ECE324 Assignment 2

Aman Bhargava — 1005189733

Section 3: Solving by Inspection

1. $w_0 = 1; w_1 = -100; w_2 = 1; w_3 = -100; w_4 = 1; w_5 = -100; w_6 = 1; w_7 = -100; w_8 = 1; b = -4;$

2. No, the answer is not unique. Another set of parameters that would yield 100% accuracy would be: $w_0 = 1; w_1 = -100; w_2 = 1; w_3 = -100; w_4 = 1; w_5 = -100; w_6 = 1; w_7 = -100; w_8 = 1; b = -4.001;$

3. For each of the 9 elements in the grid, there are two possibilities: 0 or 1. Therefore, the total number of possible configurations is 2^9.

4. *Assuming that 'my' solution is the one from question 1* — If the 'X' type pattern is a 3×3 X-shape somewhere in the grid, then yes, it does. One would simply need to apply the algorithm on all 4 of the possible sub-grids and observe if any of them contain an 'X' (rather like in a convolutional neural network). If the 'X' pattern is a 4×4 'X', then the approach would have to be slightly tweaked depending on what the chosen definition for a 4×4 'X' is. If there are many possible configurations of a 4×4 'X', the solution may not scale perfectly.

5. It would be more difficult to determine a single neuron algorithm to solve this problem, especially if the entire 5×5 grid must be used as the input of the single-neuron classifier. In that case, it would be objectively impossible because confounding examples would get in the way of the model performance, and the linearity of the single neuron classifier would be its downfall. The linear combinations of a non-shifted 'X' and an 'X' that is shifted to the left and to the right (upwards?) by one unit includes non-'X' shapes, so a single neuron classifier of the same exact architecture presented in this assignment would certainly be unable to work properly on this problem.

Section 4: Training from Data

No questions were asked for this section.

Section 5: Guidance

No quesitons were asked for this section.

Section 6: Experiments and Outputs to Hand In

Effect of Number of Epochs

```
=== PART 6.1.1: Effect of the Number of Epochs ===

REQUESTED:
* "Tabular form" for data that shows the effect of hyperparameters on accuracy.
* Hyper parameter: NUMBER OF EPOCHS
* Must "select and report reasonable values for other parameters.

EXECUTION:
* We select values of `num_epochs` = [1, 5, 10, 100, 250, 500, 1000, 2000]
* Reasonable other values:
    * alpha:            0.05
    * active. func.:    ReLU
    * random seed:      120
```

=== EFFECT OF NUMBER OF EPOCHS ON TRAINING AND VALIDATION ACCURACY ===

	Epoch	Train Acc	Valid Acc
0	1	-0.825	-0.85
1	5	0.205	0.10
2	10	0.375	0.35
3	100	0.975	1.00
4	250	0.985	1.00
5	500	0.985	1.00
6	1000	0.990	1.00
7	2000	1.000	1.00

Effect of Learning Rate

```
=== PART 6.1.2: Effect of Learning Rate ===

REQUESTED:
* "Tabular form" for data that shows the effect of hyperparameters on accuracy.
* Hyper parameter: ALPHA
* Must "select and report reasonable values for other parameters.

EXECUTION:
* We select values of `alpha` = [.0001, .001, .01, .02, .05, .1, .25, .5, 1, 2]
* Reasonable other values:
    * epochs:           250
    * active. func.:    ReLU
    * random seed:      160
```

```
=== EFFECT OF LEARNING RATE ON TRAINING AND VALIDATION ACCURACY ===
```

	LR	Train Acc	Valid Acc
0	0.0001	-1.235	-1.15
1	0.0010	0.495	0.55
2	0.0100	0.960	0.95
3	0.0200	0.980	0.95
4	0.0500	0.985	0.95
5	0.1000	0.990	1.00
6	0.2500	0.665	0.65
7	0.5000	0.665	0.65
8	1.0000	0.665	0.65
9	2.0000	0.665	0.65
10	10.0000	0.665	0.65
11	20.0000	0.665	0.65
12	50.0000	0.665	0.65
13	100.0000	0.665	0.65
14	250.0000	0.665	0.65
15	500.0000	0.665	0.65

Effect of Activation Function

```
=== PART 6.1.3: Effect of Activation Functions ===

REQUESTED:
* "Tabular form" for data that shows the effect of hyperparameters on accuracy.
* Hyper parameter: ACTIVATION FUNCTION
* Must "select and report reasonable values for other parameters.

EXECUTION:
* We select values of ACTIVATION FUNCTION \in {Linear, ReLU, Sigmoid}
* Reasonable other values:
    * epochs:       500
    * alpha:        0.05
    * random seed:  201
```

```
=== EFFECT OF LEARNING RATE ON TRAINING AND VALIDATION ACCURACY ===
```

	Activation Func	Train Acc	Valid Acc
0	Linear	0.970	0.90
1	ReLU	0.990	1.00
2	Sigmoid	0.975	0.95

- ReLU likely succeeded most because it generally tends to converge fastest for a variety of reasons (especially relative to Sigmoid).
- The sigmoid function's gradient is extinguished as it gets far away from 0 in both directions. Even when it is close to zero, the gradient is rather small. Meanwhile, the gradient of ReLU is extinguished only on the negative side and it is equal to 1 for all positive numbers.
- The Linear activation function is somewhat capped in terms of how effective it can be relative to the other functions as the non-linearity introduced by the other activation functions increase their ability to represent more complex decision boundaries.

Effect of Random Seed

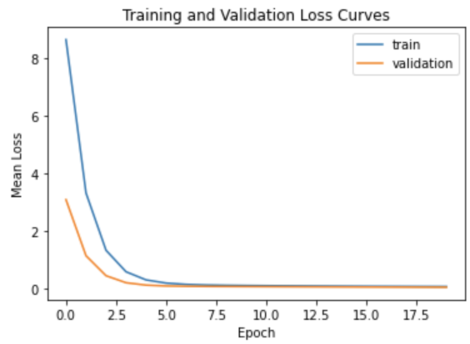
```
=== PART 6.1.4: Effect of 5 Random Seeds ===

REQUESTED:
* "Tabular form" for data that shows the effect of hyperparameters on accuracy.
* Hyper parameter: RANDOM SEED
* Must "select and report reasonable values for other parameters.

EXECUTION:
* We select values of `random_seed` = [0, 4, 2, 3, 9]
* Reasonable other values:
    * epochs:           250
    * activ. func.:     ReLU
    * alpha:            0.005
```

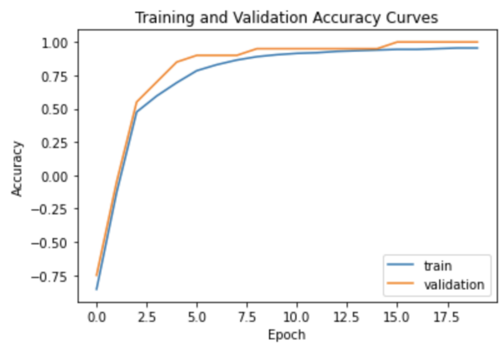
	Rand Seed	Train Acc	Valid Acc
0	0	0.890	0.85
1	4	0.925	0.80
2	2	0.905	0.95
3	3	0.860	0.85
4	9	0.950	0.90

- The answers differ here because there are multiple weights combinations that yield low loss (i.e. there exist local optima) in the loss function.
- Different random seeds lead to the algorithm reaching different local optima for its parameters.
- Moreover, minor differences in weights can yield errors in classifying one, two, or even three of the 20 validation examples. This could be responsible for what is observed in this case.

Best Possible Results in Fewest Epocs

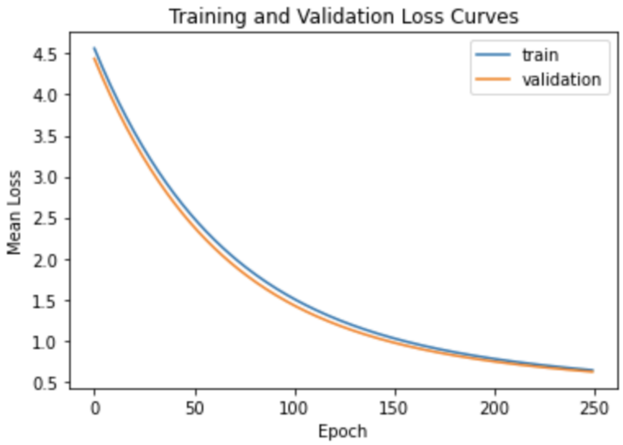
```
Activation Function:  ReLU
Learning Rate:    0.05
Number of Epochs:   20
Random Seed:    180
=================================
Final Training Accuracy:  0.955
Final Validation Accuracy:  1.0
Final Training Loss (avg):  0.07115583418703729
Final Validation Loss (avg):  0.04900431083861176
Parameters: Weights = [ 0.11070601 -0.21661655  0.17559251 -0.11500243  0.36745238  0.08907574
 -0.0258889  -0.24192523  0.25366264]
Parameters: Bias = -0.15218629422247648
```

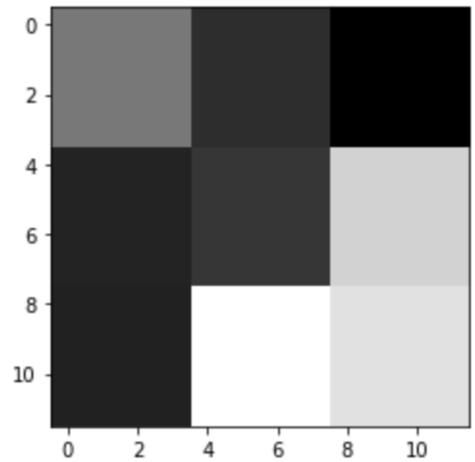


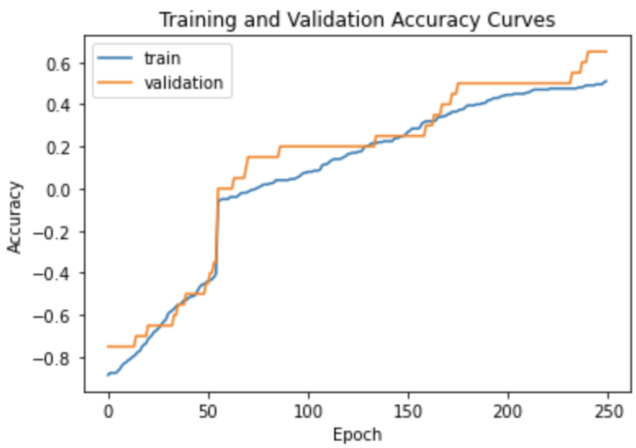


Learning Rate is Too Low

```
Activation Function:  ReLU
Learning Rate:    0.001
Number of Epochs:   250
Random Seed:    191
=================================
Final Training Accuracy:  0.51
Final Validation Accuracy:  0.65
Final Training Loss (avg):  0.6385017247671487
Final Validation Loss (avg):  0.6190566322665776
```



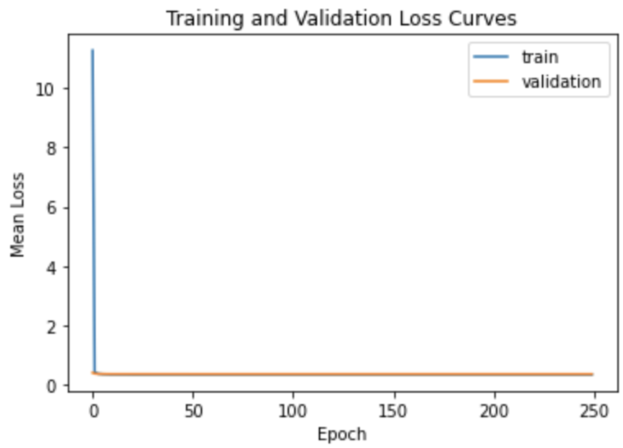


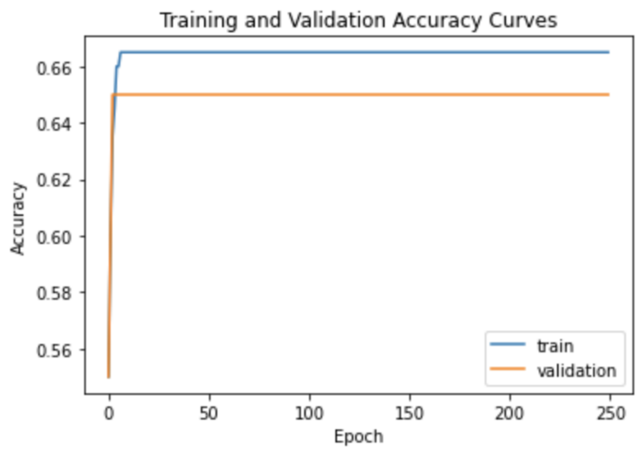


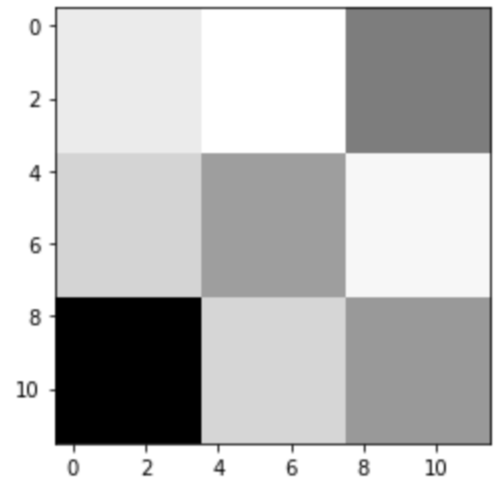
- The functions' parameters are approaching a (local) optimum for their values, but they are moving too slowly in each epoch to reach their in a reasonable amount of time.
- Therefore, the model yields sub-par parameters due to its low learning rate.

Learning Rate is Too high

```
Activation Function:  ReLU
Learning Rate:    0.15
Number of Epochs:   250
Random Seed:    194
=================================
Final Training Accuracy:  0.665
Final Validation Accuracy:  0.65
Final Training Loss (avg):  0.3350002232205514
Final Validation Loss (avg):  0.35
```



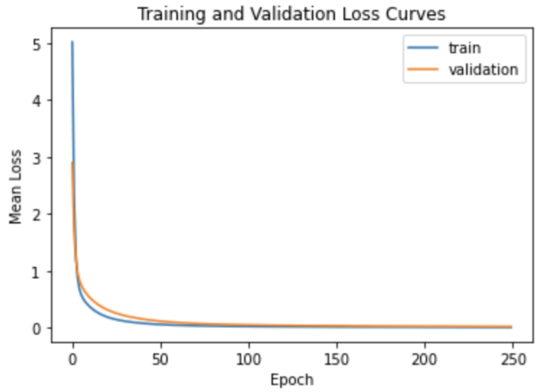


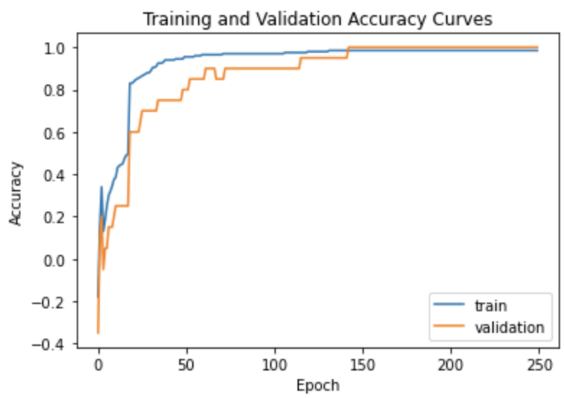


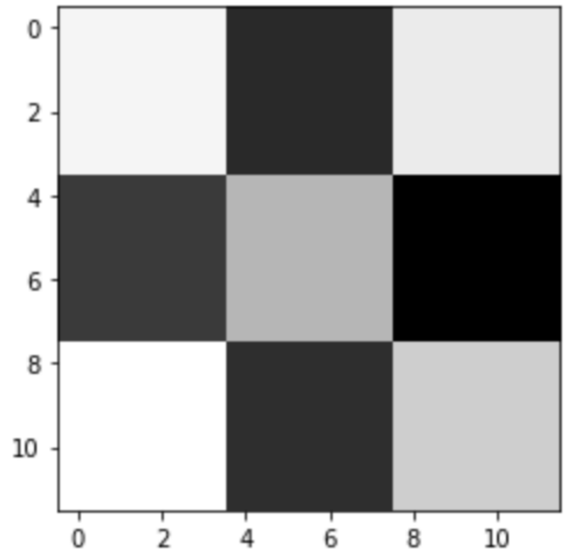
- The parameters' value is being adjusted too far each time a step is taken in the gradient descent process.
- Instead of optimizing towards progressively better values, the algorithm overshoots the nearby local optima.
- Therefore, performance suffers and the algorithm does not reach good values.

Learning Rate is Just Right

```
Activation Function:  ReLU
Learning Rate:    0.05
Number of Epochs:   250
Random Seed:    100
=================================
Final Training Accuracy:  0.985
Final Validation Accuracy:  1.0
Final Training Loss (avg):  0.01214694664125074
Final Validation Loss (avg):  0.024843129918858668
```





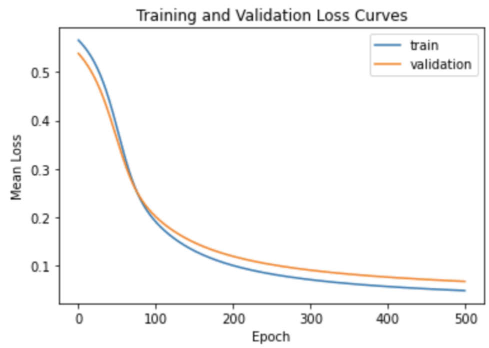


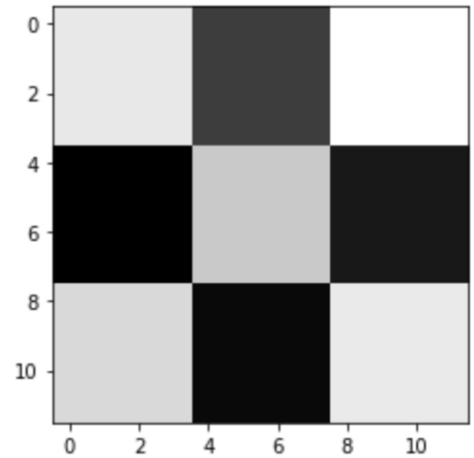
- *No explanation requested for this question.*

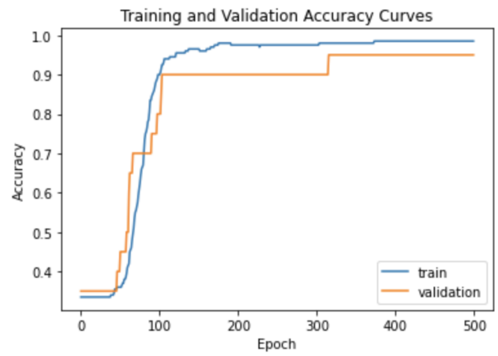
Accuracy and Loss Curves for Three Activation Functions

Sigmoid Function

```
Activation Function:  Sigmoid
Learning Rate:    0.1
Number of Epochs:   500
Random Seed:    23
=================================
Final Training Accuracy:  0.985
Final Validation Accuracy:  0.95
Final Training Loss (avg):  0.04909988204611978
Final Validation Loss (avg):  0.06807577527881666
```

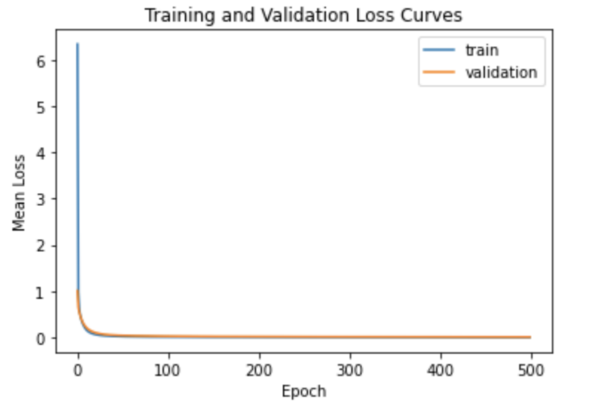


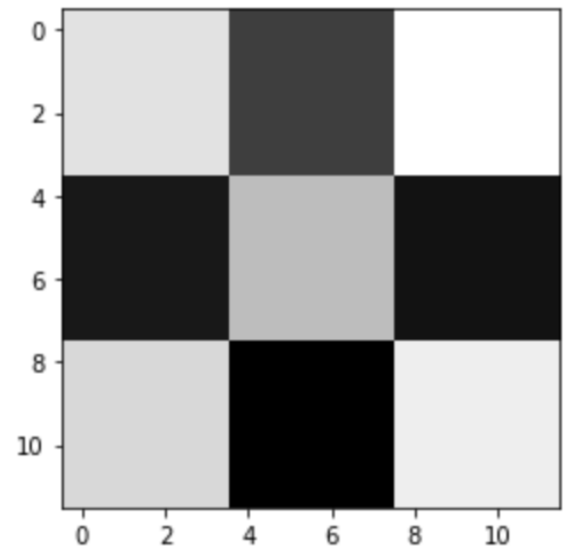


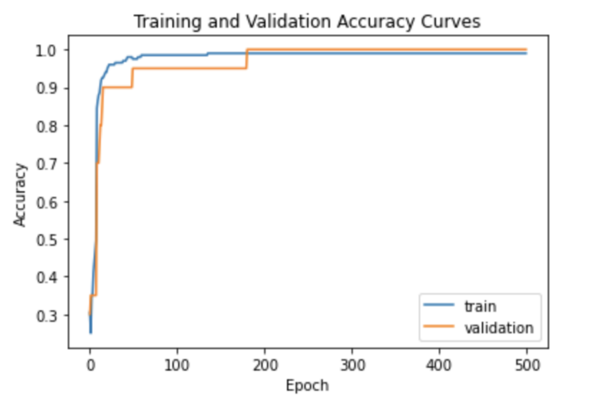


ReLU Function

```
Activation Function:  ReLU
Learning Rate:    0.1
Number of Epochs:   500
Random Seed:    23
=================================
Final Training Accuracy:  0.99
Final Validation Accuracy:  1.0
Final Training Loss (avg):  0.004589387106594914
Final Validation Loss (avg):  0.010586305277758861
```

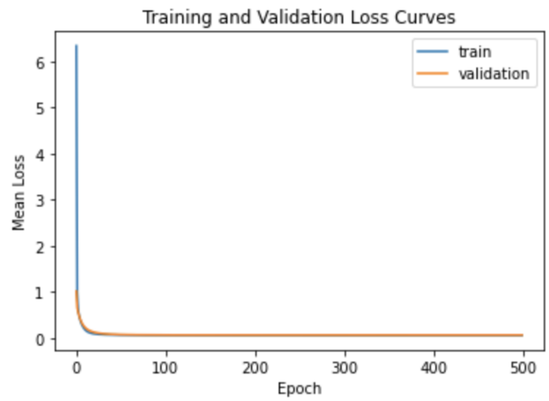


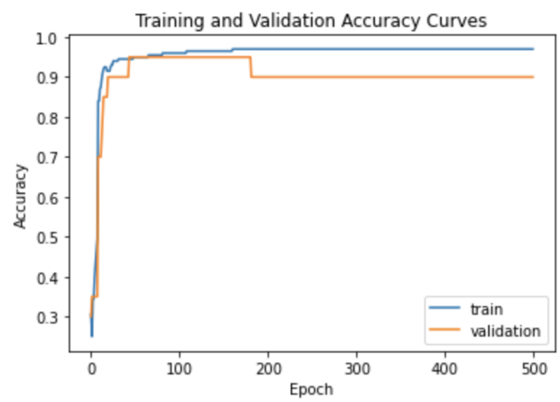


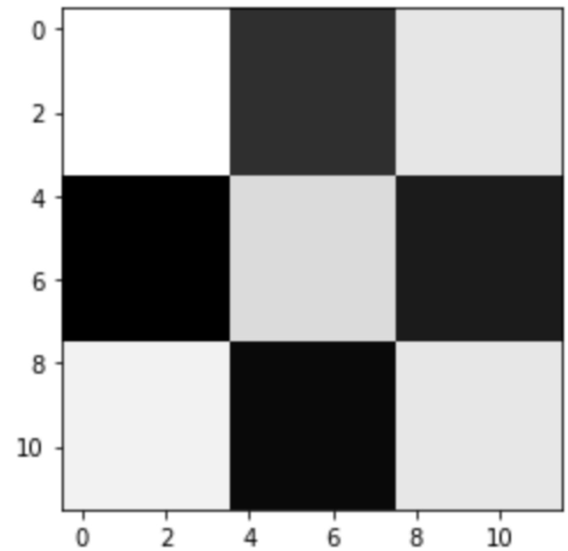


Linear Activation Function

```
Activation Function:  Linear
Learning Rate:    0.1
Number of Epochs:   500
Random Seed:    23
=================================
Final Training Accuracy:  0.97
Final Validation Accuracy:  0.9
Final Training Loss (avg):  0.05562814639898504
Final Validation Loss (avg):  0.06089841871042249
```







Section 7: Re-Implementation in PyTorch

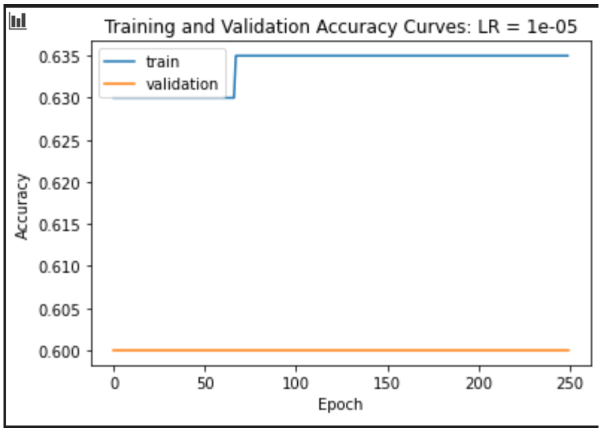
1. In my implementation, the **object** that holds the weights and biases for the neural net is `fc1`. It is of class `torch.nn.Linear`.

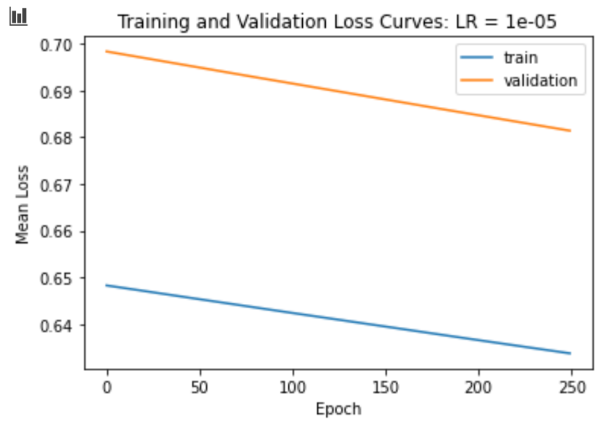
2. The name of the tensor object that contains the gradients of the weights (and bias) is `torch.nn.parameter.Parameter`. It **has an** instance of `torch.Tensor` called 'grad' that stores the gradient.

3. The line, `loss.backward()`, computes the gradient. This line must trigger a change in all the `torch.nn.parameter.Parameter` objects that were involved in producing the loss function. That change involves adding the value for the 'grad' component (a `torch.Tensor`) that belongs to each parameter. PyTorch 'knows' it must do this because it tracks all of the computations that were performed on and with objects of class `torch.nn.parameter.Parameter`.

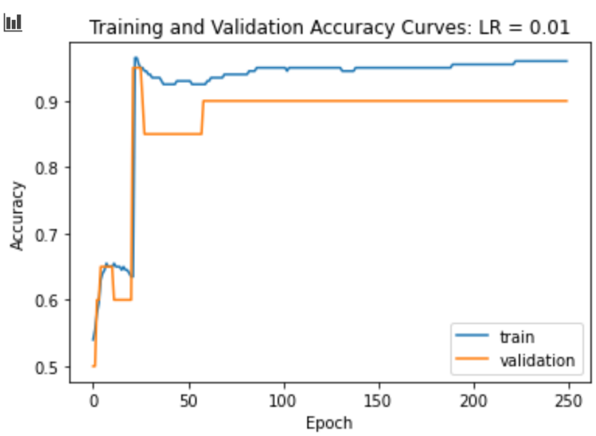
4: Training/validation loss plots, training/validation accuracy plots for 3 learning rate cases:

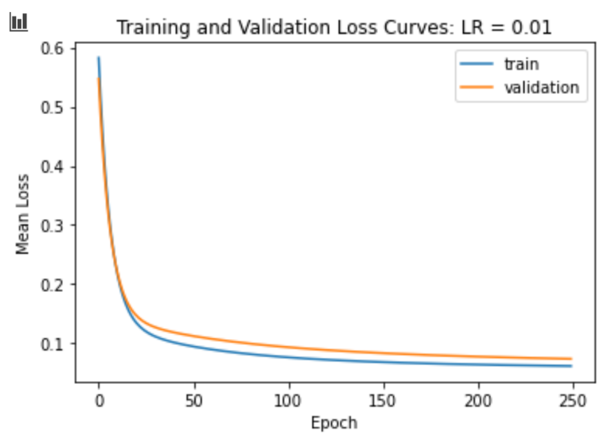
Too Slow



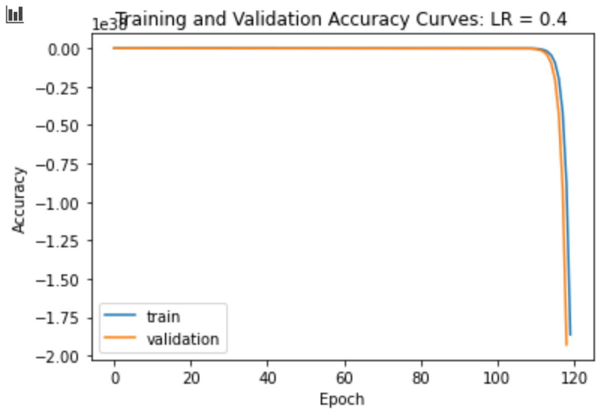


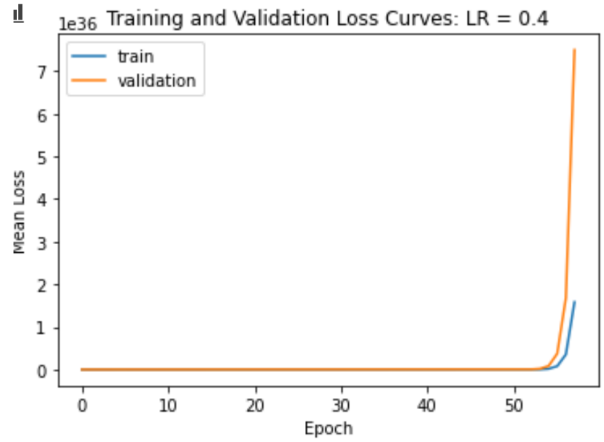
Correct Learning Rate





Too High Learning Rate


Training and Validation Accuracy Curves: LR = 0.4
1e38
Accuracy
Epoch
train
validation


Training and Validation Loss Curves: LR = 0.4
1e36
Mean Loss
Epoch
train
validation