Implementing Neural Networks:

Let us consider a neural network with one hidden layer shown in Figure 1. We have a dataset that has 128 input features for our input layer $x = [x_1, x_2, ..., x_{128}]^T$, these input features determine a digit from 0 to 9, and the hidden layer has 10 nodes $z = [z_1, z_2, ... z_{10}]^T$ and the output layer is a probability distribution $y = [y, y_2, .., y_{10}]^T$ over 10 classes [0,1,2,3,4,5,6,7,8,9]. We also have a bias to the input, $x_0 = 1$, and the hidden layer $z_0 = 1$, both of which are fixed to 1.

α is a matrix of weights from the inputs to the hidden layer and β is the matrix of weights from the hidden layer to the output layer. We use a sigmoid activation function for the hidden layer and a softmax for the output layer.

The input: x = [x1, x2, ……. x128]T

Linear combination at the first hidden layer:

X_train_s = array([[$x_{1,1}$, $x_{1,2}$, $x_{1,3}$, …, 0, 0, $x_{1,128}$],
[$x_{2,1}$, $x_{2,2}$, 0, …, 0, 0, $x_{2,128}$],
[0 , 0 , 0 , . . . , 0 , 0 , 0],
………… ….,
[0 , 0 , 1 , . . . , 1 , 1 , 0] ,
[0 , 0 , 0 , . . . , 0 , 0 , 0] ,
[$x_{500,1}$, 0, 0, …, 0, 0, $x_{500,128}$]])

We add an extra column for bias and it will be all ones as we initialized before $x_0 = 1$:

tr_x_new (500, 129) = array([[$x_{1,0}$, $x_{1,1}$, $x_{1,2}$, $x_{1,3}$, …, 0, 0, $x_{1,128}$],
[$x_{1,0}$, $x_{2,1}$, $x_{2,2}$, 0, …, 0, 0, $x_{2,128}$],
[1, 0 , 0 , . . . , 0 , 0 , 0],
………… ….,
[1 , 0 , 1 , . . . , 1 , 1 , 0] ,
[1 , 0 , 0 , . . . , 0 , 0 , 0] ,
[$x_{500,0}$, $x_{500,1}$, 0, 0, …, 0, 0, $x_{500,128}$]])

Now we have our weights to initialize:

α is the weights matrix at the first hidden layer, β is the weights matrix at the second hidden layer

α = [[$\alpha_{1,0}$ $\alpha_{1,1}$ … 0.04879827 0.02608369 $\alpha_{1,128}$]
[1. -0.02715854 0.02290215 … 0.04079655 0.06501024 0.0421194]
[1. 0.03455442 0.05347734 … 0.00971958 -0.09487136 -0.08562756]
….
[1. 0.05250957 -0.01526351 … -0.08653053 0.06871794. -0.03633839]
[1. 0.03341941 0.06333114 … -0.04539489 0.02465743 -0.09124861]
[$\alpha_{10,0}$ -$\alpha_{10,1}$ 0.06813219. … -0.00680107 -0.0598454 $\alpha_{10,128}$]]

$\beta = [\ [\ \beta_{1,0}\quad \beta_{1,1}\qquad 7.39569433e\text{-}02\ \ldots\ldots\ldots\ldots\ \beta_{1,11}]$

$\ldots\ldots\ldots\ldots,$

$[\ \beta_{10,0}\quad \beta_{10,1}\qquad -6.53615810e\text{-}02\ \ldots\ldots\ldots\ldots\ \beta_{10,11}]\ ]$

We also use the one hot-coded matrix of the classes:
Example: 6 is converted as [[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]]

Neural network Forward:

Linear forward (a):

'a' is a linear combination matrix obtained at the first hidden layer:

$a_{(10,1)} = \alpha_{(10,129)} \times tr\_x\_new[i]_{(129,1)}\ \ \forall\, i \in \{0,\ldots..128\}$

Sigmoid forward:

$z_{(10,1)} = 1/(1+np.exp(-a))$
we add the bias, a row with one to z, and get $z\_bias_{(11,1)}$

Linear forward (b):

'b' is a linear combination matrix obtained at the second hidden layer

$b_{(10,1)} = \beta_{(10,11)} \times z\_bias[i]_{(11,1)}\ \ \forall\, i \in \{0,\ldots..10\}$

Sofmax forward:

$y\_hat_{(10,1)} = exp(b_{(10,1)})/sum(np.exp(b_{(10,1)}))$

Cross entropy:

$J = -sum([one\_hot\_y_{(10,1)} * log(y\_hat_{(10,1)})]_{(10,1)})$

Neural Network Backward:

Softmax backward:

$$g\_b = (\partial J/\partial b)_{(10,1)} = y\_hat_{(10,1)} - one\_hot\_y_{(10,1)}$$

Linear backward:

$$g\_beta_{(10,11)} = g\_b_{(10,1)} \times z\_bias_{(1,11)}$$
$$g\_z_{(10,1)} = \beta\_unbias_{(10,10)} \times g\_b_{(10,1)}$$

Sigmoid backward:

$$g\_a_{(10,1)} = ((z_{(10,1)} \times g\_z_{(1,10)}) \times (1_{(10,1)} - z_{(10,1)})$$

Linear backward:

$$g\_alpha_{(10,129)} = g\_a_{(10,1)} \times X\_train\_s_{(1,129)}$$
$$g\_x_{(128,1)} = alpha\_unbias_{(10,128)} \times g\_a_{(10,1)}$$

We update the alpha and beta values for 500 times (500 number of rows). After all the iterations, we proceed by calculating the loss values.

We iterate all rows of the tr_x_new set and validation set and run Neural Network forward. For every iteration, we calculate the cross-entropy and take the average.

We follow all the above steps for the number of epochs that we provide in the beginning.

We take the optimized values of alpha and beta after running over all the epochs and predict using train and valid data sets. We then calculate the training and validation error.

The results are shown in the Jupyter Notebook.