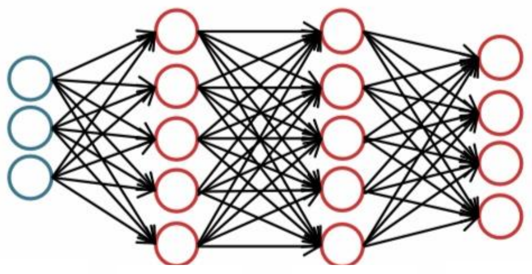
Machine Learning

By Ghazal Lalooha

Artificial Neural Networks Training

Neural Networks (classification)



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

Total number of layers in the neural network = L

Number of units (without considering bias) in the layer l = S_l

Multiclass classification (k class)

$y \in \mathbb{R}^K$

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

Passerby car motorcycle truck

K unit output

Binary classification

$y \in \{0,1\}$

one unit output

Cost Function

Logistic regression:

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

Neural network:

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

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

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

Back propagation algorithm

Computing gradient

Cost function:

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

Goal:

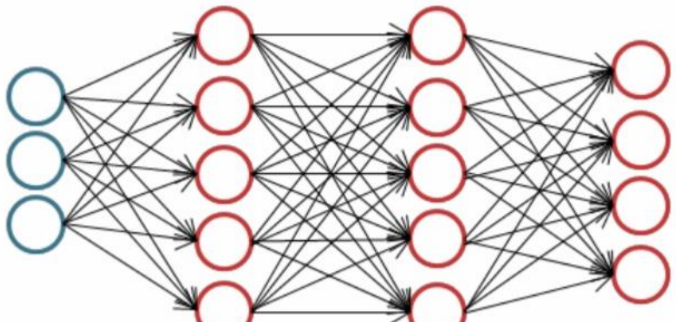
$$\min_{\Theta} J(\Theta)$$

Quantities which should be computed:

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

Computing gradient

- Having a training sample (x,y)



Forward propagation

$$a^{(1)} = x$$

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

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

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

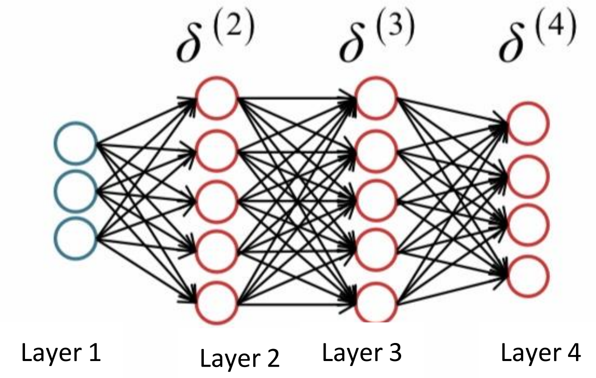
$$a^{(3)} = g(z^{(3)}) \quad \left(\text{add } a_0^{(3)}\right)$$

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

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

Computing gradient : Back propagation algorithm

The error of node j in the layer l $= \delta_j^{(l)}$



Error for output units: $(l = 4)$

$$\delta^{(4)} = \left(y - a^{(4)}\right) .\times\ g'\left(z^{(4)}\right)$$

Error for hidden units: $(l = 2,3)$

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

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

Back propagation algorithm

Training set:

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

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

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

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

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)}\delta_i^{(l+1)}$

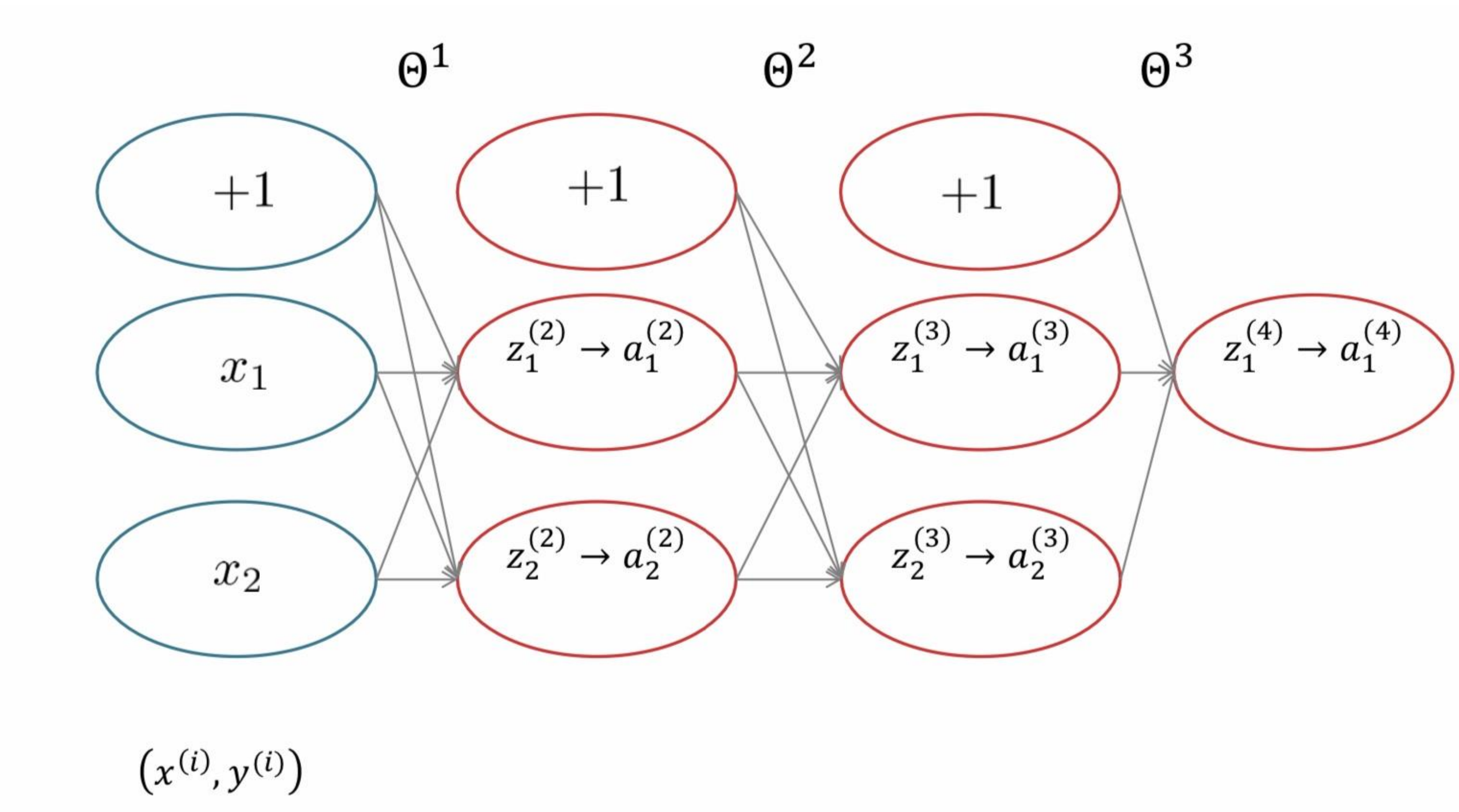
$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)}$ if $j \neq 0$

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

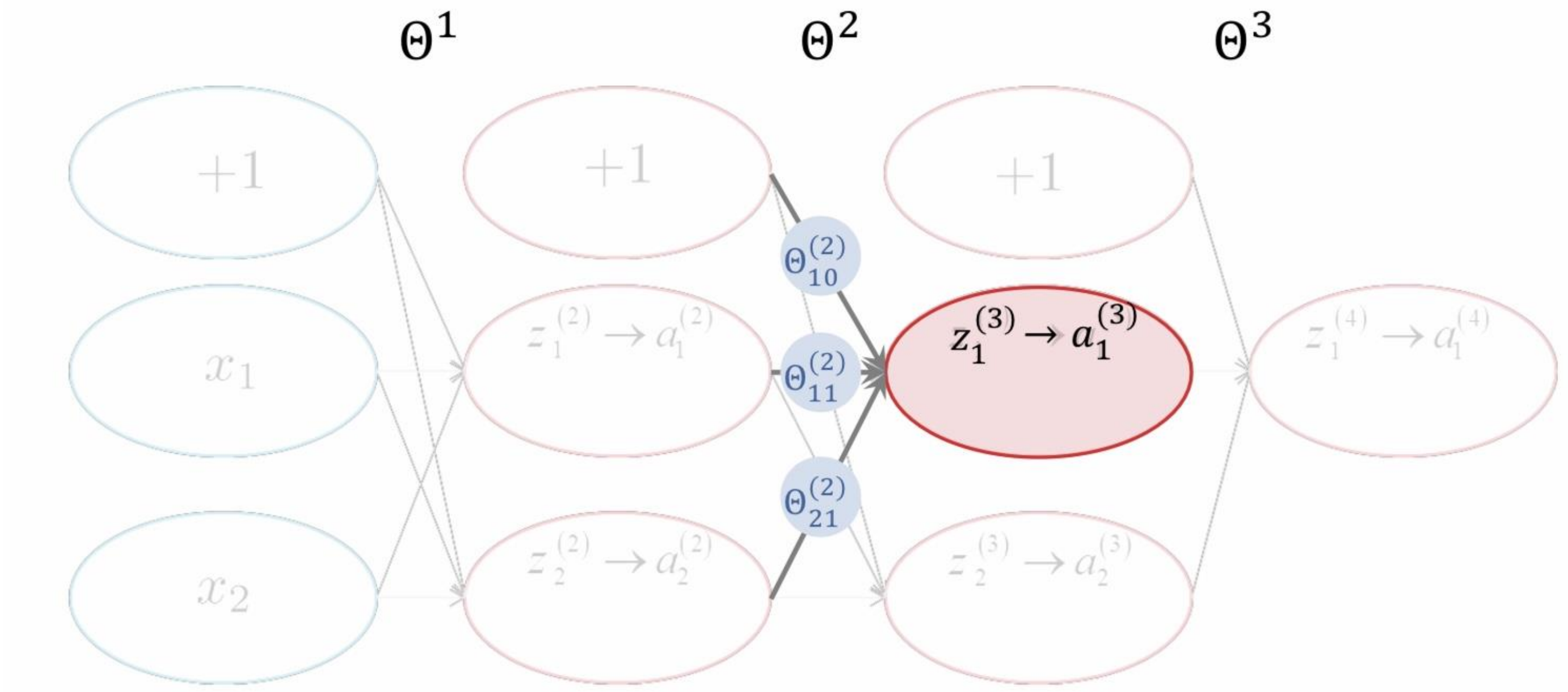
$$\boxed{\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)}}$$

Back propagation algorithm : visual definition

Forward propagation



Forward propagation



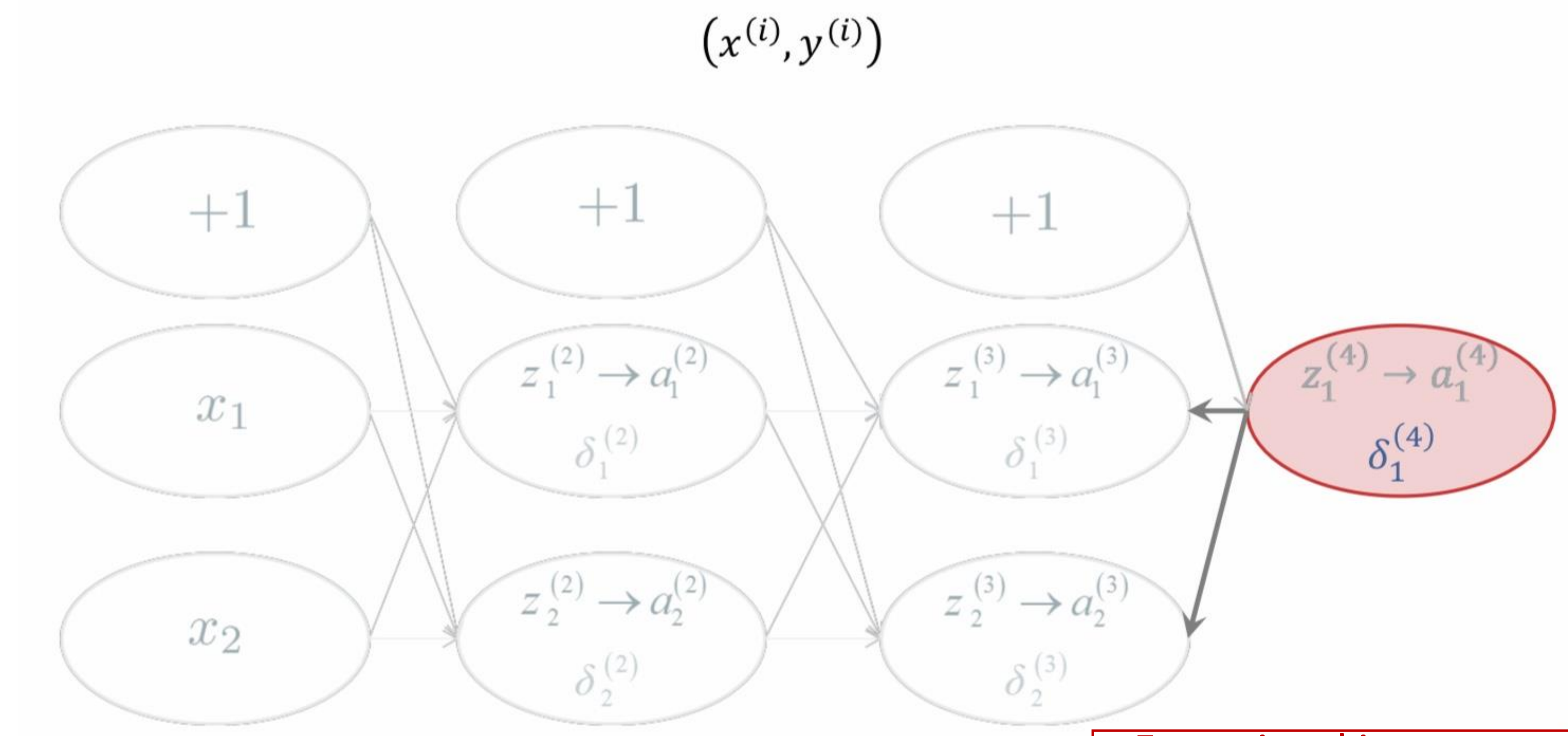
Calculate the weighted sum of inputs:

$$z_1^{(3)} = \Theta_{10}^{(2)} \times a_0^{(2)} + \Theta_{11}^{(2)} \times a_1^{(2)} + \Theta_{12}^{(2)} \times a_2^{(2)}$$

Apply a non-linear function:

$$a_1^{(3)} = g\left(z_1^{(3)}\right)$$

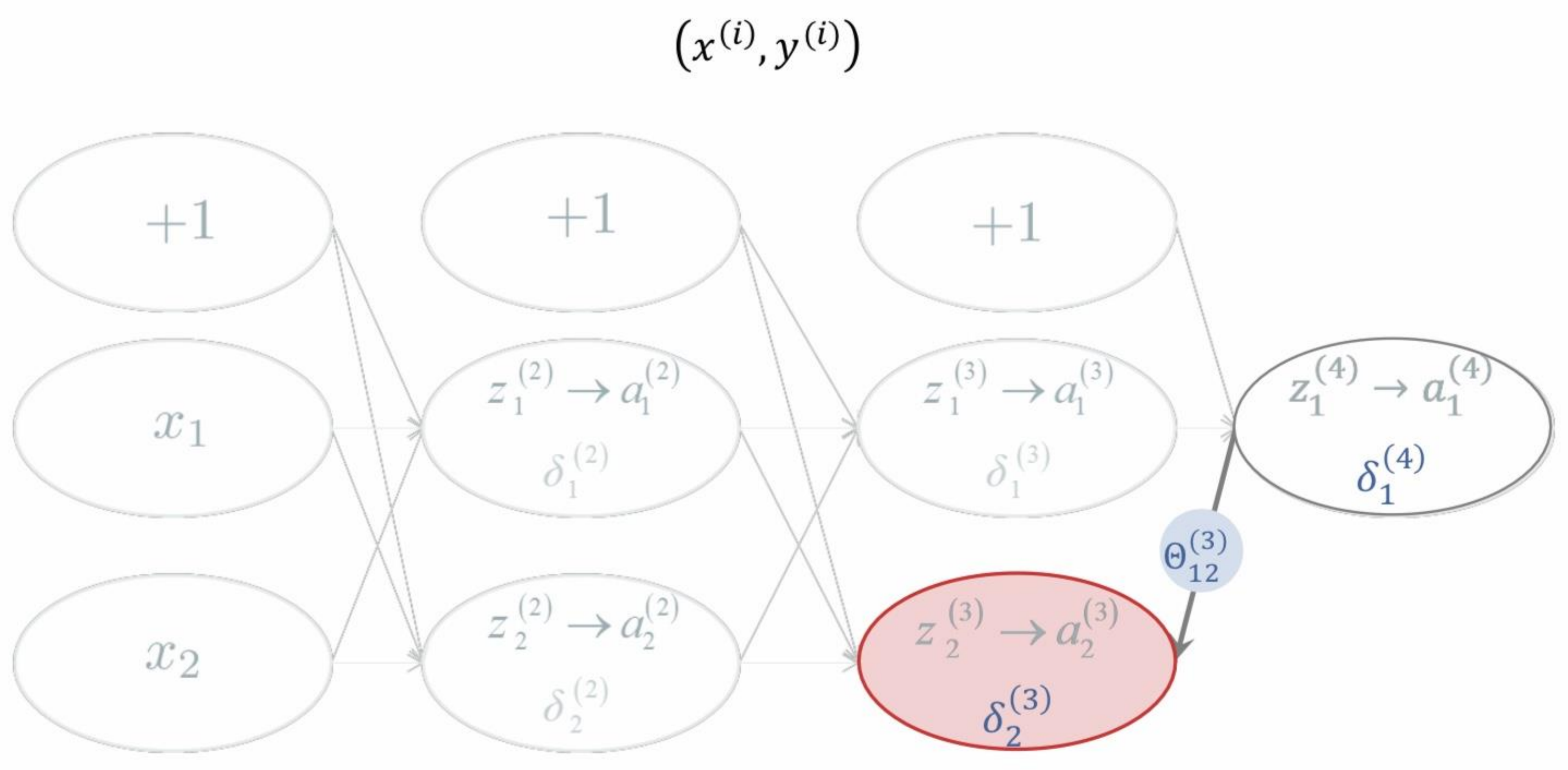
Back propagation



Error signal in output neuron:

$$\delta_1^{(4)} = \left(y^{(i)} - a_1^{(4)}\right) \times g'(z_1^4)$$

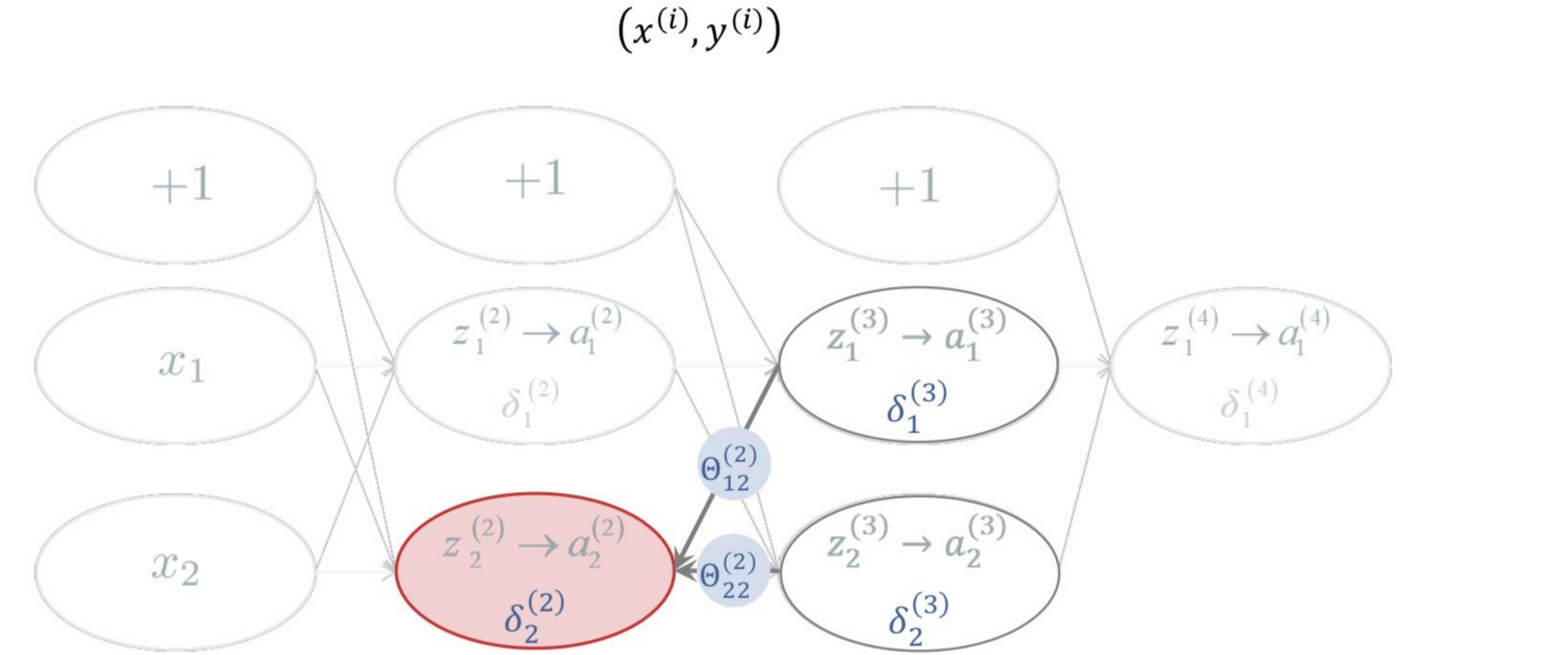
Back propagation



$$\delta^{(3)} = \left(\left(\Theta^{(3)}\right)^T * \delta^{(4)}\right) .\times g'\left(z^{(3)}\right)$$

$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)} \times g'\left(z_2^{(3)}\right)$$

Back propagation



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

$$\delta_2^{(2)} = \left(\Theta_{12}^{(2)} \times \delta_1^{(3)} + \Theta_{12}^{(2)} \times \delta_2^{(3)}\right) \times g'\left(z_2^{(2)}\right)$$

Points about BP implementation

Advanced optimization

```
function [jVal, gradient] = costFunction(theta)
...
optTheta = fminunc(@costFunction, initialTheta, options);
```

$\in \mathbb{R}^{n+1}$ $\in \mathbb{R}^{n+1}$

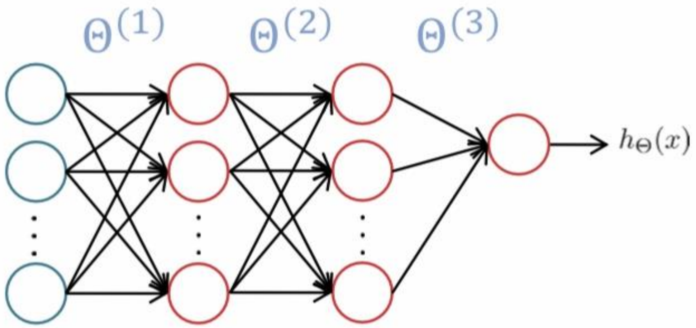
- Parameters in a 4-layer network (2 hidden layers):
  - weight matrices (theta1, theta2, theta3)
  - Weight change matrices (D1, D2, D3)
- To use advanced optimization methods, all three matrices must be converted into a vector.

Example

$s_1 = 10 \quad s_2 = 10 \quad s_3 = 10 \quad s_4 = 1$

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

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




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

Converting matrices to vector

```
Theta1 = reshape(thetaVec(  1:110), 10, 11);
Theta2 = reshape(thetaVec(111:220), 10, 11);
Theta3 = reshape(thetaVec(221:231),  1, 11);
```

Converting vectors to matrix

Learning algorithm

- Having the initial parameters of theta1, theta2 and theta3
  - Convert these matrices to the vector initialTheta so that you can pass it as an argument to the following function:

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

Then write cost function in this way:

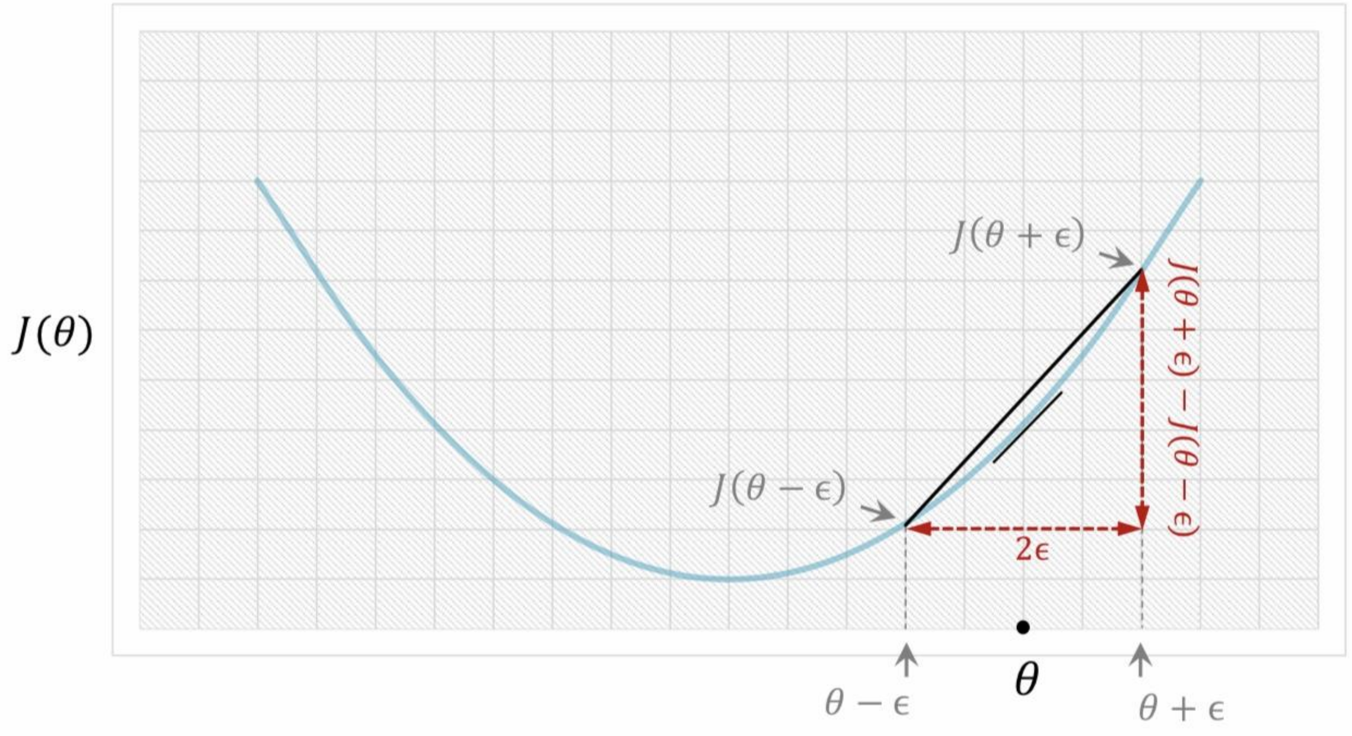
```
function [jVal, gradientVec] = costFunction(thetaVec)
```

  - Retrieve the theta1, theta2, and theta3 matrices from the thetaVec vector.
  - Calculate matrices D1, D2, D3 using forward propagation $j(\theta)$ and backpropagation.
  - Convert matrices D1, D2, D3 to vector gradientVec.

Check the gradient

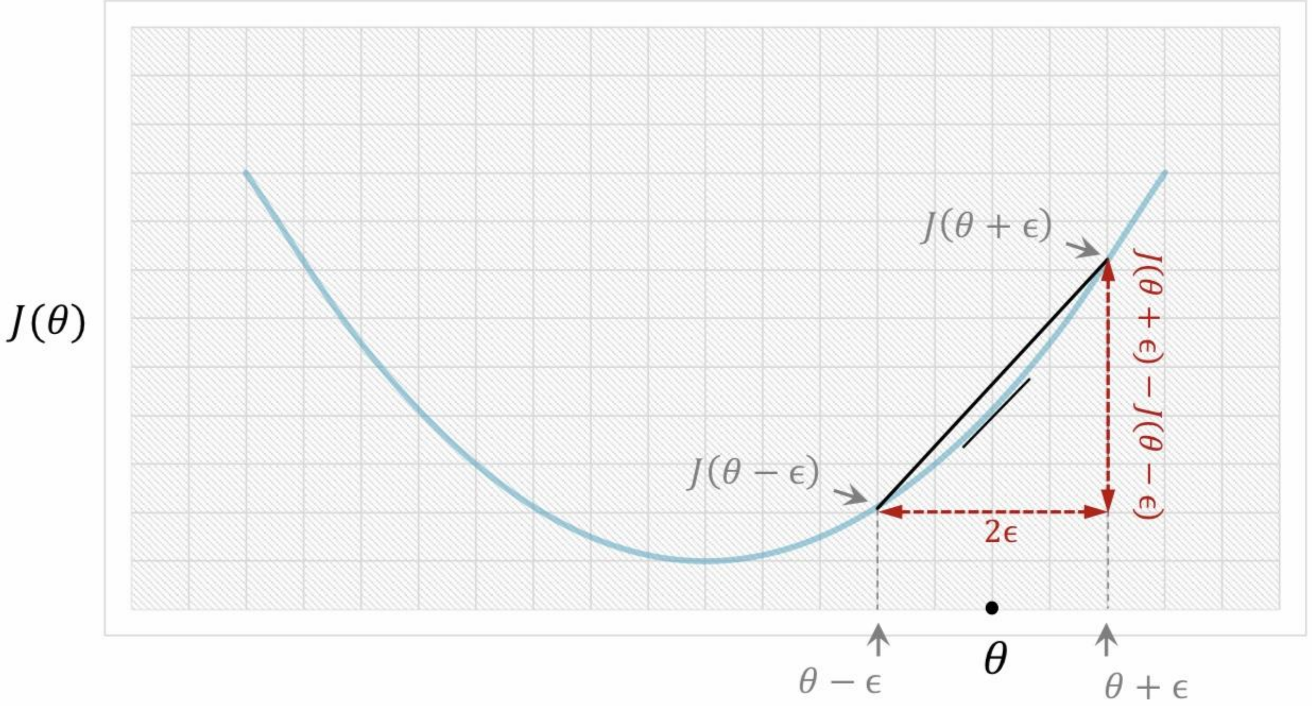
- How can we be sure that we have correctly implemented the error propagation algorithm?

Estimating the number of gradients (univariate cost function)



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

Estimating the number of gradients (univariate cost function)




```
gradApprox = (J(theta + EPSILON) - J(theta - EPSLION)) / (2 * EPSILON);
```

Estimating the number of gradients (univariate cost function)

$\theta \in \mathbb{R}^n \qquad \theta = \theta_1, \theta_2, \theta_3, \dots, \theta_n \qquad (\theta \leftarrow \Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)})$

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

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

$$\vdots$$

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

Implementation: Numerical Calculation of gradient

```
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;
```

Make sure: `DVec ≈ gradApprox`

Implementation points:

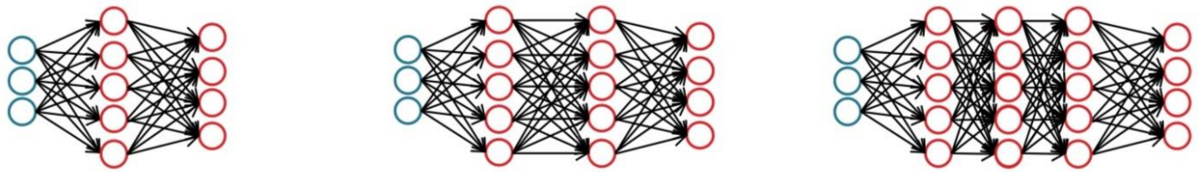
- Implement the error backpropagation algorithm to calculate the DVec vector.
- Implement the gradient number calculation function to calculate gradApprox.
- Make sure these two vectors contain the same values.
- Disable the gradient check function.
- Use backpropagation algorithm to train neural network.

- Before starting the neural network training process, make sure you disable the gradient check function, otherwise your program will run very slowly.

Conclusion

A neural network training

- Choosing an architecture for the network (pattern of connections between neurons)



- Determining the number of layers and the number of neurons in each layer
  - Number of input units: equal to the number of features
  - Number of output units: equal to the number of classes
  - Number of hidden layers: It is usually equal to one, but if there is more than one hidden layer, it is better to have the same number of neurons in these hidden layers.

A neural network implementation

- Random assignment to weights
- Implement a forward propagation step to calculate the output of the network for each input such as $x^{(i)}$
- Implement a function to calculate the value of the cost function $j(\theta)$
- Implementation of error backpropagation stage to calculate partial differentials

```
for i = 1 : m {
     perform forward propagation and backpropagation using example (x^(i), y^(i))
     (Get activations a^(l) and delta terms δ^(l) for l = 2,…, L)
     compute Δ^(l) = Δ^(l) + δ^(l) (a^(l))^T
}
compute partial derivatives of  J(Θ) considering regularization term
```

A neural network implementation

- Check the gradient:
- Implementing a function in order to numerically calculate the values of the gradients and compare these values with the values calculated by the error back propagation algorithm.
- Disabling the function of checking gradients
- Optimization:
- Using the decreasing gradient method or one of the advanced optimization methods along with the error back propagation algorithm in order to try to minimize the cost function $J(\theta)$ as a function of theta parameters.
- Due to the "non-convexity" of the cost function, gradient descent or any of the advanced optimization methods may get stuck at local optima.

Example: Autonomous driving

Autonomous driving