Project Report

Effects of hyperparameters on training and validation accuracy

1. Effects of Epochs on Training and Validation Accuracy

- Learning rate: 0.01
- Activation function: Linear
- Random Seed: 0

	Epochs	Training Accuracy	Validation Accuracy
Trial 1	50	0.395	0.30
Trial 2	100	0.785	0.70
Trial 3	500	0.945	0.95
Trial 4	1000	0.97	0.95
Trial 5	2000	0.98	0.95

2. Effects of Learning Rate on Training and Validation Accuracy

- Epochs: 1000
- Activation function: Linear
- Random Seed: 0

	Learning Rate	Training Accuracy	Validation Accuracy
Trial 1	1	0.665	0.65
Trial 2	0.1	0.98	0.90
Trial 3	0.05	0.98	0.90
Trial 4	0.01	0.97	0.95
Trial 5	0.005	0.945	0.95
Trial 6	0.001	0.78	0.7

3. Effects of Activation Function on Training and Validation Accuracy

- Epochs: 1000
- Learning Rate: 0.01
- Random Seed: 0

	Activation Function	Training Accuracy	Validation Accuracy

Trial 1	Linear	0.97	0.95
Trial 2	ReLU	0.99	0.95
Trial 3	Sigmoid	0.945	0.85

Explanations: As shown in the table, ReLU activation function yields the best training and validation accuracy. This is because the ReLU function does not activate all the neurons at the same time. If the input value is less than or equal to 0, the ReLU function will recognize it as 0 and will not update the weights and bias.

4. Effects of Random Seed on Training and Validation Accuracy

- Epochs: 1000
- Learning Rate: 0.01
- Activation Function: Linear

	Random Seed	Training Accuracy	Validation Accuracy
Trial 1	0	0.97	0.95
Trial 2	21	0.96	0.90
Trial 3	999	0.98	0.90
Trial 4	124	0.97	0.90
Trial 5	642	0.975	0.95

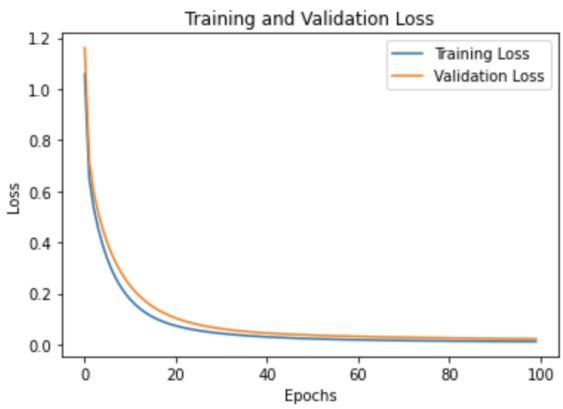
Explanations: The random seed essentially sets the initial starting point of the optimization problem. In other words, different random seeds will yield different loss functions, some have a lot of saddle points, which take longer for the gradient descent to get down to the global minima, yielding a lower accuracy; some are smoother in shape and would not require much efforts for the algorithm to get to the global minima, hence yielding a higher accuracy.

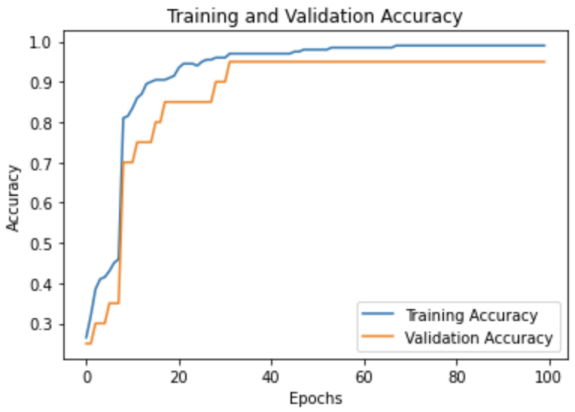
Ideal Case

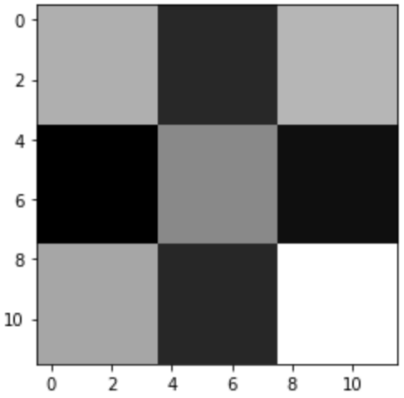
- Epochs: 100
- Learning rate: 0.1
- Activation function: ReLU
- Random seed: 0

Training Accuracy: 0.99

Validation Accuracy: 0.95



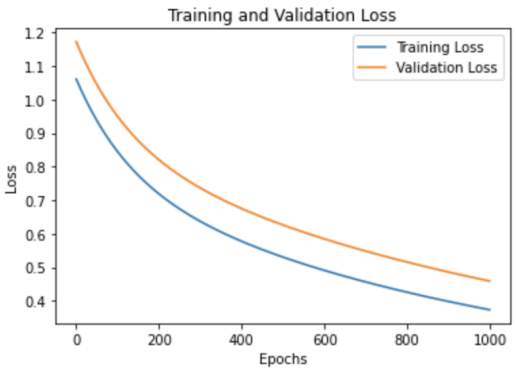


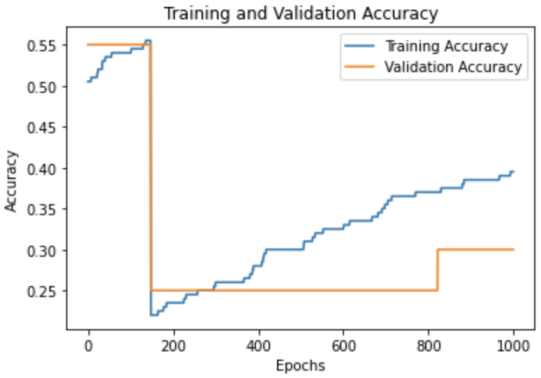


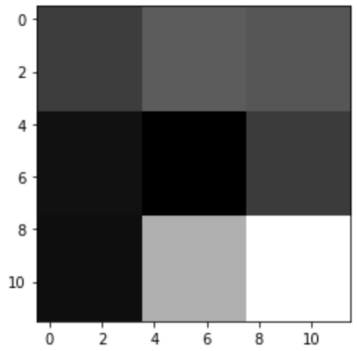
Graphs of Different Cases

1. Training rate is too low

- Epochs: 1000
- Learning rate: 0.005
- Activation Function: Linear
- Random seed: 0



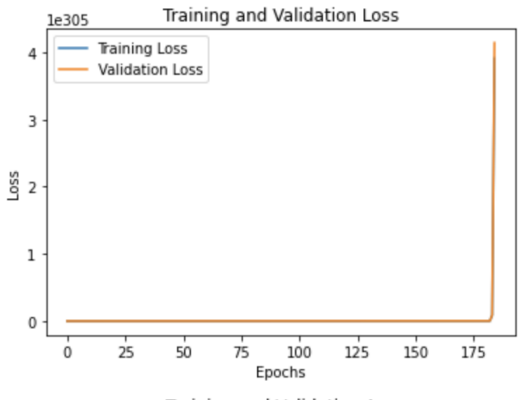


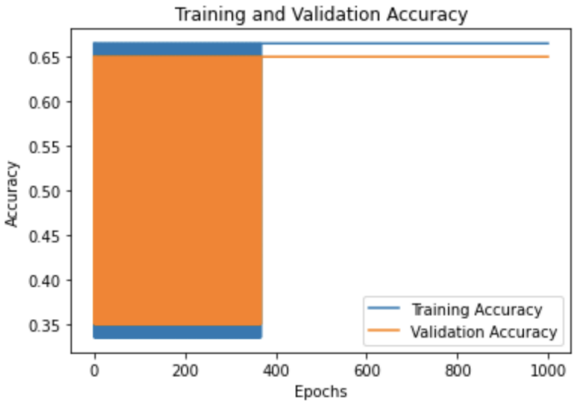


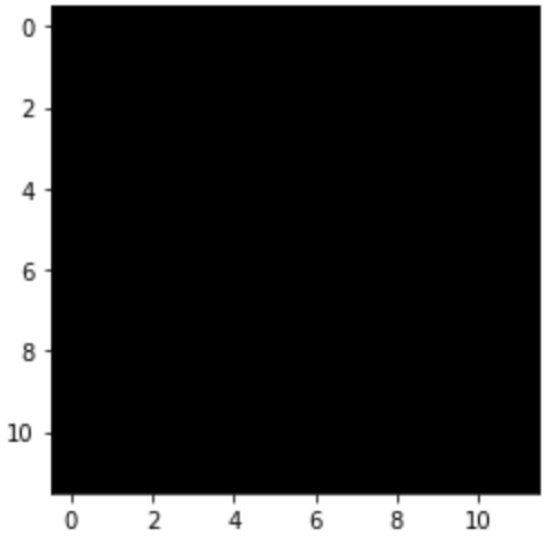
<u>Explanations:</u> The low training rate indicates a very small adjustment in the weights and bias for every epoch. As can be seen from the training and validation loss graph, even though the curves are smooth, the loss decrease is very small. In terms of accuracy, because the loss function has not reached its minimum point, the accuracy is also increasing very slowly.

2. Learning rate is too high

- Epochs: 1000
- Learning rate: 1
- Activation function: Linear
- Random Seed: 0



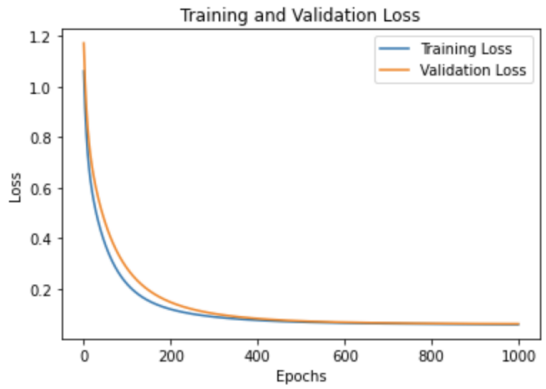


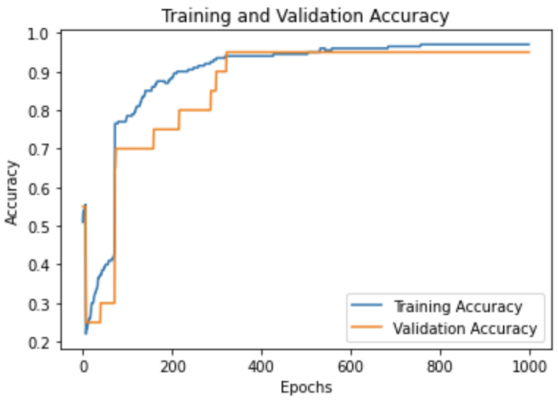


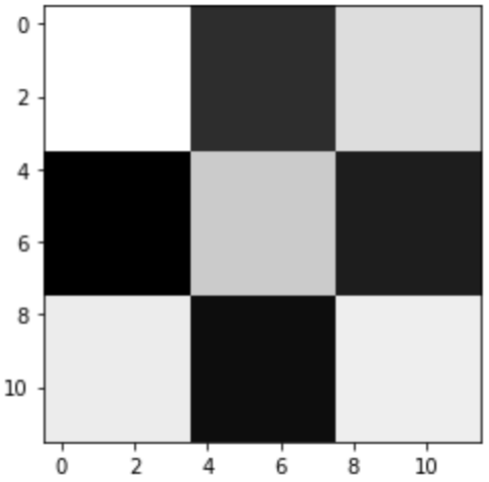
<u>Explanations:</u> As can be observed in the 2 graphs, the learning rate is so high that it causes a ***divergence in the loss function****, causing the loss to approach infinity. This can also be visualized in the dispkernal graph as the graph is completely black.*

3. A 'good' learning rate

- Epochs: 1000
- Learning rate: 0.01
- Activation function: Linear
- Random seed: 0



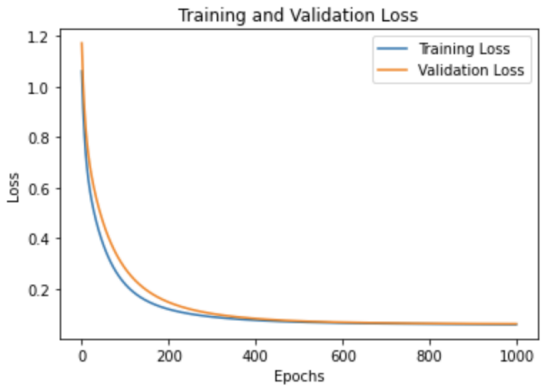


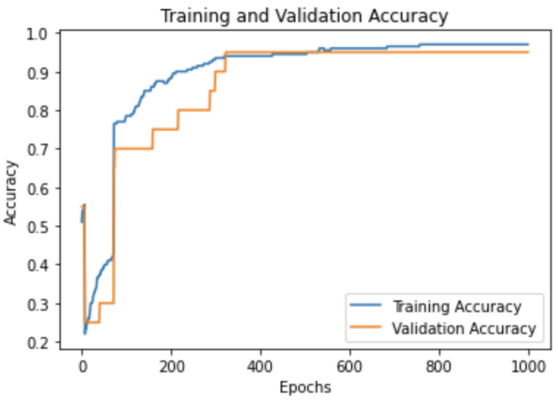


4. Different activation functions

a) Linear

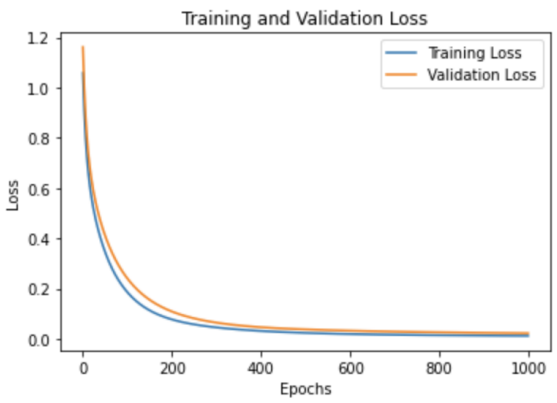
- Epochs: 1000
- Learning rate: 0.01
- Random seed: 0

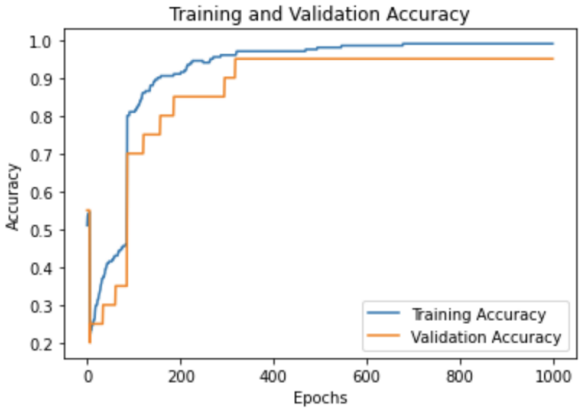




b) ReLU

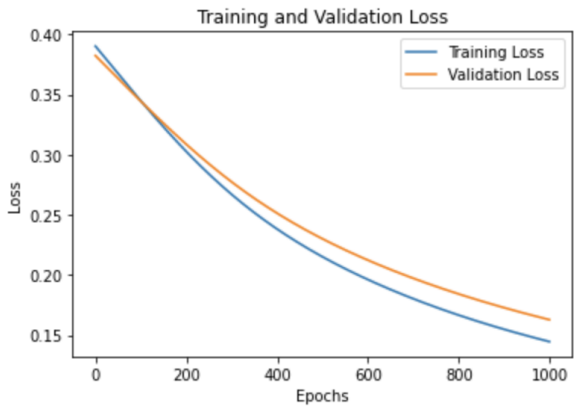
- Epochs: 1000
- Learning rate: 0.01
- Random seed: 0

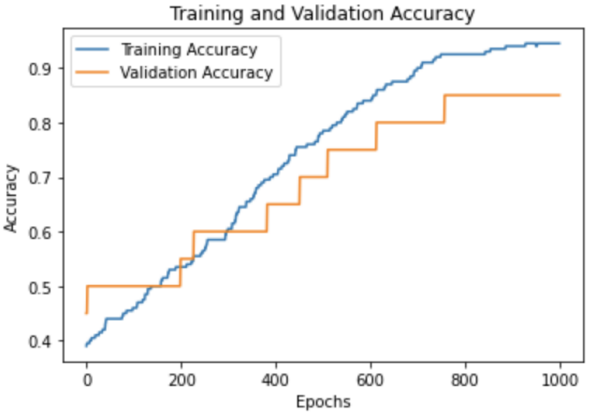




c) Sigmoid

- Epochs: 1000
- Learning rate: 0.01
- Random seed: 0





Part III. PyTorch implementation of SNC

1. 'self.fc1.weight' and 'self.fc1.bias' are the 2 tensor objects storing the weights and bias.

2. 'loss' contains the calculated gradients of the loss function.

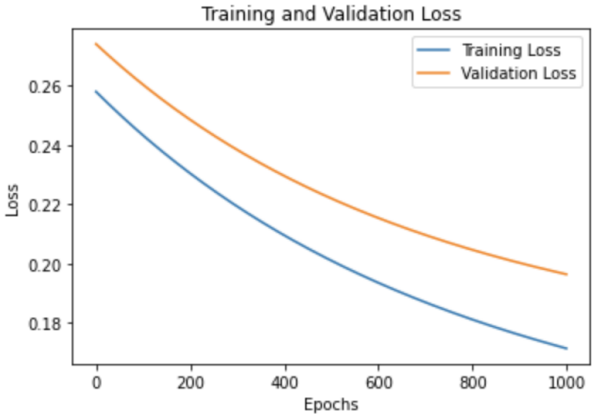
3. Line of code that computes gradients: 'loss.backward()'; this calls for 'tensors', 'grad_tensors', 'create_graph' and an important boolean value 'retain_graph', which must be

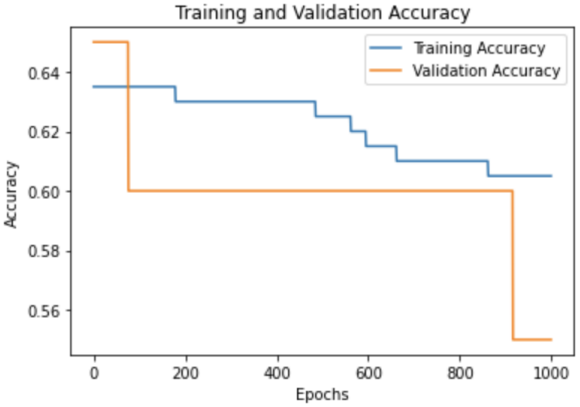
True in order for the function to know that the computed gradients for the previous epochs have been stored and that the function can have access to those previously computed gradients for further calculations.

4. Graphs of different cases

a) Learning rate too slow

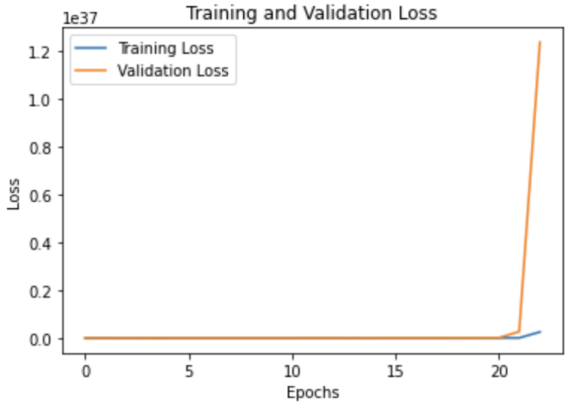
- Epochs: 1000
- Learning rate: 0.0001
- Activation function: Linear
- Random seed: 0

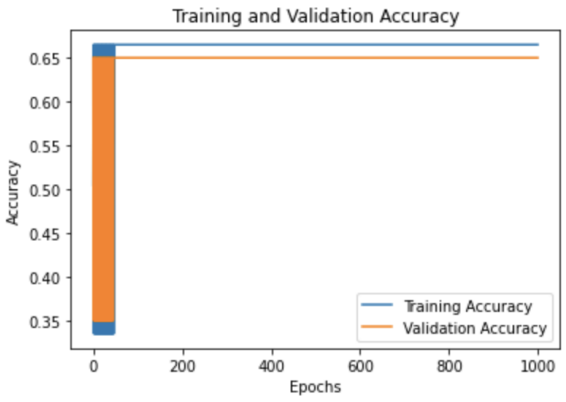




b) Learning rate too fast

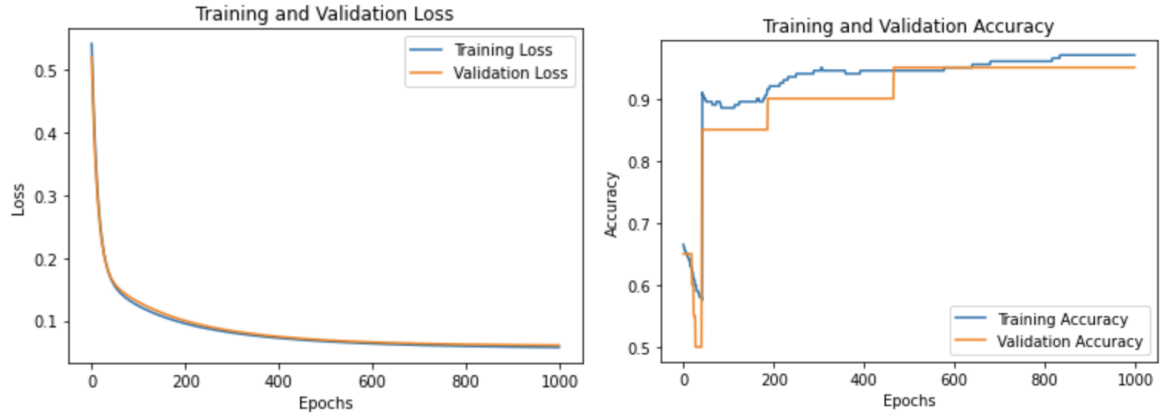
- Epochs: 1000
- Learning rate: 1
- Activation function: Linear
- Random seed: 0





c) Learning rate 'just-right'

- Epochs: 1000
- Learning rate: 0.005
- Activation function: Linear
- Random seed: 0



**Interesting finding: A learning rate that is 'just-right' and produces good training and validation accuracy in PyTorch is considered to be 'too slow' in manual implementation.*