Deep Learning (IST, 2023-24)

Homework 1

André Martins, Francisco Melo, João Santinha, Duarte Alves, Margarida Campos

Deadline: Friday, December 15, 2023.

Please turn in the answers to the questions below in a PDF file, together with the code you implemented to solve them (when applicable).

IMPORTANT: Please write 1 paragraph indicating clearly what was the contribution of each member of the group in this project. A penalization of 10 points will be applied if this information is missing.

Please submit a single zip file in Fenix under your group's name.

Distribution

- Question 1
  - **1.a)** Guilherme
  - **1.b)** Guilherme
  - **2.a)** José
  - **2.b)** José
- Question 2
  - **1)** José
  - **2.a)** José
  - **2.b)** José
  - **2.c)** José
- Question 3
  - **1.a)** Guilherme
  - **1.b)** Guilherme
  - **1.c)** Guilherme

Question 1 (35 points)

Medical image classification with linear classifiers and neural networks. In this exercise, you will implement a linear classifier for a simple medical image classification problem, using the OCTMNIST dataset. The dataset contains 109,309 valid optical coherence tomography (OCT) images for retinal diseases comprised of 4 diagnosis categories - multi-class classification task. The original training set was split with a ratio of 9:1 into training and validation set, and use its source validation set as the test set. The source images are gray-scale with sizes varying between (384–1,536)×(277–512). Images were center-cropped with a window size of length of the short edge and resize them into 1×28×28. Please do not use any machine learning library such as scikit-learn or similar for this exercise; just plain linear algebra (the numpy library is fine). Python skeleton code is provided (hw1-q1.py).

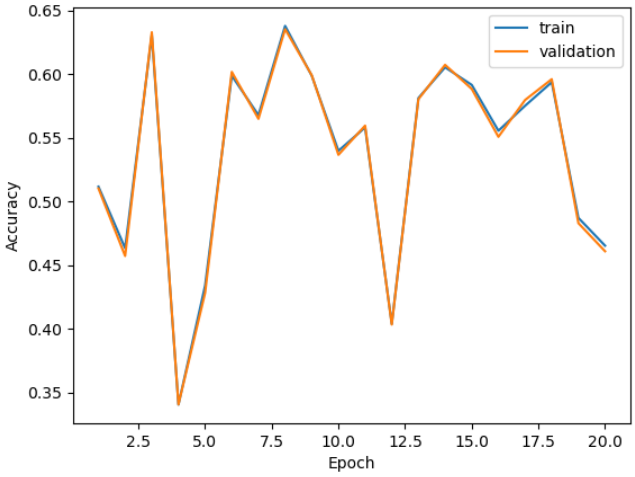
In order to complete this exercise, you will need to download the OCTMNIST dataset. You can do this by running the following command in the homework directory:

```
python download_octmnist.py
```

1. (a) (7 points) Implement the update_weights method of the Perceptron class in hw1-q1.py. Then train 20 epochs of the perceptron on the training set and report its performance on the training, validation and test sets. Plot the train and validation accuracies as a function of the epoch number. You can do this with the command

python hw1-q1.py perceptron

Solution:



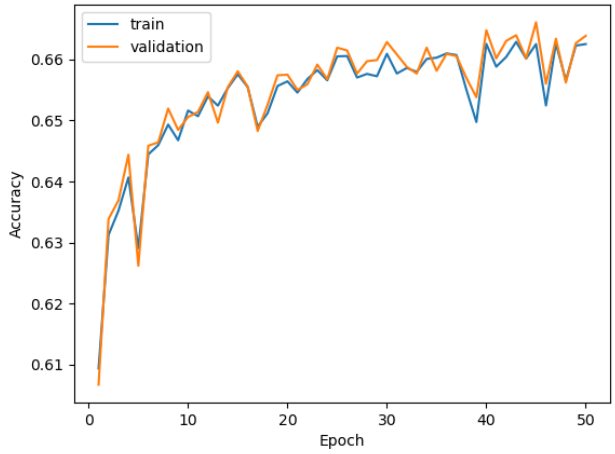
(b) (13 points) Repeat the same exercise using logistic regression instead (without regularization), using stochastic gradient descent as your training algorithm. Report the final test accuracies and compare, based on the plots of the train and validation accuracies, the models obtained using two different learning rates of $\eta = 0.01$ and $\eta = 0.001$. This can be solved by implementing the update_weights method in the LogisticRegression class. You can do this with the command.

1

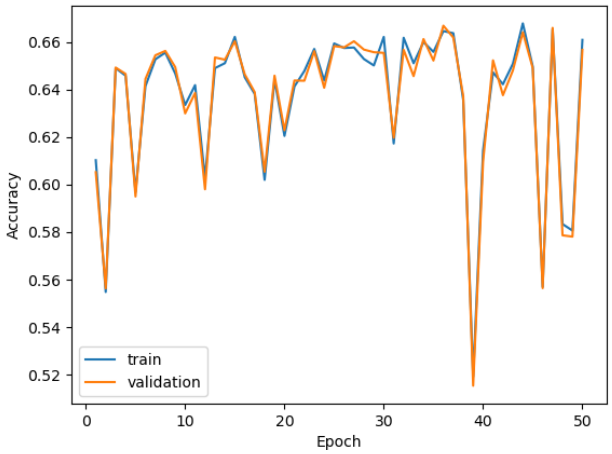
python hw1-q1.py logistic_regression -epochs 50 -learning_rate 0.01

Solution:

Learning rate: 0.001



Learning rate: 0.01



2. Now, you will implement a multi-layer perceptron (a feed-forward neural network) again using as input the original feature representation (i.e. simple independent pixel values).

   (a) (5 points) Comment the following claim: "A logistic regression model using pixel values as features is not as expressive as a multi-layer perceptron using relu activations. However, training a logistic regression model is easier because it is a convex optimization problem." Is this true of false? Justify.

   Solution:

   Logistic regression is a linear classifier, limited to linear decision boundaries, MLP on the opposite side can learn non-linear relationships between features. Which generally allows MLP to be more expressive than logistic regression.

   The relationship between pixels in an image can be very complex, so using pixel values as features will have a negative impact on the Logistic regression performance.

   However, training a MLP (using ReLU activation function) can be very resource intensive and can be very time consuming because this is not a convex problem. On the other hand logistic regression is convex problem which is easier to solve (since we don't have to deal with the local minima problem)

   In conclusion we consider this claim as True.

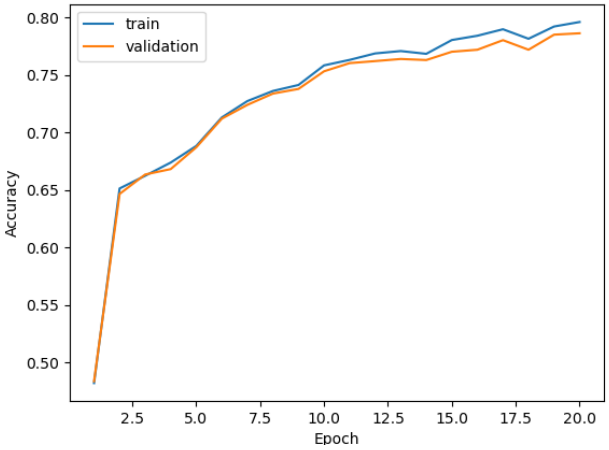
   (b) (10 points) Without using any neural network toolkit, implement a multi-layer perceptron with a single hidden layer to solve this problem, including the gradient backpropagation algorithm which is needed to train the model. Use 200 hidden units, a relu activation function for the hidden layers, and a multinomial logistic loss (also called cross-entropy) in the output layer (eventhough we are dealing with a binary classification this will allow you to use the same code for a multi-class problem). Don't forget to include bias terms in your hidden units. Train the model for 20 epochs with stochastic gradient descent with a learning rate of 0.001. Initialize biases with zero vectors and values in weight matrices with $w_{ij} \sim \mathrm{N}(\mu,\sigma^2)$ with $\mu = 0.1$ and $\sigma^2 = 0.1^2$

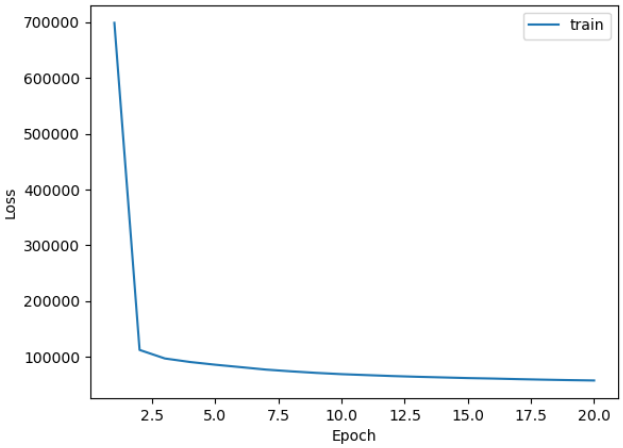
   (hint: use numpy.random.normal). Run your code with the base command, adding the necessary arguments

   python hw1-q1.py mlp

   Report final test accuracy and include the plots of the train loss and train and validation accuracies as a function of the epoch number.

   Solution: Final test accuracy - 0.7561





Question 2 (35 points)

Medical image classification with an autodiff toolkit. In the previous question, you had to write gradient backpropagation by hand. This time, you will implement the same system using a deep learning framework with automatic differentiation. Pytorch skeleton code is provided (hw1-q2.py) but if you feel more comfortable with a different framework, you are free to use it instead.

1. (10 points) Implement a linear model with logistic regression, using stochastic gradient descent as your training algorithm (use a batch size of 16). Train your model for 20 epochs and tune the learning rate on your validation data, using the following values: $\{0.001, 0.01, 0.1\}$.

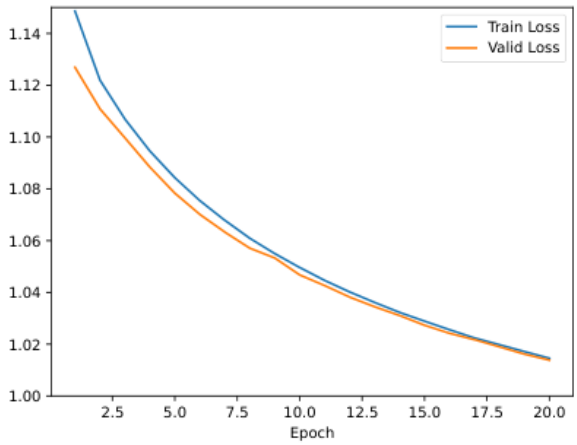
   Report the best configuration (in terms of final validation accuracy) and plot two things: the training loss and the validation accuracy, both as a function of the epoch number. Report the final accuracy on the test set.

   In the skeleton code, you will need to implement the method train_batch() and the class LogisticRegression's __init__() and forward() methods.

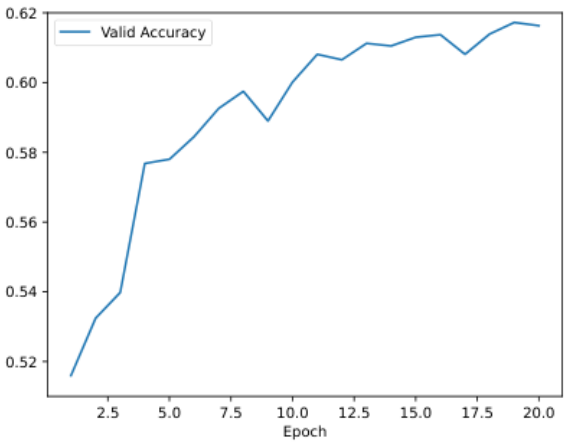
   Solution:

   Learning rate: 0.001 | Final test accuracy: 0.6503 | Final validation accuracy: 0.6163

   Loss

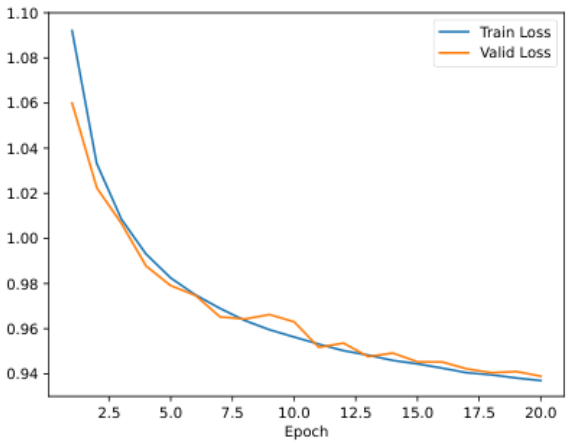
   

   Accuracy

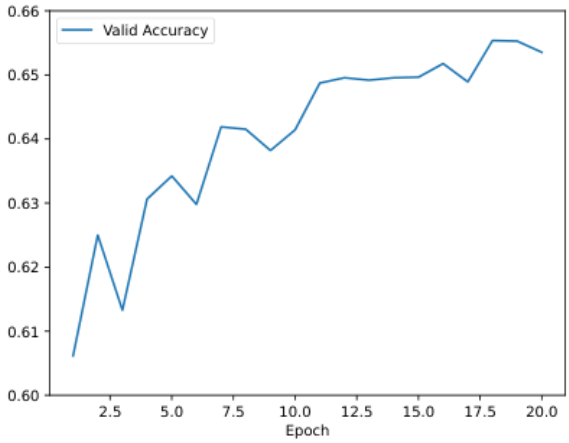
   

Learning rate: 0.01 | Final test accuracy: 0.6200 | Final validation accuracy: 0.6535

Loss

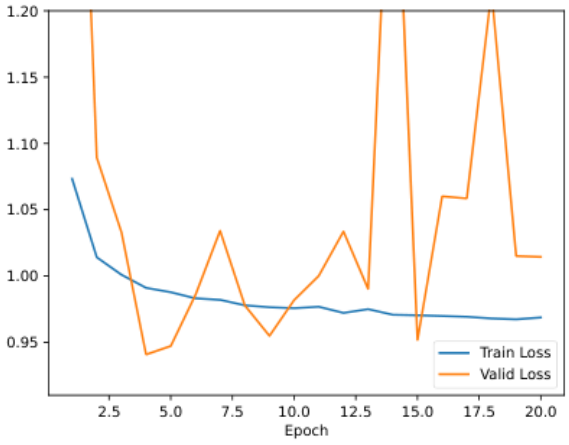


Accuracy

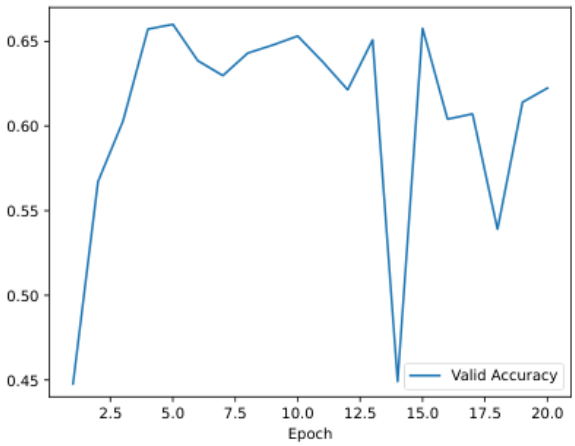


Learning rate: 0.1 | Final test accuracy: 0.5577 | Final validation accuracy: 0.6224

Loss



Accuracy



Conclusion:

The best configuration for 20 epochs is the learning rate of 0.01.

2. (25 points) Implement a feed-forward neural network using dropout regularization. Make sure to include all the hyperparameters and training/model design choices shown in Table 1. Use the values presented in the table as default.
   In the skeleton code, you will need to implement the class FeedforwardNetwork's __init__() and forward() methods.

   (a) (8 points) Compare the performance of your model with batch sizes 16 and 1024 with the remaining hyperparameters at their default value. Plot the train and validation losses for both, report the best test accuracy and comment on the differences in both performance and time of execution.

   You can time the execution of a program using time:

Number of Epochs	20
Learning Rate	0.1
Hidden Size	200
Number of Layers	2
Dropout	0.0
Batch Size	16
Activation	ReLU
L2 Regularization	0.0
Optimizer	SGD

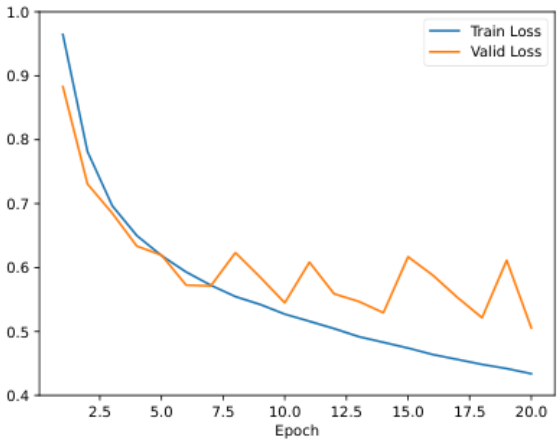
Table 1: Default hyperparameters.

time python hw1.py mlp

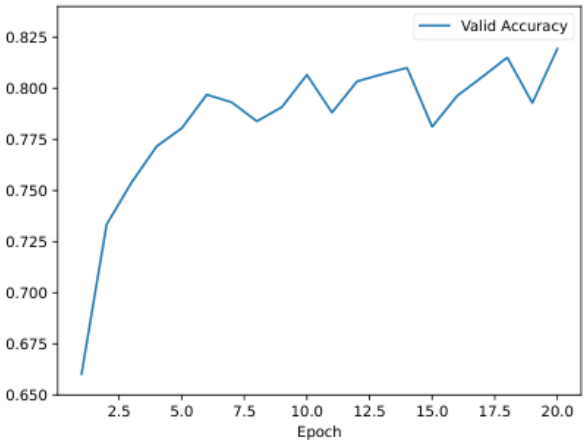
Solution:

Batch Size: 16 | Final Test Accuracy: 0.7713 | Final Valid Accuracy: 0.8193 | Time: 136.40 s

Loss

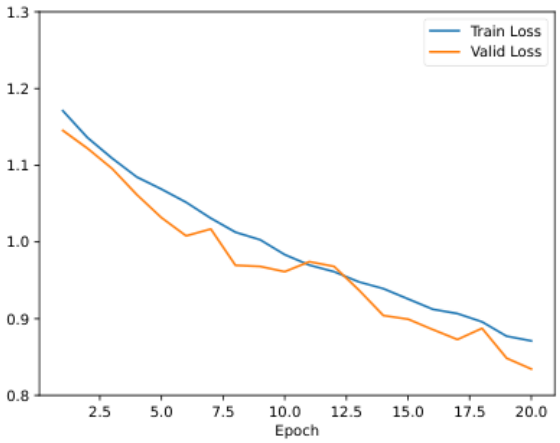


Accuracy

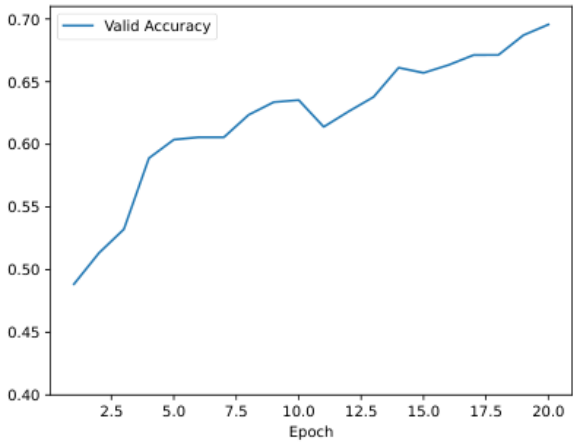


Batch Size: 1024 | Final Test Accuracy: 0.7278 | Final Valid Accuracy: 0.6956 | Time: 31.78 s

Loss



Accuracy



Conclusion:

Clearly the 1024 batch size execution time is much faster than the 16 one, but as all things this comes at a cost, and that cost is the accuracy.

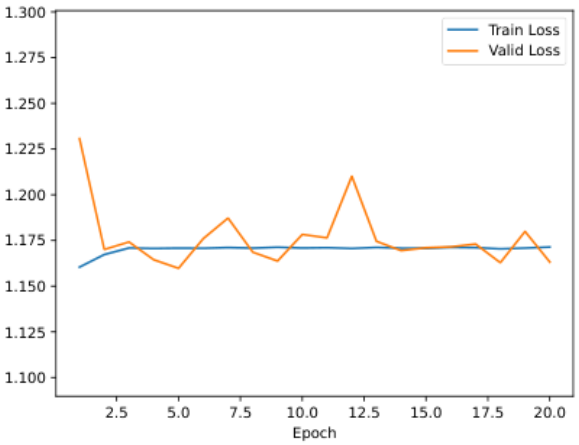
In summary, depending on the use case both are useful.

(b) (8 points) Train the model with learning rates: $1, 0.1, 0.01$ and 0.001 with the remaining hyperparameters at their default value. Plot the train and validation losses for the best and worst configurations in terms of validation accuracy, report the best test accuracy and comment on the differences in performance.

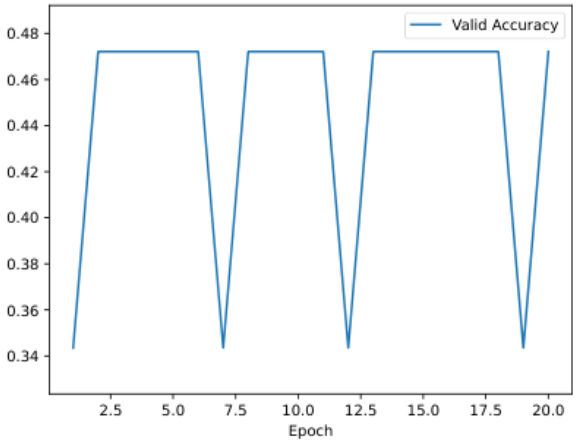
Solution:

Learning rate: 1 | Final Test Accuracy: 0.4726 | Final Valid Accuracy: 0.4721 | Time: 128.78 s

Loss

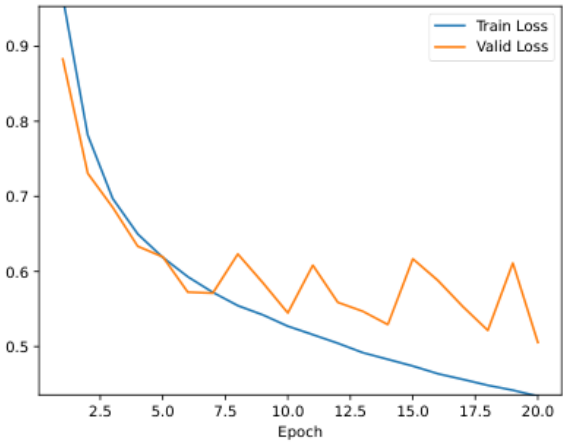


Accuracy

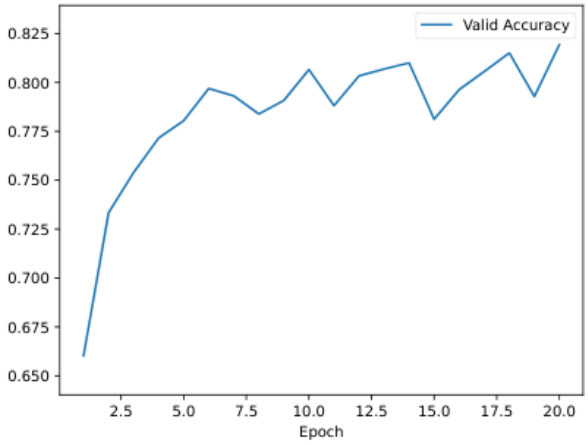


Learning rate: 0.1 | Final Test Accuracy: 0.7713 | Final Valid Accuracy: 0.8193 | Time: 129.28 s

Loss

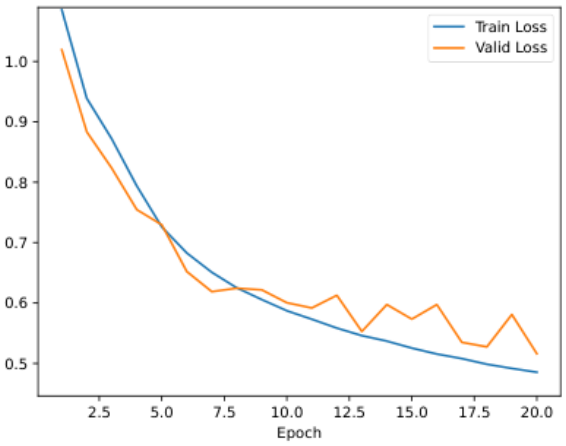


Accuracy

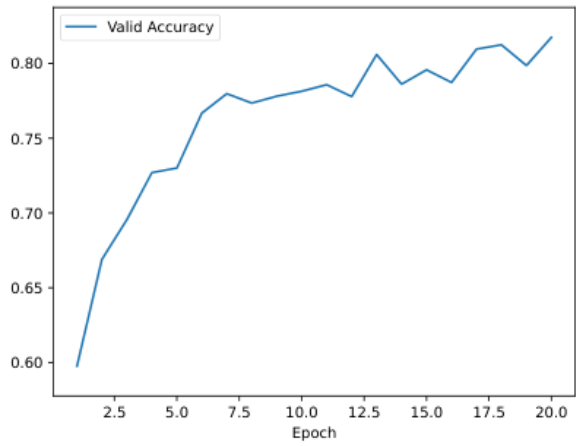


Learning rate: 0.01 | Final Test Accuracy: 0.7505 | Final Valid Accuracy: 0.8175 | Time: 135.65 s

Loss

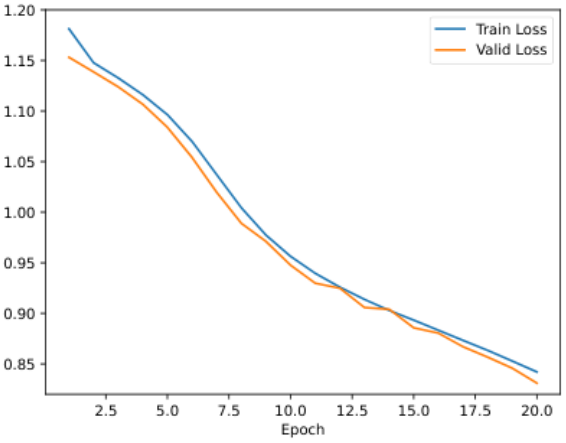


Accuracy

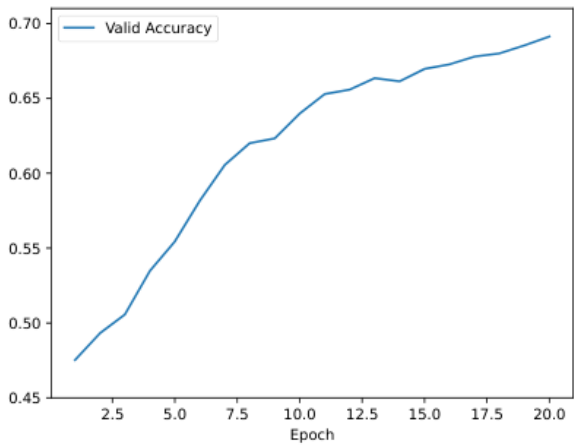


Learning rate: 0.001 | Final Test Accuracy: 0.7108 | Final Valid Accuracy: 0.6913|Time: 128.48 s

Loss



Accuracy



Conclusion:

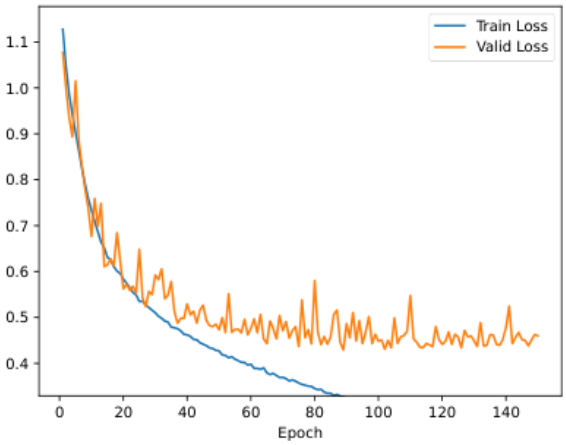
Learning rate of 0.1 has the best accuracy but it shows a great degree of variance, so it could just be that it is particularly good for this dataset, 0.01 shows the same variance with lower performance, and 0.001 has a stable accuracy increase but it can't reach such high accuracies with such low learning rate, the variation of performance in terms of time is negligible.

(c) (9 points) Using a batch size of 256 run the default model for 150 epochs. Is there overfitting? Train two similar models with the following changes: one with the L2 regularization parameter set to 0.0001 and the other with a dropout probability of 0.2. Plot the train and validation losses for the best and worst configuration in terms of validation accuracy, report the best test accuracy and comment on the differences of both techniques.

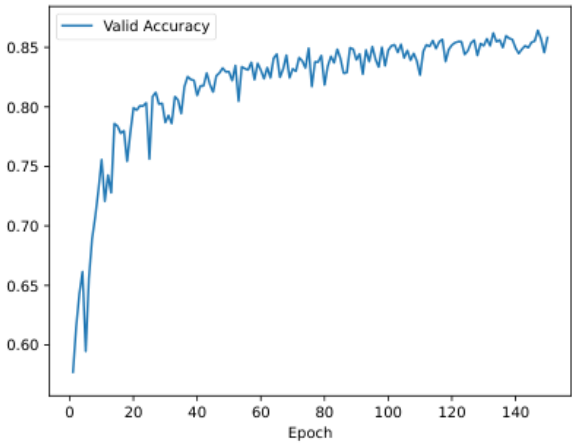
Solution:

Original | Final Test Accuracy: 0.7505| Final Valid Accuracy: 0.8582| Training Loss: 0.2175

Loss

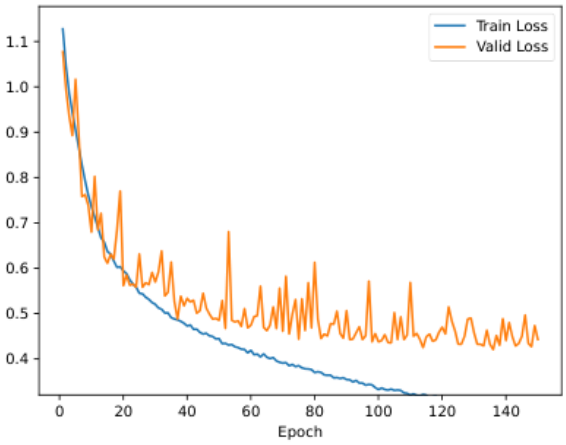


Accuracy

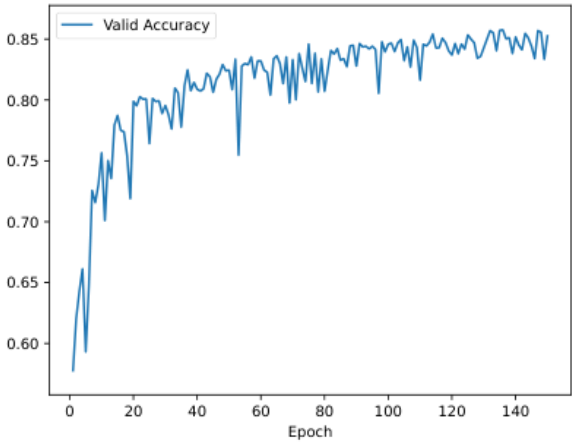


L2 0.0001| Final Test Accuracy: 0.7599 | Final Valid Accuracy: 0.8527| Training Loss: 0.2734

Loss

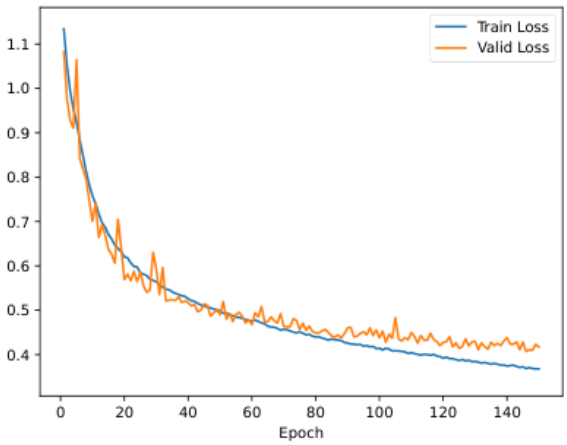


Accuracy

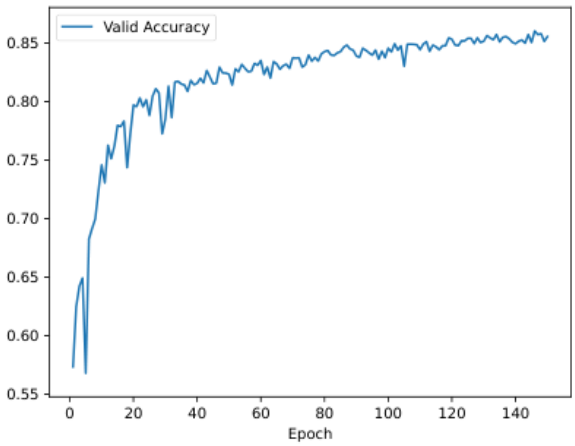


Dropout | Final Test Accuracy: 0.8034 | Final Valid Accuracy: 0.8554| Training Loss: 0.3677

Loss



Accuracy



Conclusion:

We can see the original overfit clearly by the discrepancy between accuracies, with L2 this barely improves, especially because the parameter is too small, and dropout greatly mitigates overfitting.

Question 3 (30 points)

1. In this exercise, you will design a multilayer perceptron to compute a Boolean function of D variables, $f: \{-1,+1\}^D \rightarrow \{-1,+1\}$, defined as:

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{D} x_i \in [A, B], \\ -1 & \text{otherwise,} \end{cases} \qquad (1)$$

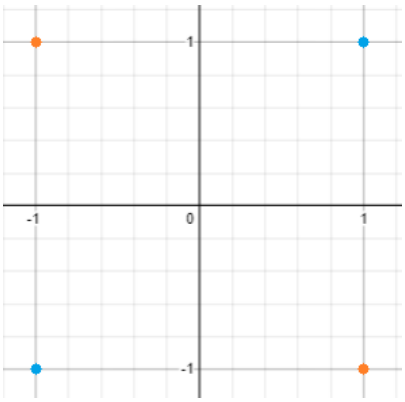
where A and B are integers such that $-D \leq A \leq B \leq D$.

(a) (5 points) Show that the function above cannot generally be computed with a single perceptron. *Hint: think of a simple counter-example.*

Solution:

D = 2 | A = -1 | B = 1

inputs	outputs
{-1, -1}	-1
{-1, 1}	1
{1, -1}	1
{1, 1}	-1



Conclusion:

This problem is not linearly separatable, just like XOR, so this function cannot generally be computed with a single perceptron.

(b) (15 points) Show that the function above can be computed with a multilayer perceptron with a single hidden layer with two hidden units and hard threshold activations g:R → {−1,+1} with

$$g(z) = \text{sign}(z) = \begin{cases} +1 & \text{if } z \geq 0, \\ -1 & \text{otherwise,} \end{cases} \quad (2)$$

and where all the weights and biases are integers (positive, negative, or zero). Provide all the weights and biases of such network, and ensure that the resulting network is robust to infinitesimal perturbation of the inputs, i.e., the resulting function h : R^D → R should be such that $\lim_{t\to 0} h(x+tv) = h(x) = f(x)$ for any $x \in \{-1,+1\}^D$ and $v \in R^D$.

Solution:

Layer units

Input	Hidden	Output
D	2	1

$X \rightarrow (D \times 1)$

- $W^{[1]} \rightarrow (2 \times D)$ matrix with 2 lines and D columns.

D = 2

$$W^{[1]} = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$$

D =3

$$W^{[1]} = \begin{pmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{pmatrix}$$

Meaning,

$$W^{[1]} = \begin{pmatrix} -- W_1^{[1]} -- \\ -- W_2^{[1]} -- \end{pmatrix}$$

$$W_1^{[1]} = ones_{[D]}$$

$$W_2^{[1]} = (-1) \times ones_{[D]}$$

- $b^{[1]} \rightarrow (2 \times 1)$

$$b^{[1]} = \begin{pmatrix} -A \\ B \end{pmatrix}$$ note: $-D \ll A \ll B \ll D$

- $W^{[2]} = (1 \quad 1)$
- $b^{[2]} = (-2)$

(c) (10 points) Repeat the previous exercise if the hidden units use rectified linear unit activations instead. (You will get partial credit if you solve for $D = 2$ and $A = B = 1$ only.)

Solution:

$W^{[1]} = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$

$b^{[1]} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$W^{[2]} = (1 \quad 1)$

$b^{[2]} = (-2)$

$\boldsymbol{D = 2 \quad A = B = 1}$

Proof:

$x1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$y1 = -1$

$z1 = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$h1 = \begin{pmatrix} relu(0) \\ relu(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$z2 = (1 \quad 1)\begin{pmatrix} 0 \\ 0 \end{pmatrix} + (-2) = -2$

$\hat{y} = sign(-2) = -1$

$x2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$

$y2 = -1$

$z1 = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} -1 \\ -1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$h1 = \begin{pmatrix} relu(0) \\ relu(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$z2 = (1 \quad 1)\begin{pmatrix} 0 \\ 0 \end{pmatrix} + (-2) = -2$

$\hat{y} = sign(-2) = -1$

$x3 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

$y3 = 1$

$z1 = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ -1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -2 \\ 2 \end{pmatrix}$

$h1 = \begin{pmatrix} relu(-2) \\ relu(2) \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$

$z2 = (1 \quad 1)\begin{pmatrix} 0 \\ 2 \end{pmatrix} + (-2) = 0$

$\hat{y} = sign(0) = 1$

$x4 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

$y4 = 1$

$z1 = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ -2 \end{pmatrix}$

$h1 = \begin{pmatrix} relu(2) \\ relu(-2) \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$

$z2 = (1 \quad 1)\begin{pmatrix} 2 \\ 0 \end{pmatrix} + (-2) = 0$

$\hat{y} = sign(0) = 1$