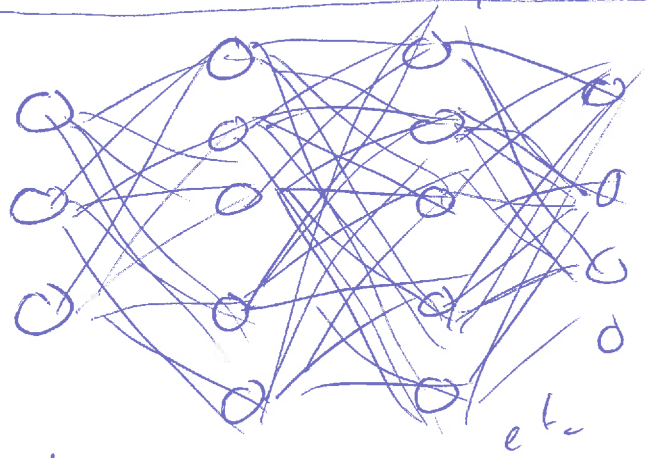
Notes: Week 5

Neural Networks learning: Cost function

Neural Networks → for Classification



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

→ L = total # of layers in network

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

$L = 4$

$s_1 = 3, \ s_2 = 5, \ s_4 = 4$

Binary classification

$y = 0$ or 1 (binary)

1 output unit

$h_\theta(x) \to \mathbb{R}$ (real #)

$S_L = 1 \ / \ K = 1$

Multi-class classification

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

pedestrian car motor cycle truck

"K" output units

$h_\theta(x) \in \mathbb{R}^K$

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

this case →

Cost function:

Logistic regression: Instead of having 1 (logistic regression) unit, we may in fact have K of them. ie

regularization term

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

Notes: Week 5

Neural Networks learning: Cost function

Neural Network error function for k output elements.

$h_\Theta(x) \in \mathbb{R}^k$ $k=1$ (for binary classification)

$(h_\Theta(x))_i = i^{th}$ output

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{k} y_k^{(i)}\log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k\right)$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\Theta_{ji}^{(l)})^2$$

$$\sum\begin{pmatrix}0\\0\\0\\0\end{pmatrix}\begin{matrix}k=1\\ \vdots\\ \vdots\\ k=4\end{matrix} \qquad y_k = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

what we want

regularization term.

do not sum over the areas where $i=0$

say first hidden layer then

$\underbrace{\Theta_{10}^{(2)}x_0}_{a_0} + \Theta_{1}^{(2)}x_1 +$

(We dont sum over this term).

Notes: Week 5

Back propagation Algorithm

~~Cost~~ Cost function $J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} \sim\sim\sim\sim\sim\right]$

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2.$$

Need code to compute: $J(\Theta)$

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

parameters are $\Theta_{ij}^{(l)} \in \mathbb{R}$ & where all parameters are real numbers.

Gradient computation:

Given one training example (x, y):

lets see what happens.

~~Forward~~ first forward propagation:

$a^{(1)} = x$

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

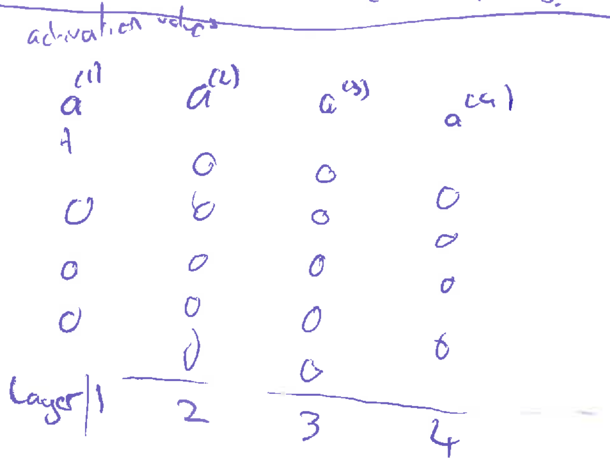
$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

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

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

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

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



Notes : Week 5

Gradient computation : Backpropagation algorithm

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

$a_j^{(l)}$ = activation of element "j" in layer l

For each output unit (layer L=4)

$\delta_j^{(4)} = (a_j^{(4)}) - y_j$ ← the actual value of the unit observed in the training example

y_j: jth element of our training set $(h_\theta(x))_j$

$a_j^{(4)}$: activation of the unit

The δ term is just the difference between what our hypothesis $(h_\theta(x))_j$ output, and our original value in the training set.

jth element

$\delta^{(4)} = a_j^{(4)} - y_j$

Vector dimension = number of output units in our network

so: $\delta_j^{(4)} = a_j^{(4)} - y_j$

$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$

elementwise multiplication

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

derivatives calculated by calculus

$a^{(3)} .* (1 - a^{(3)})$

$a^{(2)} .* (1 - a^{(2)})$

no $\delta^{(1)}$!!

Ignoring regularization $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^l \delta_i^{(l+1)}$ (ignoring λ; (if $\lambda = 0$))

Notes: Week 5

Gradient computation: backpropagation Algorithm

training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

set $\Delta^{l}_{ij} = 0$ (for all l, i, j) | used to compute $\frac{\partial}{\partial \Theta^{l}_{ij}} J(\Theta)$

loop) For $i = 1$ to m ← mth element =`s $(x^{(i)}, y^{(i)})$

1. Set $a^{(1)} = x^{(i)}$
2. Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3 \ldots L$
3. Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ ← (backprop), $\cancel{\delta^{(1)}}$

$\Delta^{l}_{ij} := \Delta^{l}_{ij} + a^{(l)}_{j} \delta^{(l+1)}_{i}$. → ij $\Delta^{(l)} := \Delta^{l} + \delta^{(l+1)} (a^{(l)})^{T}$.

loop

finally two last pieces outside the for loop:

→ $D^{l}_{ij} := \frac{1}{m} \Delta^{l}_{ij} + \lambda \Theta^{(l)}_{ij}$ if $j \neq 0$

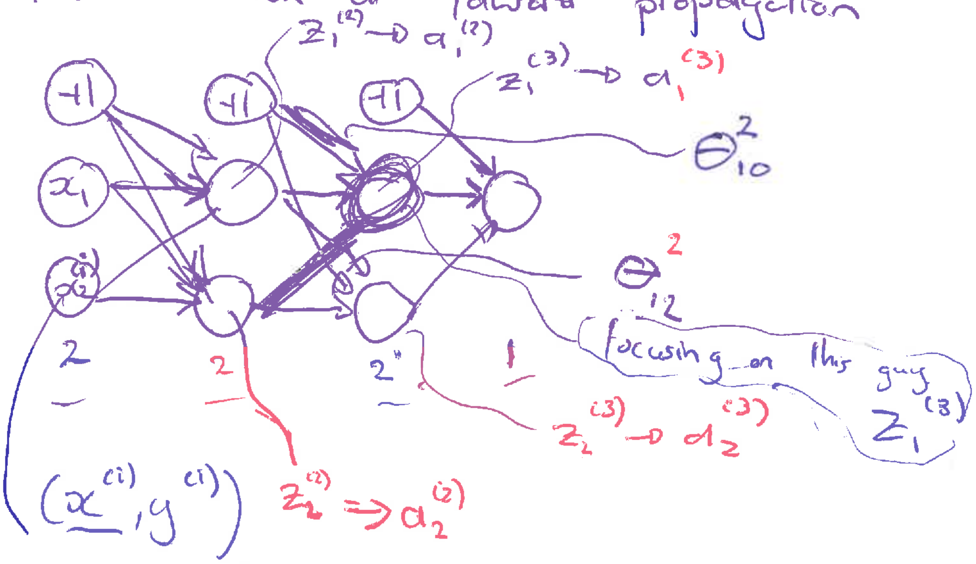
→ $D^{l}_{ij} := \frac{1}{m} \Delta^{(l)}_{ij}$ if $j = 0$. (bias term)

Finally: $\frac{\partial}{\partial \Theta^{l}_{ij}} J(\Theta) = D^{l}_{ij}$

Notes: Week 5

Backpropagation Intuition

First lets look at forward propagation



forward propagation.

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

backpropagation is very similar to FP only that the computations are now flowing back through the network in the opposite direction.

What is backpropagation doing?

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

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

how well is the network doing on example i?

→ think of $\boxed{\text{cost}(i) \approx (h_\Theta(x^{(i)}) - y^{(i)})^2}$. squared error cost function.

$z_j^{(l)}$ if we can go inside the network and change the terms

δ_j^{l} → partial derivative of the cost function with respect to the intermediate terms we are computing.

↳ and so, they are a measure of how much we would like to change the neural networks weights, in order to affect the z step intermediate values of the (FP) computation so as to affect the overall output $h_\Theta(x)$ and ∴ the overall cost.

Formally $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(i)$ " (for $j \geq 0$) where

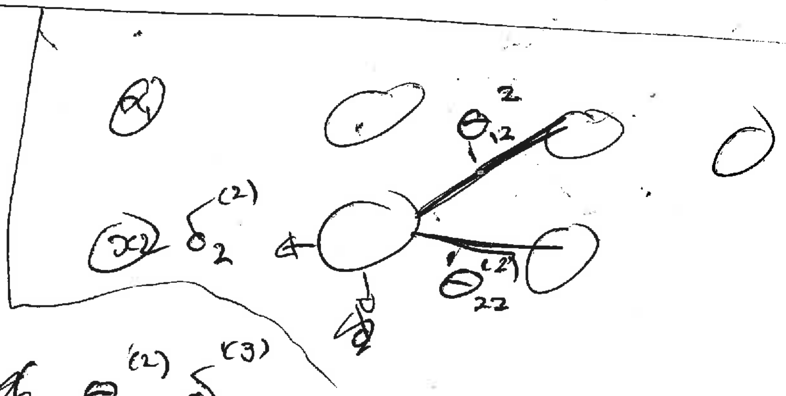
$$\text{cost}(i) = y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log h_\theta(x^{(i)})$$

so first

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

propagate backwards:

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



Notes - Week 5

Neural Networks: Learning: Implementation note: unrolling parameters

Advanced optimization

function [jVal, gradient] = costFunction(theta)
$\qquad\qquad\qquad\quad \hookrightarrow \mathbb{R}^{n+1}$

$\to \mathbb{R}^n$ (real numbers)
Vectors — Assumes that both thetas are vectors

...

optTheta = fminunc(@costFunction, initialTheta, options)

taking Neural Network (L=4) as an example:

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

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

"Unroll" into vectors

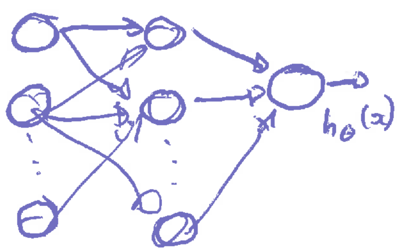
---

example — two hidden units

$\boxed{s_1 = 10}, \boxed{s_2 = 10}, \boxed{s_3 = 1}$ — output unit

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

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



matlab command:

```
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);
```

→ elements that you want

→ dimensions of matrix that you are after.

Learning Algorithm

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

Unroll to get initialTheta to pass to (step 1)

fminunc(@costFunction, initialTheta, options)

Notes :- Week 5

Neural Networks: Learning : Implementation note: unrolling parameters

Learning algorithm

function [jval, gradientVec] = costFunction (thetaVec)

From thetaVec, get $\boxed{\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)},}$ reshape to matrices

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

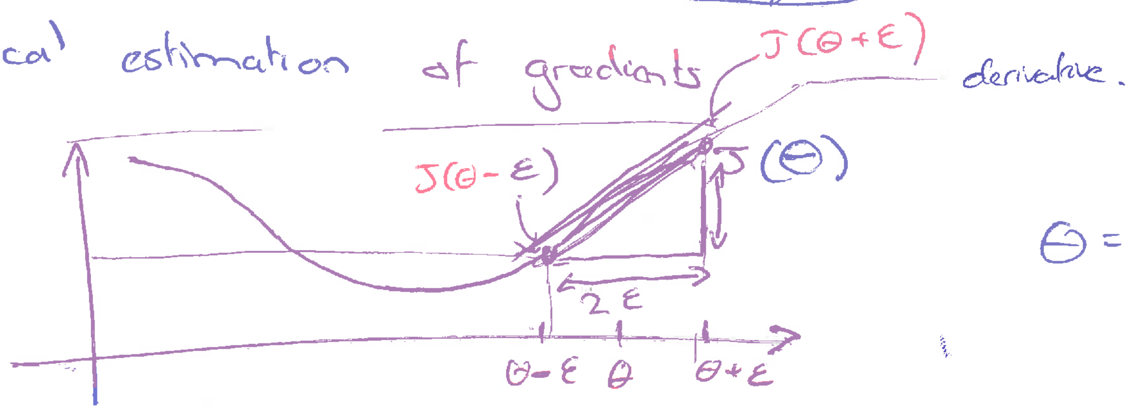
Unroll $\boxed{D^{(1)}, D^{(2)}, D^{(3)}}$ to get $\boxed{\text{gradientVec}}$

forward prop & BP to compute derivatives & cost function.

Notes - Week 5

Neural Networks: learning: gradient checking

Numerical estimation of gradients



$\Theta = \mathbb{R}$

Slope of the little line is the approximation of the derivative.

$\frac{d}{d\Theta} J(\Theta) \approx$ the approximation is ∴ given by:

$$\frac{J(\Theta+\varepsilon) - J(\Theta-\varepsilon)}{2\varepsilon}$$

"two sided difference" for estimating the derivative

$\varepsilon = 10^{-4}$

with a small ε this becomes very close to the derivative

"one sided..."

$$\cancel{\frac{J(\Theta+\varepsilon) - J(\Theta)}{\varepsilon}}$$

Implementation: $\boxed{\text{gradApprox}}$ = (J(theta + EPSILON) - J(theta - EPSILON)) / (2 * EPSILON).

Parameter vector Θ

$\Theta \in \mathbb{R}^n$ (eg Θ is "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)

$\Theta = [\Theta_1, \Theta_2, \Theta_3, \dots \Theta_n]$

$$\frac{\partial}{\partial\Theta_1} J(\Theta) \approx \frac{J(\boxed{\Theta_1+\varepsilon}, \Theta_2, \Theta_3, \dots, \Theta_n) - J(\boxed{\Theta_1 - \varepsilon}, \Theta_2, \Theta_3, \dots, \Theta_n)}{2\varepsilon}$$

$\vdots$

$$\frac{\partial}{\partial\Theta_n} \quad (\Theta_1, (\Theta_2+\varepsilon) \quad) - J(\Theta_1, (\Theta_2-\varepsilon))$$

Octave code:

... usually with unrolled version of theta,

```
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;
```

$\xrightarrow{\text{roughly equal to}}$ $\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i + \epsilon \\ \vdots \\ \theta_n \end{bmatrix}$

check that $\boxed{\text{Gradapprox}}$ is roughly $\approx$ to 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 have similar values.
- Turn off gradient checking. Using backprop code for learning.

Important

- disable gradient checking code before training classifier
  ↳ otherwise code will be very slow.

Notes - Week 5

Neural Networks learning: Random initialization

Initial value of Θ

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

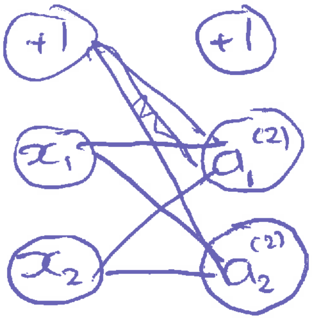
optTheta = fminunc(@costFunction, initialTheta, options)

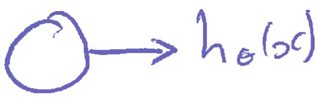
Consider gradient descent:

Set initialTheta = zeros(n, 1) $\boxed{?}$

→ this does not work when training a neural network.

Zero initialization





$a_1^{(2)} = a^2$ if $\Theta_{ij}^{(l)} = 0$ for all i, j, l.

Also $\delta_1^{(2)} = \delta_2^{(2)}$

$$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)$$

∴ even after gradient descent $\Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

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

Notes - Week 5

Neural Networks Learning: Random initialization: Symmetry breaking

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

→ random 10x11 matrix (between 0 e 1)

different ϵ than gradient checking

E.g./ Theta1 = rand(10, 11) * (2 * INIT_EPSILON)
- INIT_EPSILON;

→ $[-\epsilon, \epsilon]$

Theta2 = rand(1, 11) * (2 * INIT_EPSILON) - INIT_EPSILON;

Notes – Week 5:

Neural Networks Learning: Putting it All together

Training a neural network:

– Pick a neural network architecture (connectivity pattern between the neurons)

- No of input units: Dimension of features $x^{(i)}$ ✓
- No output units: number of classes ✓

$$y \in \{1, 2, 3, \ldots, 10\} \qquad y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$y = 5$$

- Reasonable default: 1 hidden layer or if >1 hidden layer, have same no of hidden units in every layer (usually the more the better)

Training a neural network

1.) Randomly initialize weights
2) Implement forward propagation to get $h_\theta(x^{(i)})$ for any $x^{(i)}$
3) Implement code to complete cost function $J(\theta)$
4) Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta^{l}_{jk}} J(\Theta)$.

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 $l = 2, \ldots, L$)

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

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

5) Use gradient checking to compare $\frac{\partial}{\partial \Theta^{l}_{jk}} J(\theta)$ computed using backpropagation vs. using numerical estimate 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. $J(\Theta)$ – non convex ∴ susceptible to local minima.

"back propagation" predicts the direction of the gradient as gradient descent moves towards a local optimum"