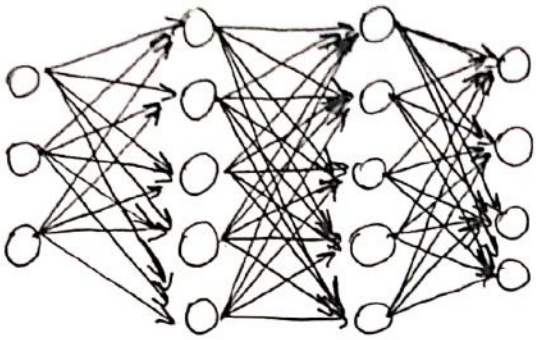
week 5

Neural Networks : Learning

Cost Function



K = Çıkış düğümlerinin sayısı

L = Ağdaki toplam katman sayısı

S_l = Her l. katmandaki toplam birim sayısı

Not: Ağdaki giriş katmanı toplam katman sayısına katılmaz.

$L = 3 \quad S_0 = 3 \quad S_1 = 5 \quad S_2 = 5 \quad S_3 = 4$

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

Not: Lojistik regresyonun maliyet fonksiyonundan farklılık olarak tüm çıkış düğümleri için kayıp fonksiyonunu hesaplamak için $\sum_{k=1}^{K}$ eklenmiştir. Ayrıyeten regularization için katmanlardaki tüm ağırlıkların hesaplanması için revize edilmiştir.

Back propagation Algorithm

Bir olayda geri gidebilmek için önce ileri gitmeliyiz. Bu olay neural networkste forward propagation olarak adlandırılır.

a. Forward Propagation

$a^{(1)} = x$

$z^{(2)} = \Theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$

$z^{(3)} = \Theta^{(2)} a^{(2)}$

$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$

$z^{(4)} = \Theta^{(3)} a^{(3)}$

$a^{(4)} = h_\Theta(x) = g(z^{(4)})$

Not: Sayfanın başındaki ağ örnek alınarak oluşturulmuştur.

Backpropagation algoritmasının ana mantığı her katmandaki düğümlerin hatalarını temsil eden değerleri bulmaktır ve ağırlıkları gradient ile güncellemektir.

$d z^{(4)} = \delta^{(4)} = a^{(4)} - y$

$d z^{(3)} = \delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$ (Aktivasyon fonk. türevi $a^{(3)}.(1-a^{(3)})$)

$d z^{(2)} = \delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$

Yukarıdaki katman örneğindeki katmanlar için backpropagation nasıl hesaplandığı gösterilmiştir.

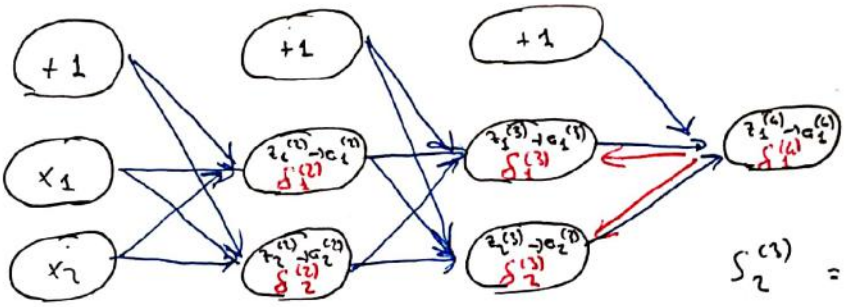
Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)} \Longrightarrow \Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)}$ if $j \neq 0$

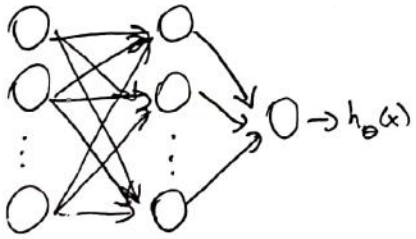
$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$



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

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

Parametre Oluşturma Örneği



$S_0 = 10, \quad S_2 = 10, \quad S_3 = 1$

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

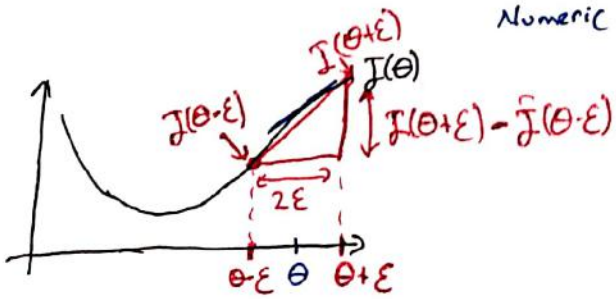
$D^{(1)} \in \mathbb{R}^{10 \times 11}, \quad D^{(2)} \in \mathbb{R}^{10 \times 11}, \quad D^{(3)} \in \mathbb{R}^{1 \times 11}$

```
thetaVec = [Theta1(:); Theta2(:); Theta3(:)]
DVec     = [D1(:); D2(:); D3(:)]
```

```
Theta1 = reshape(thetaVec(1:110), 10, 11)
Theta2 = reshape(thetaVec(111:220), 10, 11)
```

Gradient Checking

Neurel network gibi karışık algoritmalarda back propagation bazen hatalı(bug) çalışır fakat maliyet fonksiyonu azalmaya devam eder. Hatanın var olup olmadığını anlamak için gradient checking kullanırız.



Numeric Gradient Algorithm

$$\frac{d}{d\theta} J(\theta) \cong \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$

Epsilon yaklaşık olarak 10^{-4} seçilir daha küçük seçmek türeve yaklaştırır ancak matematiksel sorunlar yaratabiliyor.

Eğer bir vektör içindeki parametreler için uygulamak istersenız vektörün içinde teker teker dolaşıp formülü uygulamalısınız. Matlab kodu aşağıdaki gibidir.

```
for i = 1:n,
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + Epsilon;
    thetaMinus = theta;
    thetaMinus(i) = thetaMinus(i) - Epsilon
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus)) / (2*Epsilon)
end;
```

Bizim backpropagation'da türevleri veren vektörümüz DVec idi. Buradaki değerler ile gradApprox'taki değerler birbirine yaklaşık olarak eşit olursa doğru çalışıyor diyebiliriz.

Modelinizin doğru çalışıp çalışmadığını kontrol ettikten sonra bir daha bu karşılaştırmayı çalıştırmayın. Çünkü oldukça pahalı bir türev hesaplama yöntemi.

Random Initialization Parameter

Eğer parametreleri 0'a eşitlersek hem forward propag. hem de backward propagation'da tüm nöronlar birbirine eşit olur. Bunu çözmek için parametreleri simetrik olmayacak şekilde rastgele sayılardan seçmeliyiz.

```
Theta = rand(10,11) * (2* Init_Epsilon) - Init_Epsilon;
```

→ $[-\varepsilon, \varepsilon]$

↳ Gradient checking epsilonlarla değil

Neurel Network Eğitme Adımları

1) Ağırlıklar random olarak oluşturulur.
2) Tahmini üretebilmek için ($h_\Theta(x)$) forward propagation uygulanır.
3) Cost function($J(\Theta)$) hesaplayan kod yazılır.
4) Θ'nın kısmi türevlerini hesaplamak için back propagation uygulanır.
5) Numeric estimate gradient kullanarak backpropagation parametreleri kontrol edilir.
6) Gradient descent gibi optimizasyon fonksiyonu kullanılarak optimum kayıp hesaplanır.