

Neural Networks: Cost Function

- [Classification](#)
- [Cost function](#)
- [Backpropagation algorithm](#)
 - [Gradient computation](#)
 - [Backpropagation algorithm](#)
- [Backpropagation intuition](#)
- [Implementation note: Unrolling parameters](#)
 - [Example](#)
 - [Learning Algorithm](#)
- [Gradient checking](#)
- [Random initialization](#)
 - [Symmetry breaking](#)
- [Putting it together](#)
 - [Training a neural network](#)

Classification

L = total no. of layers in network

s_l = no. of units (not counting bias unit) in layer l

Binary classification	Multi-class classification (K classes)
$y = 0 \text{ or } 1$	$y \in \mathbb{R}^K$
1 output unit	K output units

Cost function

Logistic regression:

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m (y^{(i)} \log(h_{\Theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\Theta}(x^{(i)}))) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \Theta_j^2$$

Neural network:

$$h_{\Theta}(x) \in \mathbb{R}^K, (h_{\Theta}(x))_i = i^{th} output$$

$$J(\Theta) = -\frac{1}{m} [\sum_{i=1}^m \sum_{k=1}^K (y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - h_{\Theta}(x^{(i)})_k))] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

Backpropagation algorithm

Gradient computation

$$J(\Theta) = -\frac{1}{m} [\sum_{i=1}^m \sum_{k=1}^K (y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - h_{\Theta}(x^{(i)})_k))] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

need to compute:

$$J(\Theta)$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$$

Backpropagation algorithm

$$\begin{aligned} \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) &= \frac{\partial J(\Theta)}{\partial a_i^{l+1}} * \frac{\partial a_i^{l+1}}{\partial \Theta_{ij}^{(l)}} \\ &= \frac{\partial J(\Theta)}{\partial a_i^{l+1}} * \frac{\partial a_i^{l+1}}{\partial z_i^{l+1}} * \frac{\partial z_i^{l+1}}{\partial \Theta_{ij}^{(l)}} \end{aligned}$$

$$\frac{\partial z_i^{l+1}}{\partial \Theta_{ij}^{(l)}} = w_i = a_j^l$$

$$\frac{\partial a_i^{l+1}}{\partial z_i^{l+1}} = \frac{\partial g(z_i^{l+1})}{\partial z_i^{l+1}} = g(z_i^{l+1})(1 - g(z_i^{l+1})) = a_i^{l+1}(1 - a_i^{l+1})$$

$$\begin{aligned} \frac{\partial J(\Theta)}{\partial a_i^{l+1}} &= \sum_j \frac{\partial J(\Theta)}{\partial a_j^{l+2}} * \frac{\partial a_j^{l+2}}{\partial a_i^{l+1}} \\ &= \sum_j \frac{\partial J(\Theta)}{\partial a_j^{l+2}} * \frac{\partial a_j^{l+2}}{\partial z_j^{l+2}} * \frac{\partial z_j^{l+2}}{\partial a_i^{l+1}} \end{aligned}$$

$$\frac{\partial z_j^{l+2}}{\partial a_i^{l+1}} = \Theta_{ji}^{(l+1)}$$

If we have got $\frac{\partial J(\Theta)}{\partial a_j^{l+2}}$, we can calculate $\frac{\partial J(\Theta)}{\partial a_j^{l+1}}$, and next $\frac{\partial J(\Theta)}{\partial a_j^l}$, and so on.

Backpropagation intuition

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_{\Theta}(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

Focusing on a simple example $(x^{(i)}, y^{(i)})$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

$$\text{cost}(i) = y^{(i)} \log(h_{\Theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\Theta}(x^{(i)}))$$

$$\text{or } \text{cost}(i) \approx (h_{\Theta}(x^{(i)}) - y^{(i)})^2$$

$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l). However, **in my opinion**, it's "error" of cost $z_j^{(l)}$ from a view of calculus.

$$\delta_j^{(l)} = \frac{\partial J(\Theta)}{\partial a_i^l} * \frac{\partial a_j^l}{\partial z_j^l} = \frac{\partial J(\Theta)}{\partial a_i^l} * a_i^l * (1 - a_i^l), \text{ so we have } \frac{\partial J(\Theta)}{\partial a_i^l} = \sum_j \frac{\partial J(\Theta)}{\partial a_j^{l+1}} * \frac{\partial a_j^{l+1}}{\partial z_j^{l+1}} * \frac{\partial z_j^{l+1}}{\partial a_i^l} =$$

$$\sum_j \Theta_{ji}^{(l)} * \delta_j^{(l+1)} = [\Theta_{ji}^{(l)}]^T * \delta_j^{(l+1)}$$

For example,

$$\delta_1^{(4)} = a_1^{(4)} - y^{(i)}$$

$$\delta_2^{(3)} = \Theta_{12}^{(3)} * \delta_1^{(4)}$$

$$\delta_2^{(2)} = \Theta_{12}^{(2)} * \delta_1^{(3)} + \Theta_{22}^{(2)} * \delta_2^{(3)}$$

...

Implementation note: Unrolling parameters

Advanced optimization

Example

$$s_1 = 10, s_2 = 10, s_3 = 1$$

$$\Theta^{(1)} \in \mathbb{R}^{10 \times 11}, \Theta^{(2)} \in \mathbb{R}^{10 \times 11}, \Theta^{(3)} \in \mathbb{R}^{1 \times 11}$$

$$D^{(1)} \in \mathbb{R}^{10 \times 11}, D^{(2)} \in \mathbb{R}^{10 \times 11}, D^{(3)} \in \mathbb{R}^{1 \times 11}$$

```
thetaVec = [ Theta1(:); Theta2(:); Theta3(:)];  
DVec = [ D1(:); D2(:); D3(:)];  
Theta1 = reshape(thetaVec( 1: 110), 10, 11);  
Theta2 = reshape(thetaVec( 111: 220), 10, 11);  
Theta3 = reshape(thetaVec( 221: 231), 1, 11);
```

Learning Algorithm

1. Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$;
2. Unroll to get *initialTheta* to pass to *fminunc*(@costFunction, initialTheta, options) ;
3. *function* [jval, gradientVec] = costFunction(thetaVec);
 - 3.1 From *thetaVec*, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$;
 - 3.2 Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$;
 - 3.3 Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get *gradientVec*.

Gradient checking

Implement: $gradApprox = \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}$

Check that $gradApprox \approx DVec$

Implement Node:

1. Implement backprop to compute *DVec* (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$);
2. Implement numerical gradient check to compute *gradApprox*;
3. Make sure they give similar values;
4. Turn off gradient checking. Using backprop code for learning.

Random initialization

Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value $[-\epsilon, \epsilon]$.

```
Theta1 = rand(10, 11) * (2 * INIT_EPSILON) - EPSILON;
```

Putting it together

Training a neural network

Pick a network architecture.

1. Randomly initialize weights;
2. Implement forward propagation to get $h_{\Theta}(x^{(i)})$ for any $x^{(i)}$;

3. Implement code to compute cost function $J(\Theta)$;
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$;
 for $i = 1 : m$
 Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
 (Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \dots, L$);
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using
 numerical estimate of gradient of $J(\Theta)$;
 Then disable gradient checking node;
6. Use gradient descent or advanced optimization method with backpropagation to try to minimize
 $J(\Theta)$ as a function of parameters Θ .