Name: Isha Aggarwal

Date: Dec'18, 2020

Softmax regression and Feed Forward Neural Network

Abstract: The task is to implement softmax regression classifier and feed forward neural network and compare the both. The dataset used is handwritten digits dataset. The report contains 2 sections for each of the classifier, the first one explains the working of classifier in brief while the second one presents the observations and results. The last part of the report compares the 2 classifiers.

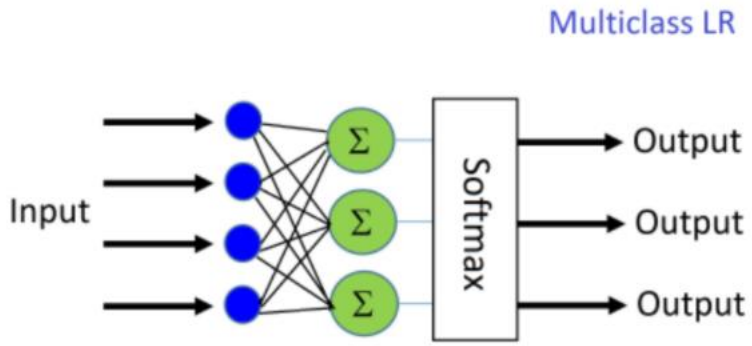
Part 1)

Softmax regression classifier

Section 1:

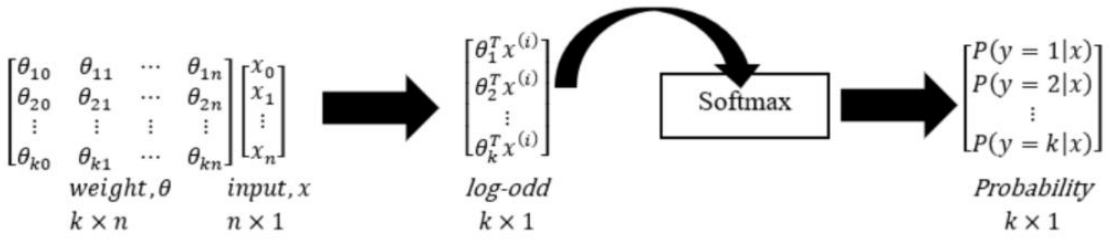
Theory behind Implementation:

Softmax regression classifier is a generalization of logistic regression classifier, and can classify multi-class data. It results in piece-wise linear discriminants.



The parameters are taken as: (bias term is taken care of by including augmenting input pattern with 1)

$$\theta = \begin{bmatrix} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \dots & \theta^{(K)} \\ | & | & | & | \end{bmatrix}$$



Softmax

- $$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

$$P(y^{(i)} = k|x^{(i)};\theta) = \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}$$

Loss function used is cross-entropy loss function.

Cost function

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}\right]$$

- Each training example contributes to this cost.

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right]$$

Here summation is over the training examples we have got in the random sample for the stochastic mini-batch gradient descent.

Used plain-mini-batch gradient descent.

Note: Also wrote code for stochastic mini-batch gradient descent; this also works fine.

Stopping criteria used:

Stop when epochs limit (i.e = 7) is reached. Mini-batch size used = 1, learning rate = 0.001

Section 2:

Results:

Accuracy over test set achieved by using plain mini-batch gradient descent: 90.2 %

Accuracy over training set : 94.45 %

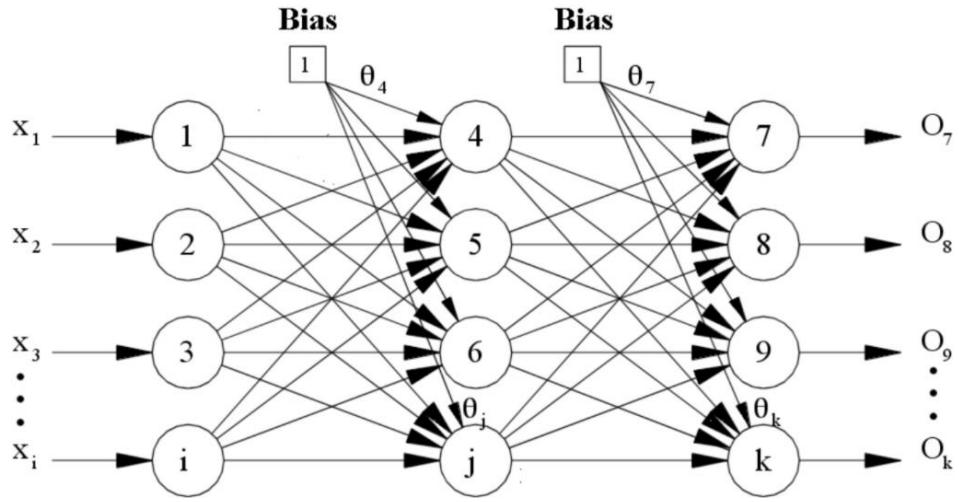
Part 2)

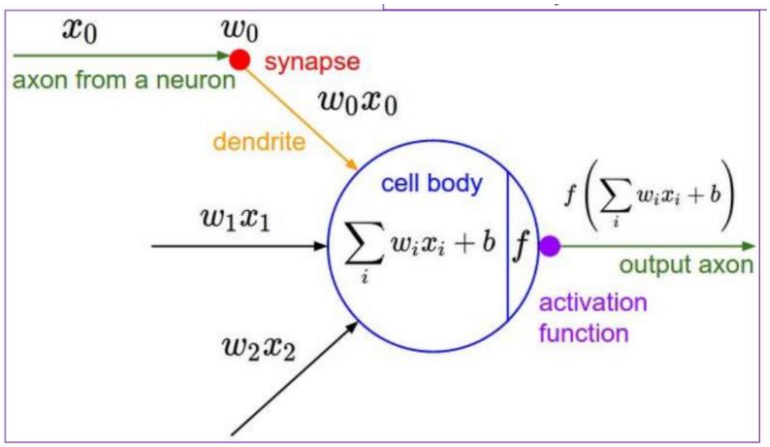
Feed Forward Neural Network

Section 1:

Muti-layer feed forward neural network can classify muti-class data, and it results in non-linear discriminants if the activation function used is non-linear. It updates all the parameters using error backpropagation algorithm.

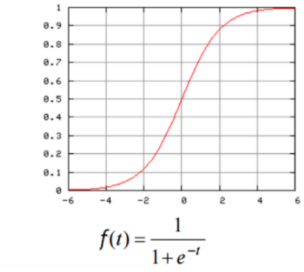
Working/Logic:



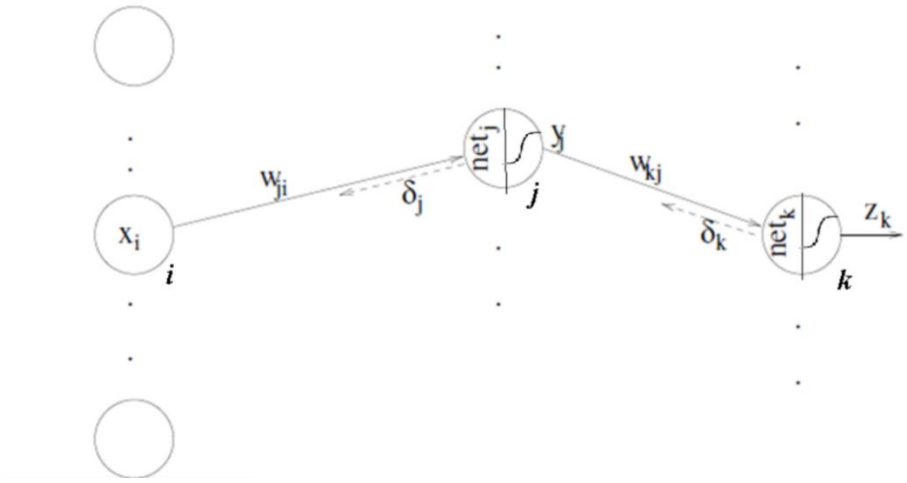


Activation function

Sigmoid function $f'(t) = f(t)(1 - f(t))$



Error Back Propagation-For updation of parametrs:



Update rule for weights between hidden and output units:

- Let w_{kj} be the weight between j^{th} hidden node and k^{th} output node.
- We need to find $\partial J/\partial w_{kj}$
- $J(W) = \frac{1}{2}\sum_{r=1}^{c}(t_r - z_r)^2$, So $\frac{\partial J}{\partial z_k} = -(t_k - z_k)$
- $$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial \text{net}_k}\underbrace{\frac{\partial \text{net}_k}{\partial w_{kj}}}_{y_j} = \frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial \text{net}_k}y_j = \underbrace{-(t_k - z_k)f'(\text{net}_k)}_{\frac{\partial J}{\partial net_k} = -\delta_k} y_j$$
$$= -\delta_k y_j$$
- $\triangle w_{kj} = \eta y_j \delta_k$

Update rule for weights between input and hidden units:

- Let w_{ji} be the weight between i^{th} input node and j^{th} hidden node.

$$\frac{\partial J}{\partial w_{ji}} = \underbrace{\frac{\partial J}{\partial net_j}}_{-\delta_j} \underbrace{\frac{\partial net_j}{\partial w_{ji}}}_{x_i} = \left(\sum_{r=1}^{c} \underbrace{\frac{\partial J}{\partial \text{net}_r}}_{-\delta r} \underbrace{\frac{\partial \text{net}_r}{\partial y_j}}_{w_{rj}} \underbrace{\frac{\partial y_j}{\partial \text{net}_j}}_{f'(net_j)} \right) x_i$$

$$= \underbrace{\sum_{r=1}^{c} -\delta_r w_{rj} f'(net_j)}_{\frac{\partial J}{\partial net_j} = -\delta_j} x_i$$

- $\triangle w_{ji} = \eta x_i \delta_j$

The above theory explains the case when only one hidden layer is present. It can be extended to multiple hidden layers.

---> Implemented from scratch for handwritten digits dataset, with **64 neurons in hidden layer 1, 10 neurons in the second hidden layer and 5 in the last hidden layer**.

Used plain mini-batch gradient descent to find the optimal parameters value. Loss function used is sum of squared errors.

Note: Also wrote code for stochastic mini-batch gradient descent , but that is very slow.

Stopping criteria used:

Stop when epochs limit i.e. 150 is reached. (Batchsize = 1, learning rate = 0.01)

Translated the above theory into code and used vector-matrix operations instead of individual updates.

Section 2:

Results →

Accuracy achieved is:

Over training set : 97 %

Over test set: 78.6 %

Part 3)

Comparison of the 2 classifiers

Discriminant:

For softmax regression classifier: piece-wise linear discriminant

For FFN, non-linear discriminant, when the activation function is non-linear

Accuracy achieved:

For softmax regression classifier, accuracy over test set : 90.2 %

For FFN, accuracy over test set: 78.6 %

Parameters to be learnt:

For softmax regression classifier, number of parameters to be learnt is: (784 + 1)*10 = 7850

For FFN, number of parameters to be learnt is: (784 + 1)*64 + (64 + 1)*10 + (10 + 1)*5 + (5 + 1)*10 = 50240 + 650 + 55 + 60 = 51005
