Machine Learning

Assignment 3

Ishaan Bassi, 2061238

Q1. (a)

Case 1 – Sigmoid activation with 1 hidden layer (100 units)

Final training accuracy = 98.93%

Test set accuracy = 97.505%

Learning Rate = 0.9

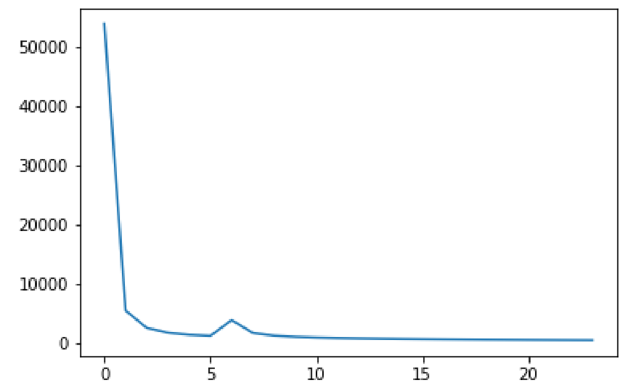


Figure 1 : Cross entropy vs Epochs

Case 2 – Sigmoid activation with 3 hidden layers (100, 50, 50 units)

Final training accuracy = 98.81%

Test set accuracy = 98.04%

Learning rate = 0.9

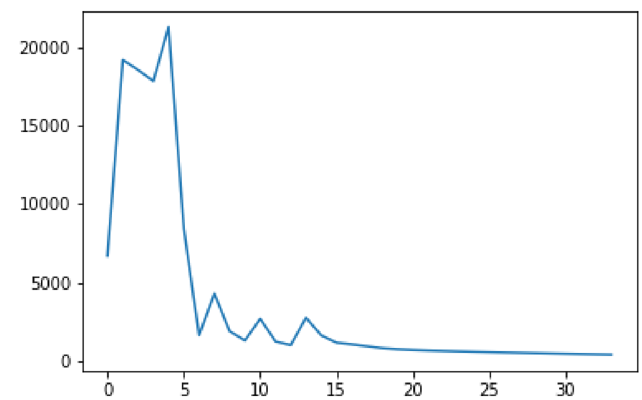


Figure 2 : Cross entropy vs Epochs

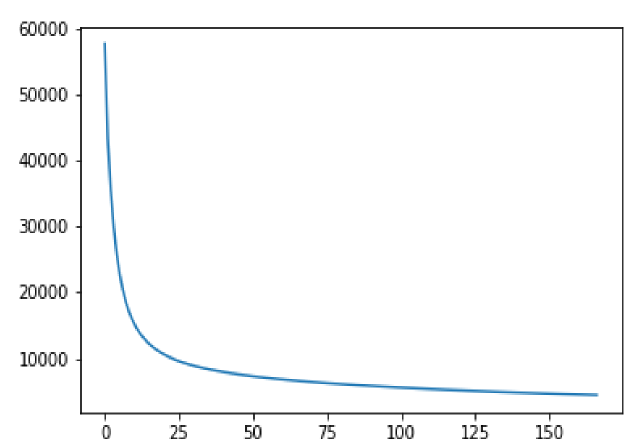
(c)

Case 1 - Relu activation with 1 hidden layer (100 units)

Final training accuracy = 92.6%

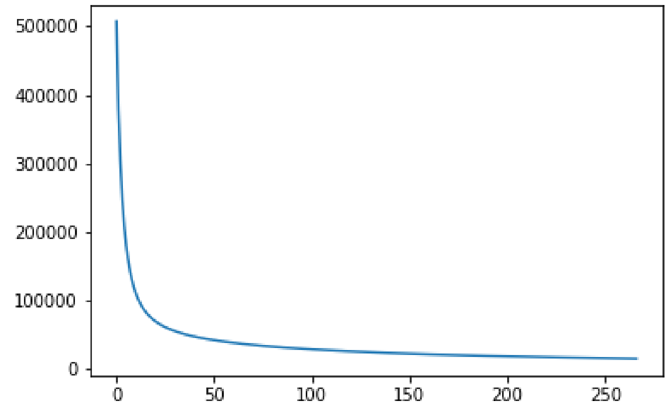
Test set accuracy = 92.023%

Learning rate = .001



Case 4 – Relu activation with 3 hidden layers (100,50,50 units)

Final Training Accuracy = 91.04%
Test set accuracy = 90.47%
Learning Rate = .0001



(b) A model is probably overfitting if the test accuracy is much lower as compared to accuracy on training set. In all the above cases we can see that the training and test accuracies are close. Moreover the graphs show that the cross entropy loss reduces drastically with time. Hence it can be concluded that the model is well trained and can generalize well.

(d) From sklearn library, a SVM with RBF kernel was used. Following are the results –

Test set accuracy =97.694
Train set accuracy = 97.96
C = 10

We can see that SVM gives almost the same level of performance as simple neural networks.

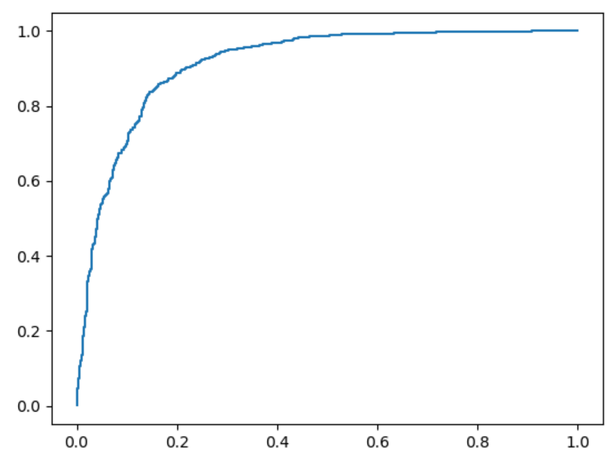
Q2.

Accuracy on the test set – 85.3%

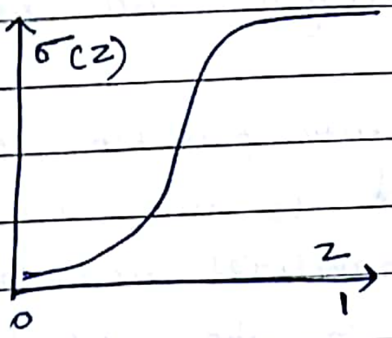
Accuracy on train set - 87.97%

Confusion Matrix

811	189
105	895



3. Sigmoid activation suffers from the vanishing gradients problem.



It can be seen from the figure that the slope of the function is very close to 0 which makes the change in weights very less at each epoch.

On the other hand, RELU activation function has derivative 0 or 1. Due to the large derivative, neural network learns much faster in this case but there is a risk (with deeper networks) of values getting too large, although we can say that the problem is now reduced.

Batch normalization can help in both cases. In this, we scale the input before feeding it to the next hidden layer. This way the scale of values becomes same ~~and~~ and hence reducing the epochs till the neural network reaches minima. It also reduces the effect of previous layers hence the data is not too large or small for the next layer.

We should initialize the weights randomly or else ~~as~~ all the neurons get the same signal and weights are updated in the same way. The initial weights must be

Date:
Page No.

small as large weights can cause neural network to overfit. Hence we can initialize the weights with random value b/w -1 to 1.

For a classification problem, cross entropy should be used as we model the output as probabilities and cross entropy loss is based on the probabilistic interpretation. Moreover cross entropy increases exponentially if the output becomes different from input (i.e. mis classification is heavily penalized) something which can not be done with MSE. Also MSE can cause learning slowdown with sigmoid activation.

Date:
Page No.

4. The mean square error is given by -

$$C = (y-a)^2$$

where $a = \sigma(z)$

$$\Rightarrow \frac{dC}{dw_i} = 2(y-\sigma(z))\frac{da}{dw_i}$$

$$= 2(y-\sigma(z))\,\sigma'(z)\frac{dz}{dw_i}$$

$$= 2(y-\sigma(z))\,\sigma'(z)\,x_i$$

Let the target value be 1 and ~~sigma~~ $\sigma(z)$ be close to 0. We know that $\sigma'(z) = \sigma(z)(1-\sigma(z))$. Hence if $\sigma(z)$ is close to 0, then $\sigma'(z)$ is also very small due to which there is very less change in the weights when they are updated. Similarly, when $\sigma(z)$ is close to 1 and target value is 0, then again $\frac{\partial C}{\partial w_i}$ is very small.

On the other hand cross entropy for ~~binary~~ ~~case~~ is given by -

$$C = -\sum_{i=1}^{n} y_i \log a_i$$

where n = total no. of classes.

then

$$\frac{\partial C}{\partial w_j} = -\frac{y_i}{a_i}\frac{\partial a_i}{\partial w_j}$$

$$= -\frac{y_i}{a_i}\,\sigma'(z)\frac{\partial z}{\partial w_j}$$

$$= \frac{-y_i}{\sigma(z)}\,\sigma(z)(1-\sigma(z))\frac{\partial z}{\partial w_j}$$

$$= -y_i (1-\sigma(z)) \frac{\partial z}{\partial w_j}$$

$$= -y_i (1-\sigma(z)) x_j$$

Here the $\sigma(z)$ cancels out, and hence there is no learning slowdown due to absence of $\sigma'(z)$ term.

Similarly for the binary case,

$$c = -\left(y_i \log(a_i) + (1-y_i)\log(1-a_i)\right)$$

(for one sample)

$$\frac{\partial c}{\partial w_j} = -\left(\frac{y}{a_i}\frac{\partial(a_i)}{\partial w_j} - \frac{(1-y_i)}{(1-a_i)}\frac{\partial(a_i)}{\partial w}\right)$$

$$= -\left(\frac{y}{a_i} a_i(1-a_i)x_j - \frac{(1-y_i)}{(1-a_i)} x_j a_i (1-a_i)\right)$$

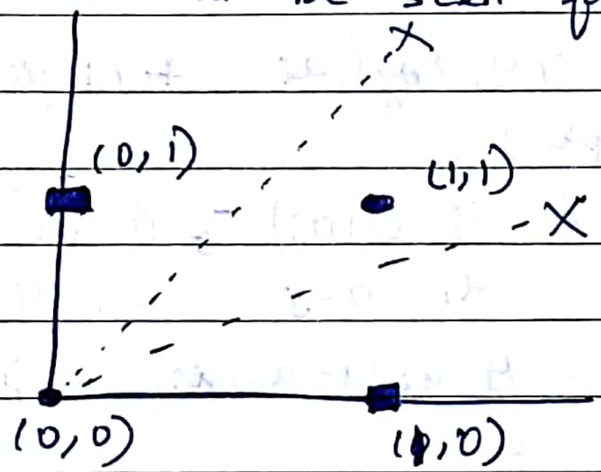
$$= -x_j \left(y_i(1-a_i) - (1-y_i) a_i\right)$$

$$= -x_j \left(y_i - \cancel{y_i a_i} - a_i + \cancel{y_i a_i}\right)$$

$$\boxed{\frac{\partial c}{\partial w_j} = (a_i - y_i) x_j}$$ where $a_i = \sigma(z)$

which again does not contain $\sigma'(z)$.

5. No, a neural network with just linear activations can not be used to create XOR truth table because an arbitrary neural network with just linear activations is same as ~~the~~ a neural network with linear activation and single layer and a linear function can not be used to create a decision boundary that seperates the ~~points~~ 2 classes as can be seen from below diagram



Input	Target
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0