**Machine Learning by Andrew Ng**

**Week 5 short notes**

Cost Function for multiclass classification is obtained by generalization of existing cost function for Binary classifications.

This new expression consists of a few nested summations to account for the multiple output nodes .

In the first part we have nesting summations that loops through all the output nodes to get corresponding values and sum them up.

In the second part or the regularization part we have 3 nested summations which account for the multiple theta matrices

Note:

* the double sum simply adds up the logistic regression costs calculated for each cell in the output layer
* the triple sum simply adds up the squares of all the individual **Θs** in the entire network.
* the i in the triple sum does **not** refer to training example i

**Back Propagation Algorithm** is a way of minimizing the cost function in neural networks just like gradient descent in case of conventional linear and logistic regressions.

As in case of conventional linear and logistic regressions we are to find the value of **∂(*J*(*θ*)) / ∂*θj*** in order to compute optimal value.

Here this is achieved by introducing a new term called ‘error’.

**δ(l)j** = "error" of node j in layer l.

**δ(L)=y−a(L)** ….. here *a*(*L*) is the vector outputs of activation units of the last layer.

The errors in the previous layers can be calculated by :

**δ(l)=((Θ(l))Tδ(l+1)) .∗ g′(z(l))……** here ‘l’ corresponds to the layer number.

The value of the derivative of the sigmoid functions is given by:

**g′(z(l))=a(l) .∗ (1−a(l))**

now

**∂(J(Θ)) / ∂ Θ(l) i,j=(1/m)∑a(t)(l)j . δ(t)(l+1)i**

When we have more than one data entity in our dataset, the same backward propagation is implemented in a more complex way, introducing a new accumulation term upper case delta Δ.

**Δ(l):=Δ(l)+δ(l+1) . (a(l))T**

* **D(l)i,j:=(1/m)(Δ(l)i,j + λΘ(l)i,j), if j≠0.**
* **D(l)i,j:=(1/m)Δ(l)i,j If j=0**

**D(l)i,j = ∂(J(Θ)) / ∂ Θ(l) i,j**

In order to check the correctness of the Backward Propagation algorithm, we implement a method called gradient check

**(∂J(Θ)/∂Θ) ≈ [J(Θ+ϵ)−J(Θ−ϵ)]/2ϵ….** If the values obtained by both methods come out to be approximately same, then we can safely say that we have a very functional backward propagation algorithm and hence a more accurate hypothesis.

Initially we saw in conventional regressions we set the initial values of the parameters to zero and by the end of multiple iterations performed for minimization of the cost function we arrived at the necessary parameters. This seems to no longer hold good in case of neural networks as it ends up producing the same features and hence leading to an inaccurate hypothesis.

This problem is overcome by setting random initial values for the parameters between zero and one, by this we can ensure a more promising and accurate hypothesis.