Report

In compute\_activation of layers.py file, we compute the input data with weight matrix using dot product and add bias. In compute\_gradient, we calculate dw (derivative of weight matrix)by doing dot product of input\_data and output\_error\_gradient, calculate db (derivative of bias matrix) by applying dot product of output\_error\_gradient and matrix of 1 with row equals column of output\_error\_gradient and column equals 1, and lastly calculate input\_error\_gradient by using dot product of output\_error\_gradient and weight matrix. In update\_weight, we update weight and bias by taking original matrix of weight and bias minus its derivative multiply by learning\_rate respectively.

In neural\_network.py file, to compute output we first compute\_activation value of x for the first layer and use the result of it for the next layer recursively. Loss is genereated by using compute\_acivation with output got from above and given target as 2 inputs. For compute\_gradient, first we have to compute\_gradient of loss and use that result to set output error gradient for each layer in reversed order. Computing gradient in that layer and pass it to previous layer recursively until all layers are finished. In update\_weight, we update\_weight for each layer using given learning\_rate as well as update\_weight for loss.

As we have trained the model toy\_example\_regressor.py, we obtain validation loss equals to 0.0112853… After we trained the model prime\_classifier.py, we obtain validation loss of 0.02454448 and validation accuracy percentage of 97.12.

A screenshot of a cell phone

Description automatically generated