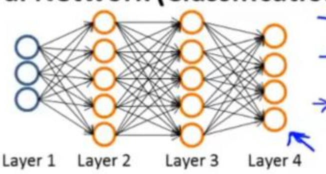
Neural Networks: Learning

1 Cost function

Neural Network (Classification)



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

$L =$ total no. of layers in network $\quad L=4$

$s_l =$ no. of units (not counting bias unit) in layer l $\quad s_1=3,\ s_2=5,\ s_4=s_L=4$

Binary classification

$y = 0$ or 1

$h_\Theta(x)$

1 output unit

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

$s_L = 1$, $\quad K=1$

Multi-class classification (K classes)

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

pedestrian car motorcycle truck

K output units

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

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

Andrew Ng

Cost function

Logistic regression:

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

θ_0

Neural network:

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

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

$\Theta_{ji}^{(l)}$

$\Theta_{i0}^{(2)}x_0 + \Theta_{i1}^{(2)}x_1 + \dots$

a_0

$i = 0 \dots s_l$

y_k $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$

Andrew Ng

2 Backpropagation algorithm

Gradient computation

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

$$\min_\Theta J(\Theta)$$

Need code to compute:

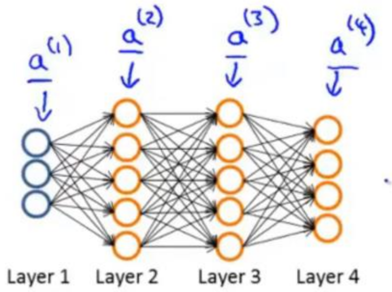
- $J(\Theta)$
- $\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)$

$\Theta_{ij}^{(l)}\in\mathbb{R}$

Gradient computation

Given one training example (x, y):

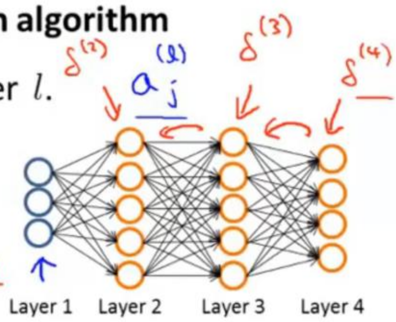
Forward propagation:



$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)}a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)}a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)}a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$

Gradient computation: Backpropagation algorithm

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



For each output unit (layer L = 4)

$$\boxed{\delta_j^{(4)} = \boxed{a_j^{(4)}} - y_j} \quad (h_\Theta(x))_j \quad \delta^{(4)} = a^{(4)} - y$$

$$\delta^{(3)} = (\Theta^{(3)})^T\delta^{(4)} .* g'(z^{(3)}) \qquad a^{(3)} .* (1 - a^{(3)})$$
$$\delta^{(2)} = (\Theta^{(2)})^T\delta^{(3)} .* g'(z^{(2)}) \qquad a^{(2)} .* (1 - a^{(2)})$$

(No $\delta^{(1)}$)

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)}\delta_i^{(l+1)} \quad (\text{ignoring } \lambda;\ \text{if } \lambda = 0)$$

Backpropagation algorithm

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

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

For $i = 1$ to m $(x^{(i)}, y^{(i)})$.

- Set $a^{(1)} = x^{(i)}$
- Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$
- Using $y^{(i)}$, compute $\delta^{(L)} = \boxed{a^{(L)}} - \boxed{y^{(i)}}$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ $\cancel{\delta^{(1)}}$
- $\boxed{\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)}\delta_i^{(l+1)}}$ $\quad \triangle^{(l)} := \triangle^{(l)} + \delta^{(l+1)}(a^{(l)})^T.$

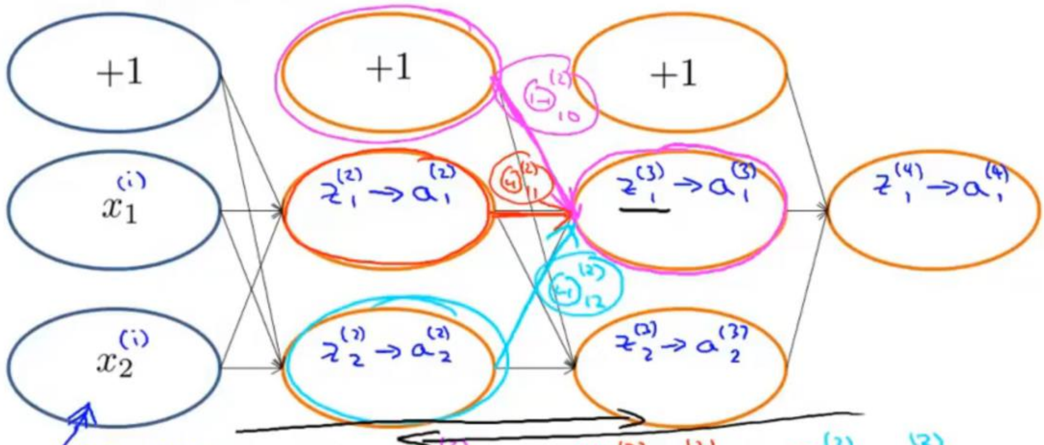
$\boxed{D_{ij}^{(l)}} := \frac{1}{m}\triangle_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)}$ if $j \neq 0$

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

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

3 Backpropagation intuition

Forward Propagation



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

What is backpropagation doing?

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

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

$(x^{(i)}, y^{(i)})$

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

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

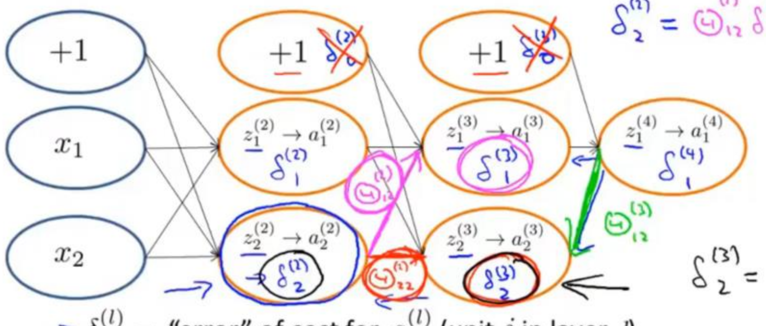
(Think of $\mathrm{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)

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

Forward Propagation

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

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



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

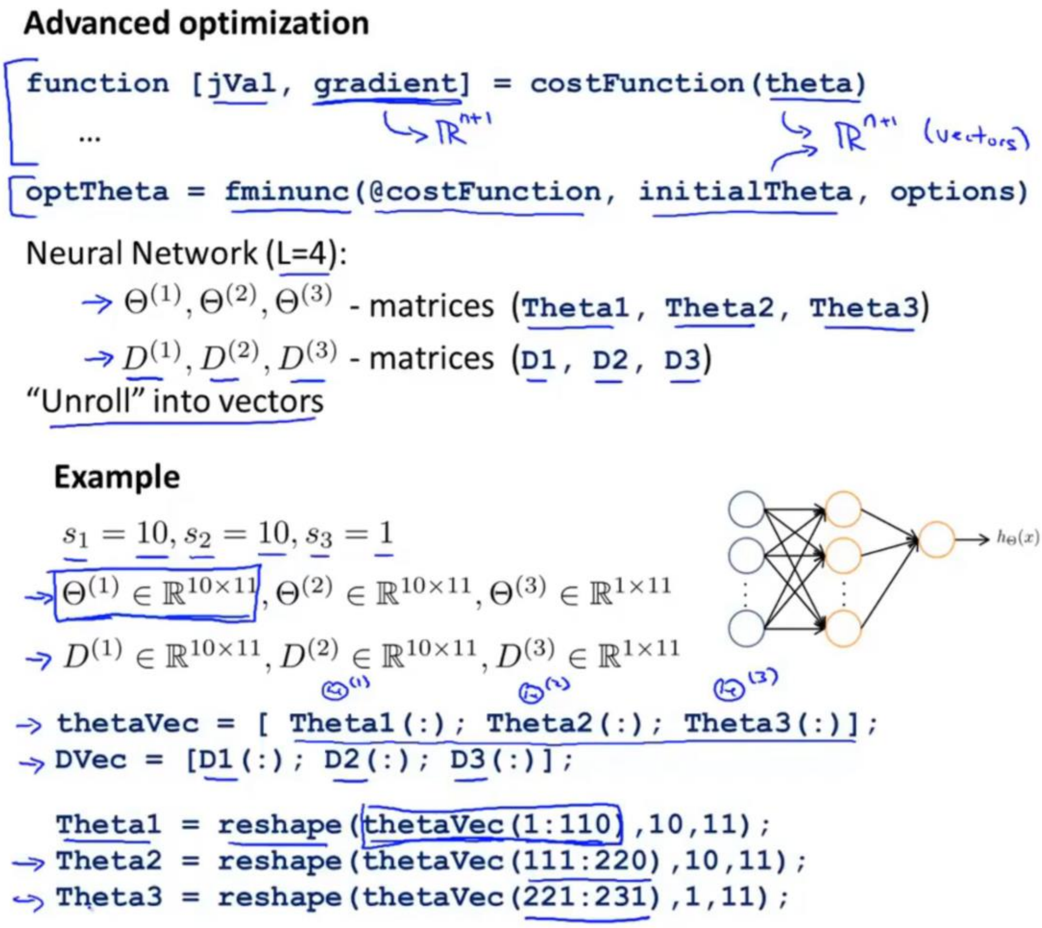
$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l).

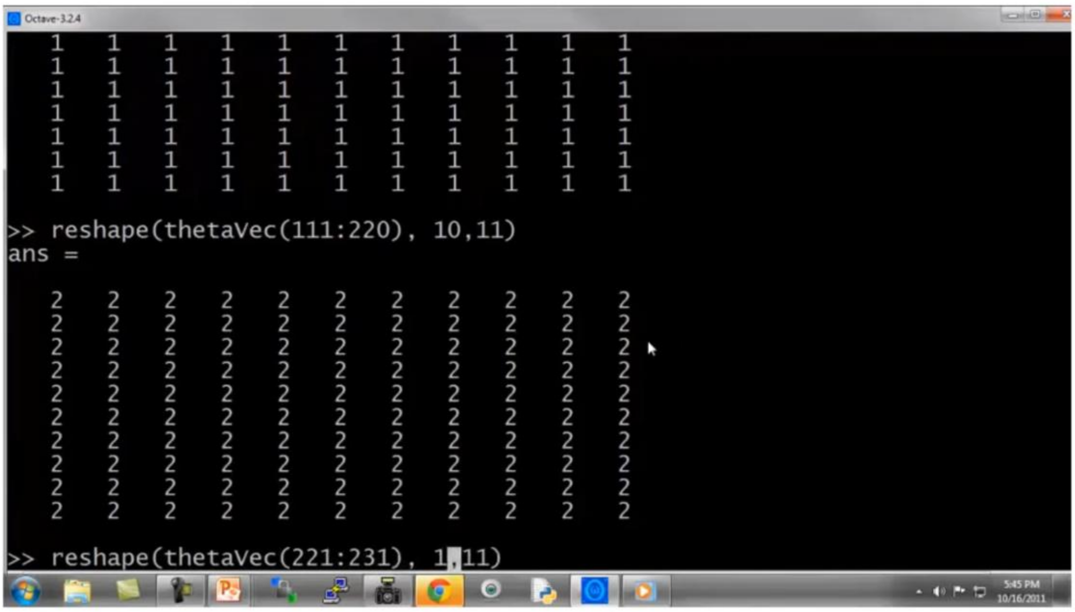
Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \mathrm{cost(i)}$ (for $j \geq 0$), where

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

Andrew Ng

4 Implementation note: Unrolling parameters





Learning Algorithm

- Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.
- Unroll to get `initialTheta` to pass to
- `fminunc(@costFunction, initialTheta, options)`

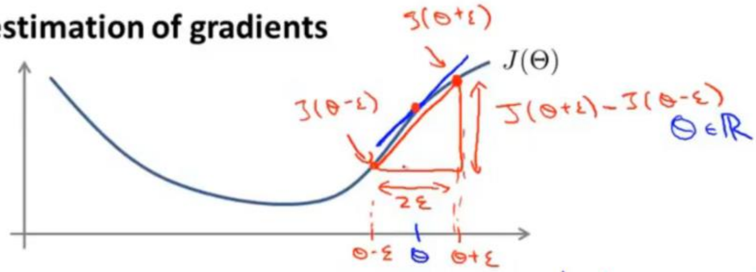
```
function [jval, gradientVec] = costFunction(thetaVec)
```

- From `thetaVec`, get $\boxed{\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}}$ 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`.

5 Gradient checking

Numerical estimation of gradients



$$\frac{d}{d\theta} J(\theta) \approx \underbrace{\frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}}$$

$\epsilon = 10^{-4}$

$$\cancel{\frac{J(\theta+\epsilon) - J(\theta)}{\epsilon}}$$

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 $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)
- $\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]$

$$\frac{\partial}{\partial\theta_1} J(\theta) \approx \frac{J(\boxed{\theta_1+\epsilon},\theta_2,\theta_3,\ldots,\theta_n) - J(\boxed{\theta_1-\epsilon},\theta_2,\theta_3,\ldots,\theta_n)}{2\epsilon}$$

$$\frac{\partial}{\partial\theta_2} J(\theta) \approx \frac{J(\theta_1,\boxed{\theta_2+\epsilon},\theta_3,\ldots,\theta_n) - J(\theta_1,\boxed{\theta_2-\epsilon},\theta_3,\ldots,\theta_n)}{2\epsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial\boxed{\theta_n}} J(\theta) \approx \frac{J(\theta_1,\theta_2,\theta_3,\ldots,\boxed{\theta_n+\epsilon}) - J(\theta_1,\theta_2,\theta_3,\ldots,\boxed{\theta_n-\epsilon})}{2\epsilon}$$

```
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;
```

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i + \varepsilon \\ \vdots \\ \theta_n \end{bmatrix} \rightarrow \theta_i - \varepsilon$$

$\frac{\partial}{\partial \theta_i} J(\theta)$.

Check that `gradApprox` ≈ `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 code for learning.

$\hookrightarrow$ DVec

$\delta^{(4)}, \delta^{(3)}, \delta^{(2)}$

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.

6 Random initialization

Initial value of Θ

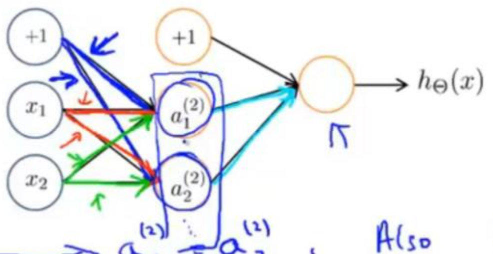
For gradient descent and advanced optimization method, need initial value for Θ.

```
optTheta = fminunc(@costFunction,
           initialTheta, options)
```

Consider gradient descent

Set `initialTheta = zeros(n,1)` ?

Zero initialization



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

$a_1^{(2)} = a_2^{(2)}$. Also $\delta_1^{(2)} = \delta_2^{(2)}$.

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

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

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

Random initialization: Symmetry breaking

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

E.g. random 10×11 matrix (betw. 0 and 1)

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

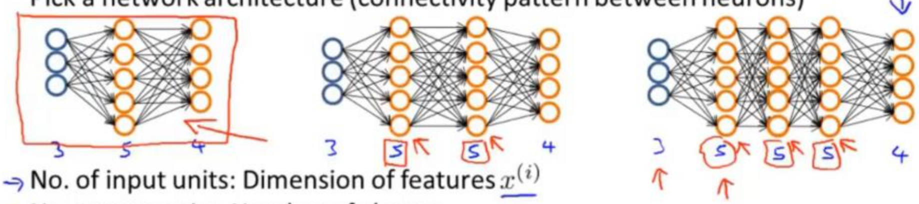
$[-\epsilon, \epsilon]$

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

7 Putting it together

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



- No. of input units: Dimension of features $x^{(i)}$
- No. output units: Number of classes
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

$y \in \{1, 2, 3, \dots, 10\}$ $\qquad$ ~~y=5~~

$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ or $\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ $\qquad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

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 compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\boxed{\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)}$

```
for i = 1:m {
```

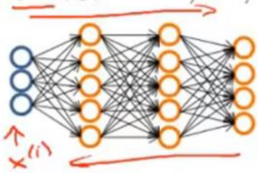
$(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$, ... $(x^{(m)}, y^{(m)})$

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



An

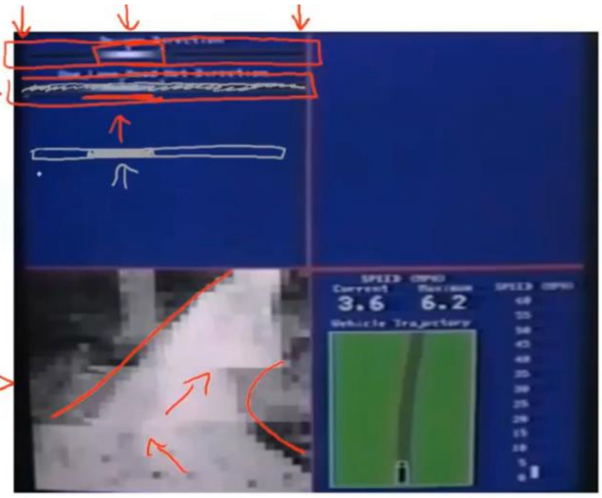
Training a neural network

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

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

$J(\Theta)$ — non-convex.

8 Autonomous driving example



[Courtesy of Dean Pomerleau]