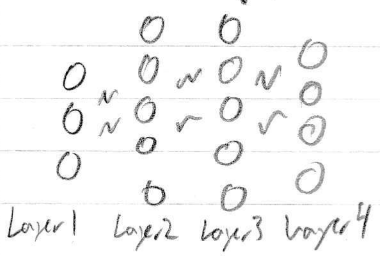
Machine Learning - Week 5 - Neural Networks: Learning

T1 - Cost Function and Backpropagation

L1 - Cost Function

Neural Network (Classification)



$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

L = total # of layers in network

s_l = number of units (not counting bias unit) in layer l

Recall:

Binary Classification

$y = 0$ or 1

1 output unit

$h_\Theta(x) \in \mathbb{R}$

$S_L = 1$

$K = 1$

Multi-class Classification (K Classes)

$y \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

person, car, motorcycle, truck

K output units $\rightarrow h_\Theta(x) \in \mathbb{R}^K$

$S_L = K \quad (K \geq 3)$

Cost Function

Recall: 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:

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

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

L2- Backpropagation Algorithm

Recall what $J(\Theta)$ is (on previous page)

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

Need code to compute:

→ $J(\Theta)$

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

Gradient Computation:

Given one training example (x, y)

Forward Propagation:

$a^{(1)} = x$

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

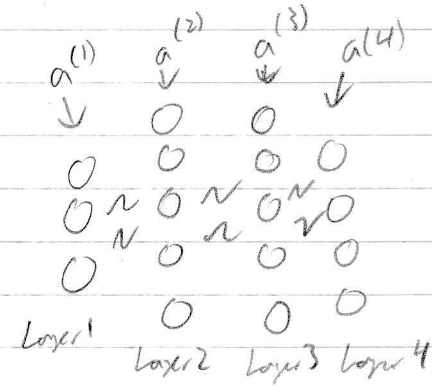
$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

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

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

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

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



Backpropagation Algorithm

Intuition: $d_j^{(l)}$ = "error" of node j in layer l.

For each output unit (layer L=4)

$d_j^{(4)} = a_j^{(4)} - y(j)$ $(h_\Theta(x))_j$ Vectorized: $d^{(4)} = a^{(4)} - y$

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

$d^{(2)} = (\Theta^{(2)})^T d^{(3)} .* g'(z^{(2)})$ $a^{(2)} .* (1 - a^{(2)})$

No $d^{(1)}$, since the first layer is the input layer (feature layer) and we don't want to change those values

$$\frac{d}{d\Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} d_i^{(l+1)}$$ (ignoring λ; if $\lambda = 0$)

Backpropagation algorithm.

Training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j). (Used to compute $\frac{d}{d\Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to m

- Set $a^{(1)} = x^{(i)}$
- Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \dots, L$
- Using $y^{(i)}$, compute $d^{(L)} = a^{(L)} - y^{(i)}$
- Compute $d^{(L-1)}, d^{(L-2)}, \dots, d^{(2)}$
- $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} d_i^{(l+1)}$
- Vectorized: $\Delta^{(l)} := \Delta^{(l)} + d^{(l+1)} (a^{(l)})^T$

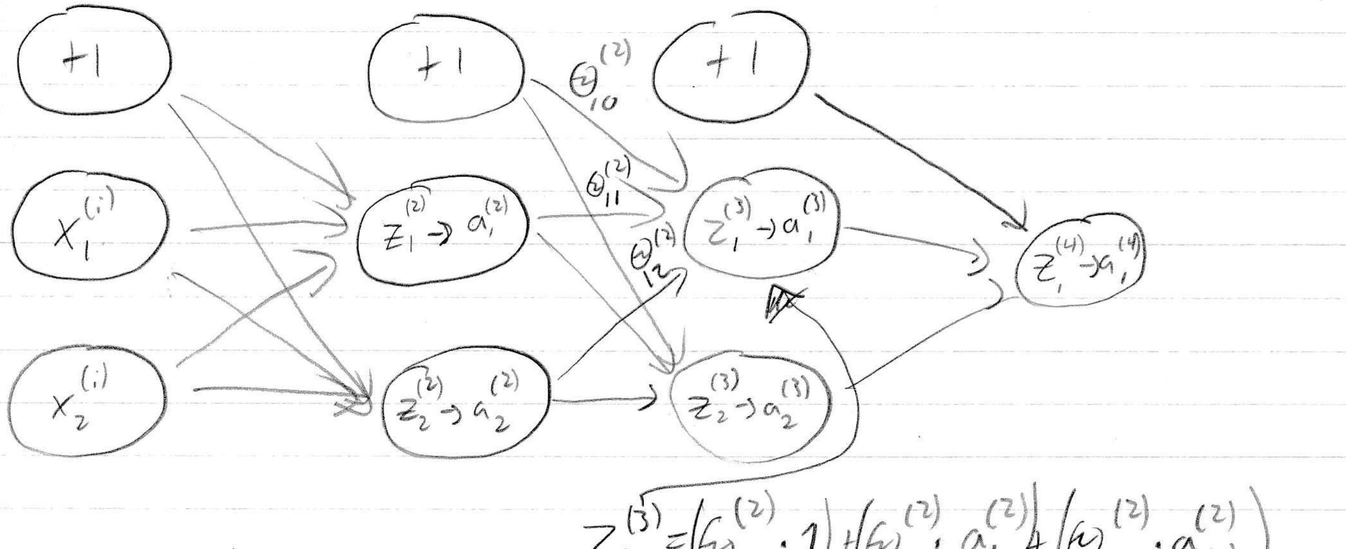
$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0$$

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

$$\frac{d}{d\Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

L3 - Backpropagation Intuition

Recall Forward Propagation



$$z_1^{(3)} = (\Theta_{10}^{(2)} \cdot 1) + (\Theta_{11}^{(2)} \cdot a_1^{(2)}) + (\Theta_{12}^{(2)} \cdot a_{(2)}^{(2)})$$

What is Backpropagation doing?

Recall: $$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} \underbrace{y^{(i)}\log(h_\Theta(x^{(i)})) + (1-y^{(i)})\log(1-(h_\Theta(x^{(i)})))}_{(x^{(i)},\,y^{(i)})}\right]$$
$$\cancel{+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ij}^{(l)}\right)^2}$$

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

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

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

i.e. how well is the network doing on example i?

<u>watch rest of video for example</u>

T2 - Backpropagation in Practice
L1 - Implementation Note: Unrolling Parameters

Advanced Optimization

function [jval, gradient] = costFunction(theta)

$\hookrightarrow \mathbb{R}^{n+1}$ $\hookrightarrow \mathbb{R}^{n+1}$ (vectors)

...

optTheta = fminunc(@costFunction, initialTheta, options)

Neural Network ($L=4$):

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)

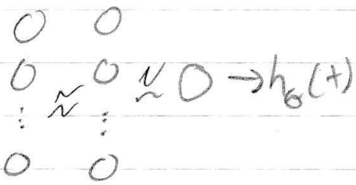
$D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (D1, D2, D3)

"Unroll" into vectors

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

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

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



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

Have initial parameter $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.

Unroll to get initialTheta to pass to fminunc(@costFunction, initialTheta, options)

function [jval, gradientVec] = costFunction(thetaVec)

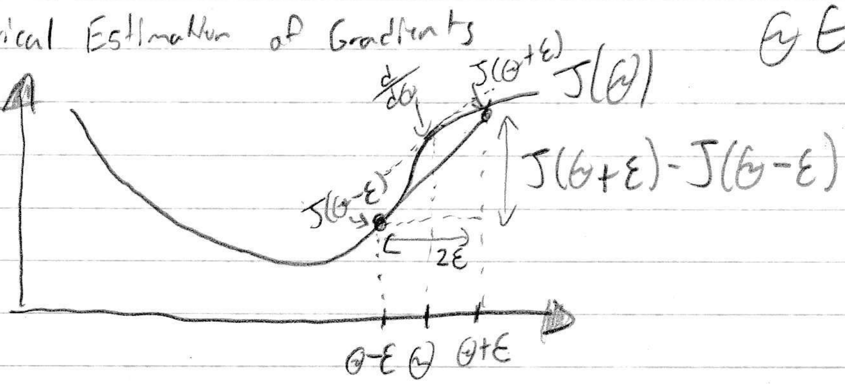
→ From thetaVec, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$. (reshape)

→ Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$.

→ Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get gradientVec.

L2 - Gradient Checking

Numerical Estimation of Gradients $\quad \theta \in \mathbb{R}$



$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon} \qquad \varepsilon = 10^{-4}$$

Implement: gradApprox = (J(theta + EPSILON) - J(theta - EPSILON))/(2·EPSILON)

Parameter Vector θ

$\theta \in \mathbb{R}^n$ (e.g. θ is "unrolled" version of $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$)

$\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]$

$$\frac{d}{d\theta_1} J(\theta) \approx \frac{J(\theta_1+\varepsilon, \theta_2, \theta_3, \ldots, \theta_n) - J(\theta_1-\varepsilon, \theta_2, \theta_3, \ldots, \theta_n)}{2\varepsilon}$$

$$\frac{d}{d\theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\varepsilon, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2-\varepsilon, \theta_3, \ldots, \theta_n)}{2\varepsilon}$$

$\vdots$

$$\frac{d}{d\theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n+\varepsilon) - J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n-\varepsilon)}{2\varepsilon}$$

Code:
```
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;
```

Check that gradApprox $\approx$ DVec

From backprop

Implementation Note:

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

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.

L3-Random Initialization

Initial value of Θ

For gradient descent and advanced optimization method, need initial value for Θ.

optTheta = fminunc(@costFunction, initialTheta, options)

Consider gradient descent

Set initialTheta = zeros(n,1)?

NO

So $\Theta_{ij}^{(l)} = 0$ for all i, j, l.

$\therefore a_1^{(2)} = a_2^{(2)}$

Also $\delta_1^{(2)} = \delta_2^{(2)} \rightarrow \frac{d}{d\Theta_{01}^{(1)}} J(\Theta) = \frac{d}{d\Theta_{02}^{(1)}} J(\Theta) \rightarrow \Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

↳ After each update, parameters corresponding to inputs going into each of two hidden units are identical.

$\Rightarrow$

Different ε than previous slides

Random Initialization: Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$ i.e. $-\varepsilon \le \Theta_{ij}^{(l)} \le \varepsilon$)

eg Theta1 = rand (10,11) · (2·INIT_EPSILON) − INIT_EPSILON;

Theta2 = rand (1,11) · (2·INIT_EPSILON) − INIT_EPSILON;

L4 - Putting it together

Training a neural network

- Pick a network architecture (connectivity pattern between neurons):
  - Number of input units: Dimension of features $x^{(i)}$
  - Number of output units: Number of Classes
  - Reasonable default: 1 hidden layer, or if >1 hidden layer, have same number of hidden units in every layer (usually the more the better)

Then:

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 $l = 2, \ldots, L$).

   $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

   end;

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

5) Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimation of gradient of $J(\Theta)$

   Then disable gradient checking code
6) Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ.