CMPT310 - A4 Report
Maxwell Libbrecht
Justin Liu – 301070053

toy_example_regressor.py Output:
[Epoch 950]: loss: 0.01760831514529372
Validation Loss 0.0119637193613

prime_classifier.py Output
[Epoch 49]: validation loss: 0.02454812, validation accuracy: 97.08%

layers.py
compute_activation(self,x)

- Activation is determined by the formula: dot product(input,weights) + bias. Should the output after compute_activation >= some predefined threshold, then this neuron is considered live (up to programmer to determine the live vs not live output)

compute_gradient(self)

- Self.dw = dot product of change in error with respect to change in output and change in output with respect to change in weight
- Self.db = dot product of change in error with respect to change in output and change in output with respect to change in bias
- Self.dx = dot product of change in error with respect to change in output and change in output with respect to change in input error gradient

update_weights(self,learning_rate):

- Learning rate limits the change in weights due to error of outputs. If the difference in one output and the expected output is x, we want to make a proportional change (learning rate) toward x without overcompensating for the error

neural_network.py
compute_activation(self,x)

- x is the set of initial inputs. We pass x into the first layer, which produces output which is then used as input to the next layer.
- Loss is the error between the last layer's output vs expected output

compute_gradient(self)

- We have the loss in the last layer (difference between actual output and expected output), we now go through each layer (starting from last) and calculate the errors between each layer and set the error in outputs of the previous layer

update_weights(self,learning_rate):

- Weights are updated after each iteration given the proportional error distribution by weights and the predefined error rate. This is done to each layer as all weights need to be adjusted by their proportional responsibility.