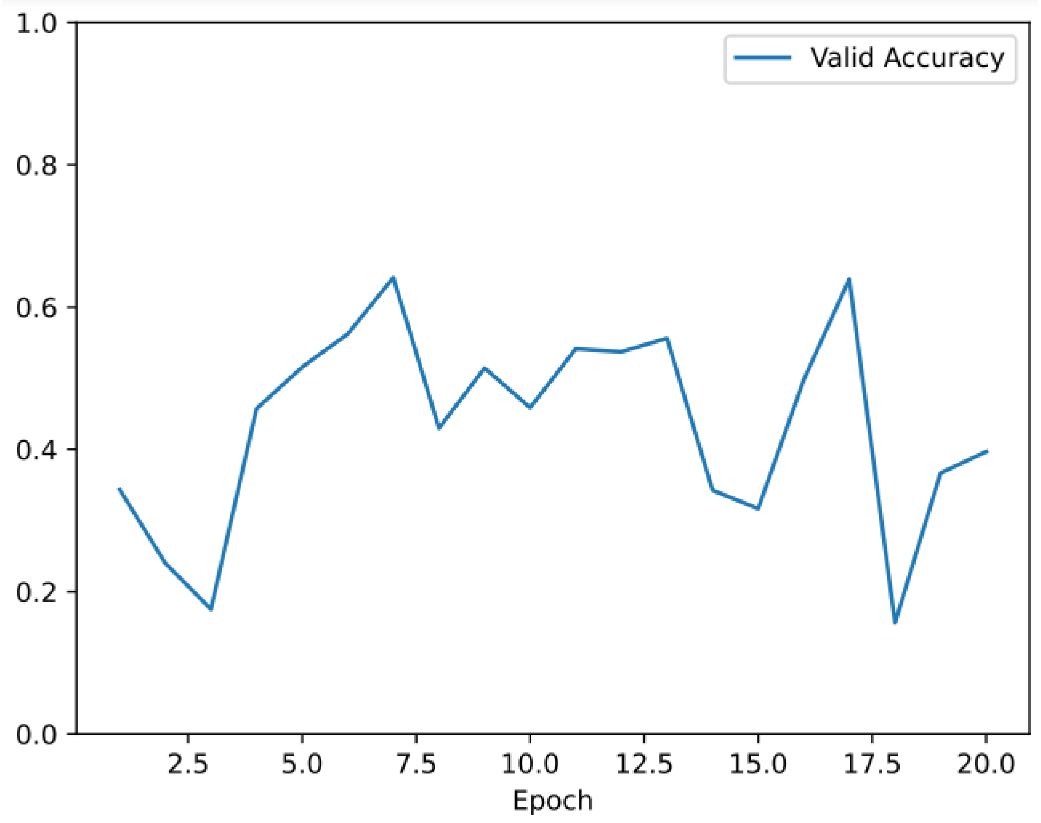
As we can observe above, the best test accuracy was **0.6503** for the lower learning rate (0.001). As the learning rate increased from 0.001 to 1.0, the test accuracy progressively decreased. A learning rate that is too high might cause overshooting of the minimum or diverge the optimization process. The selection of the learning rate is critical in neural networks training. A learning rate that's too low might result in slow convergence, while a learning rate that's too high can hinder convergence or cause instability in training.

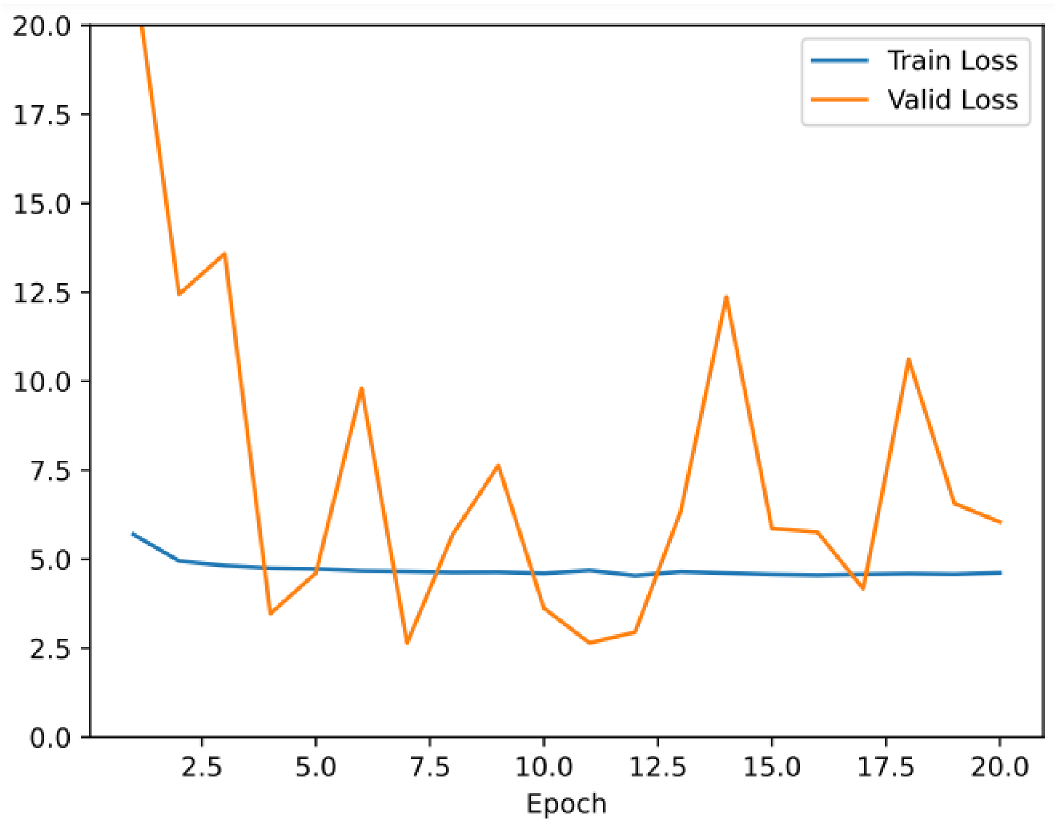
Plots for best configuration in terms of validation accuracy (LR of 0.0001):





Plots for worst configuration in terms of validation accuracy (LR of 1.0):





2. c)

- python3 hw1-q2.py -epochs 150 -batch_size 256 -learning_rate 0.1 -l2_decay 0.0 -hidden_size 200 -layers 2 -dropout 0.0 -activation relu -optimizer sgd logistic_regression

Final Test Accuracy for default values – **0.6181**

- python3 hw1-q2.py -epochs 150 -batch_size 256 -learning_rate 0.1 -l2_decay 0.0001 -hidden_size 200 -layers 2 -dropout 0.0 -activation relu -optimizer sgd logistic_regression

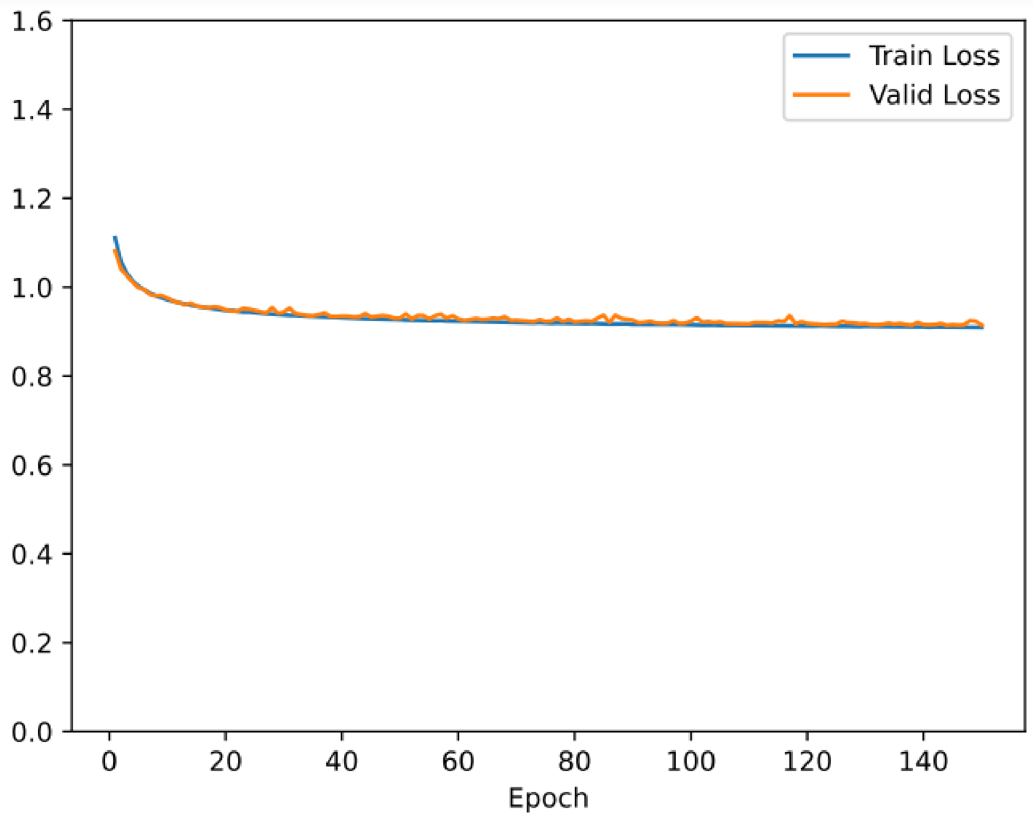
Final Test Accuracy for L2 Regularization of 0.0001 – **0.6181**

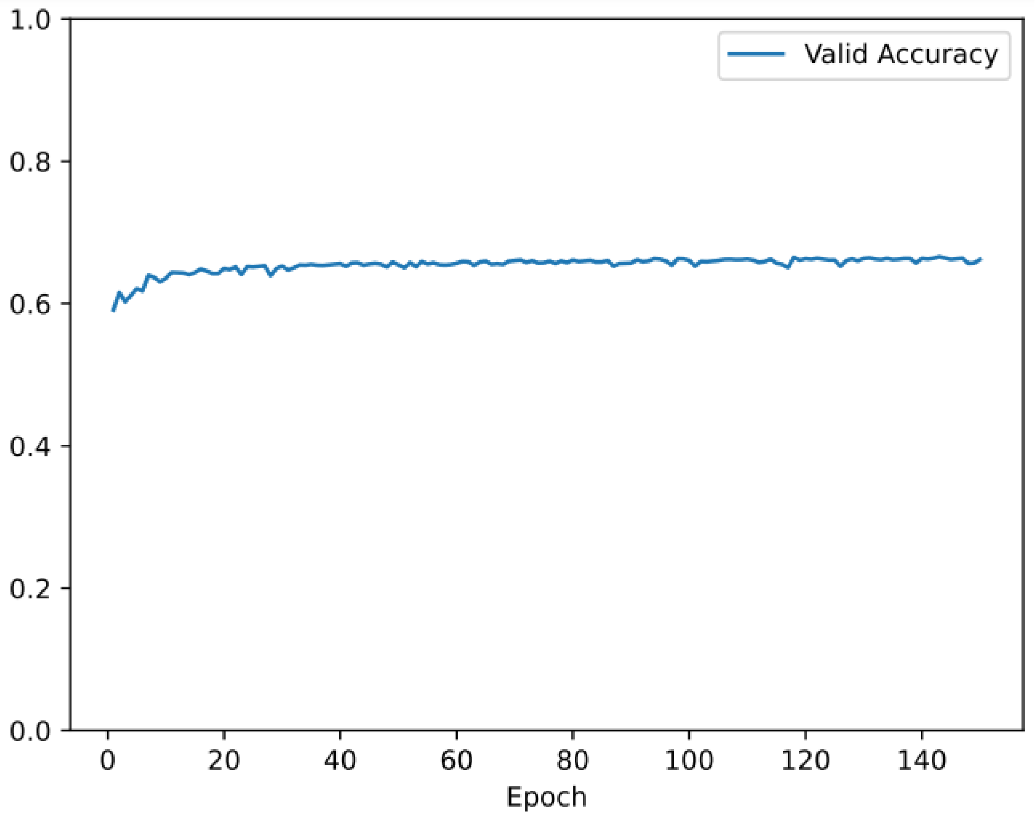
- python3 hw1-q2.py -epochs 150 -batch_size 256 -learning_rate 0.1 -hidden_size 200 -layers 2 -dropout 0.2 -activation relu -optimizer sgd logistic_regression

Final Test Accuracy for Dropout of 0.2 – **0.6181**

Across the different configurations, all models achieved identical final test accuracy. This observation suggests that the default model might not have been susceptible to overfitting. The application of regularization techniques such as L2 or Dropout did not alter the model's accuracy or significantly impact its ability to generalize. This observing implies that the default model was robust enough to maintain consistent performance across alterations, indicating a balanced capacity to learn and generalize from the data, likely without encountering overfitting issues.

The plots for each model are equal:





Question 3

1. a)

The function provided cannot be computed with a single perceptron and in the following lines we demonstrate it by a counterexample.

The function f(x) is defined based on the sum of input variables x_i in range [A, B]. So, we have D input variables, and we want to compute using a single perceptron.

The output is $\text{sign}\left(\sum_{i=1}^{D} w_i x_i + b\right)$

Now, we need to set weights w_i and bias b to the perceptron display correctly the output aimed. We will attribute some values to A, B and D and proof the counterexample.

$-D \leq A \leq B \leq D$

A = 1, B = 2 and D = 2

- If x_1 and x_2 are both +1 => output = 1 => **inside of range [A, B]**

- If x_1 and x_2 are both -1 => output = -1 => **outside of range [A, B]**

- If x_1 and x_2 are different (x_1 = +1 and x_2 = -1 or vice-versa) => output = -1 => **outside of range [A, B]**

As we can see we cannot set the weights and bias to compute f(x), so we conclude that the function cannot generally be computed with a single perceptron.

1. b)

$$h(x) = z1(z0(x + tv))$$

$$z_0 = \text{sign}(W_0 x + b_0)$$

$$z_1 = W_1 z_0 + b_1$$

$$W_0 = \begin{bmatrix} 1 & \cdots & 1 \\ -1 & \cdots & -1 \end{bmatrix}$$

$$b_0 = \begin{bmatrix} -A \\ B \end{bmatrix}$$

$$W_0 x + b_0 = \begin{bmatrix} \sum x - A \\ B - \sum x \end{bmatrix}$$

Se os valores forem positivos ou 0 (+1), as condições são cumpridas.

Se os valores forem negativos (-1), as condições não são cumpridas.

$$W_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$b_1 = [-1]$$

A função de ativação final é linear ou não existe.

1. c)

$$z_0 = \mathrm{relu}(W_0 x + b_0)$$

$$z_1 = g(W_1 z_0 + b_1)$$

$$W_0 = \begin{bmatrix} -1 & \cdots & -1 \\ 1 & \cdots & 1 \end{bmatrix}$$

$$b_0 = \begin{bmatrix} A \\ -B \end{bmatrix}$$

$$W_0 x + b_0 = \begin{bmatrix} -\sum x + A \\ -B + \sum x \end{bmatrix}$$

Se os valores forem negativos (0), as condições são cumpridas.

Se os valores forem positivos (+), as condições não são cumpridas.

$$W_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$b_1 = [0]$$

Função de ativação:

g=1, se z=0

g=-1, se z>0

This adaptation of the network structure with ReLU activations allows for increased flexibility in representing nonlinear functions compared to hard threshold activations. In fact, this neural network works similarly for binary or real numbers due to the presence of this ReLU activation function.