One of Pioneer of Neutral Network：Yann LeCun

Multi-class classification:

Suppose we have four class;

The output layer is the essentially four logistic regression classifiers. Each of them will capture one of the class.

Training set is the (x1,y1), (x2,y2),(x3,y3)….(xm,ym)

Y is [1,0,0,0]/[0,1,0,0]/[0,0,1,0]/[0,0,0,1] to represent the four classes.

Back propagation algorithm:

Intuition: delta\_l\_j equals the “error” of node j in layer l.

E.g: for each output unit(layer L = 4)

delta\_4 = a\_4 – y;

delta\_3 = Theta\_3' \* delta\_4 .\* g'(z\_3)(note: g'(z\_3) equals a\_3 .\* (1 – a\_3));

delta\_2 = Theta\_2' \* delta\_3 .\* g'(z\_2)(note: g'(z\_2) equals a\_2 .\* (1 – a\_2));

Implementation of Back Propagation Algorithm:

Training set {(x1,y1), (x2, y2), … , (xm,ym)}

set Delta\_l\_ij = 0 (for all l, I, j)

For I = 1 to m

Set a\_1 = x\_i

Perform forward propagation to compute the a\_l for l = 2, 3, … , L

Using y\_i, compute delta\_L = a\_L – y\_i

Compute delta\_L-1， delta\_L-2, …, delta\_2

Delta\_l\_ij = Delta\_l\_ij + a\_l \_j \* delta\_l+1\_i

D\_l\_ij := 1/m \* Delta\_l\_ij + lambda \* Theta\_l\_ij if j != 0

D\_l\_ij := 1/m \* Delta\_l\_ij if j == 0(bias unit!)

Note: partial derivative of cost J(Theta)\_l\_ij equals to D\_l\_ij;

Implementation Note:

**Gradient Checking:**

gradApprox = (J(Theta + epsilon) – J(Theta – epsilon)) / (2 \* epsilon);

Implementation Node:

Implement back propagation to compute the Dvec(unrolled D1,D2,D3)

Implement numerical gradient check to compute gradApprox(Computationally expensive and slow)

Make sure they give the similar values.

Turn off gradient checking. Using back prop code for learning.(back prop is computationally efficient way fo computing for derivatives!)

**Important: Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of costFunction(…)) your code will be very slow!**

**Random initialization**

Symmetry breaking

Initialize each Theta\_l\_ij to a random value in [-epsilon, epsilon].

E.g.

Theta1 = rand(10, 11) \* (2 \* Init\_epsilon) – Init\_epsilon;

Theta2 = rand(1, 11) \* (2 \* Init\_epsilon) – Init\_epsilon;

**Summary of all pieces of above, and put them together.**

**Pick a network architecture:**

Number of input units is the dimension of features xi.

Number of output units is determined by the number of class in a classification problems.

Number of hidden units and hidden layers:

Reasonable default: 1 hidden layer, or if > 1 hidden layer, have same number of hidden units in every layer(usually the more the batter)

**Training a neural network:**

1.Randomly initialize weights

2.Implements forward propagation to get h(xi) for any xi

3.implements code to compute cost function J(Theta)

4.implements back propagation to compute the partial derivatives J(Theta)

5.Use gradient checking to compare the partial derivatives J(Theta) computed using back propagation vs. numerical estimate of gradient of J(Theta).

6.Use gradient descent or other advanced optimization method with back propagation to try to minimize J(Theta) as a function of parameters Theta.