Elective Course

Course Code: CS4103

Autumn 2025-26

Lecture #40

Artificial Intelligence for Data Science

Week-11:

MACHINE LEARNING (Part VIII)

Neural Network Learning

Course Instructor:

Dr. Monidipa Das

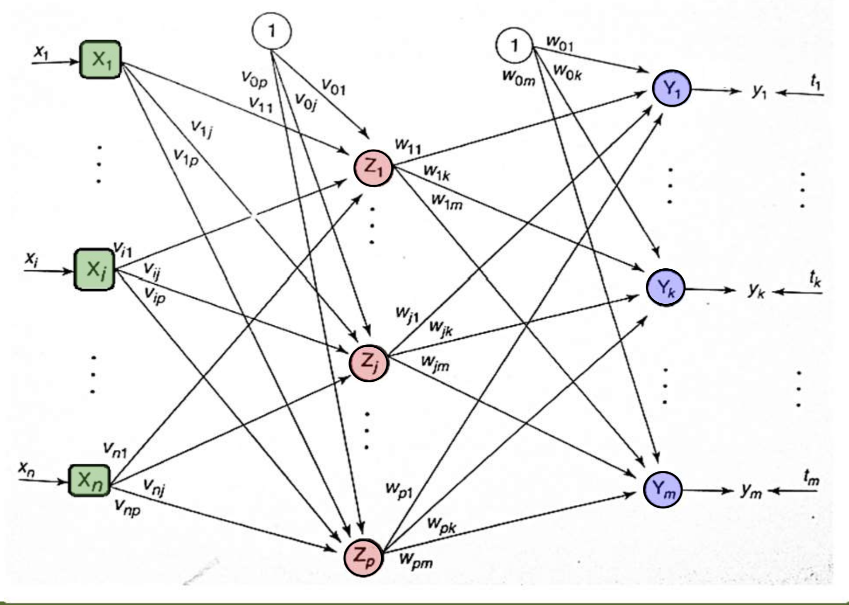
Assistant Professor

Department of Computational and Data Sciences

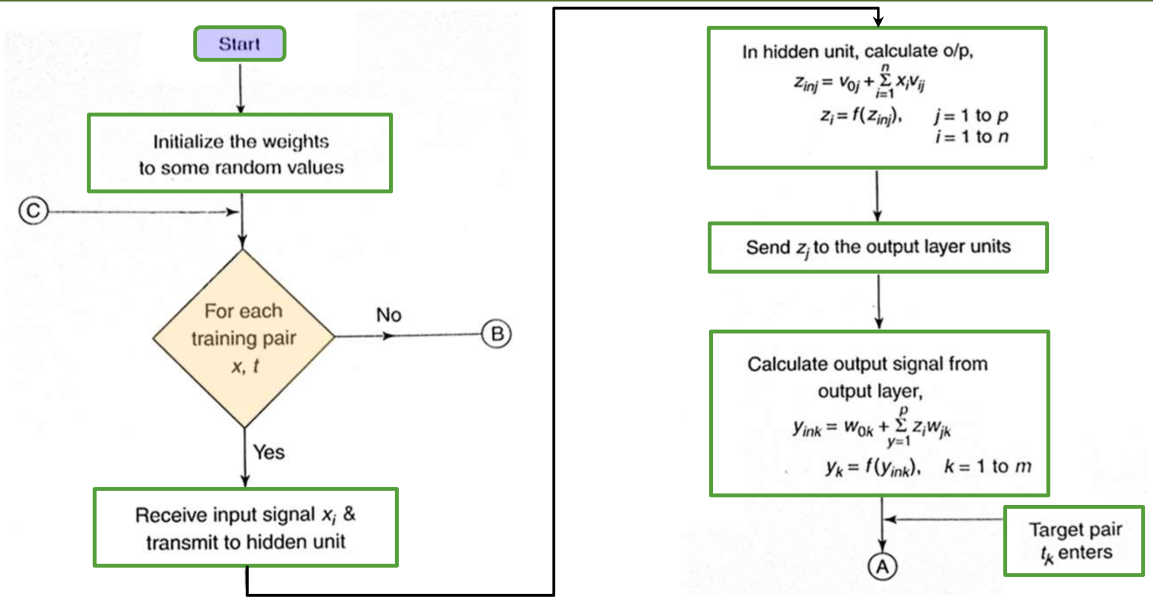
Indian Institute of Science Education and Research Kolkata, India 741246

Back Propagation Network (BPN)

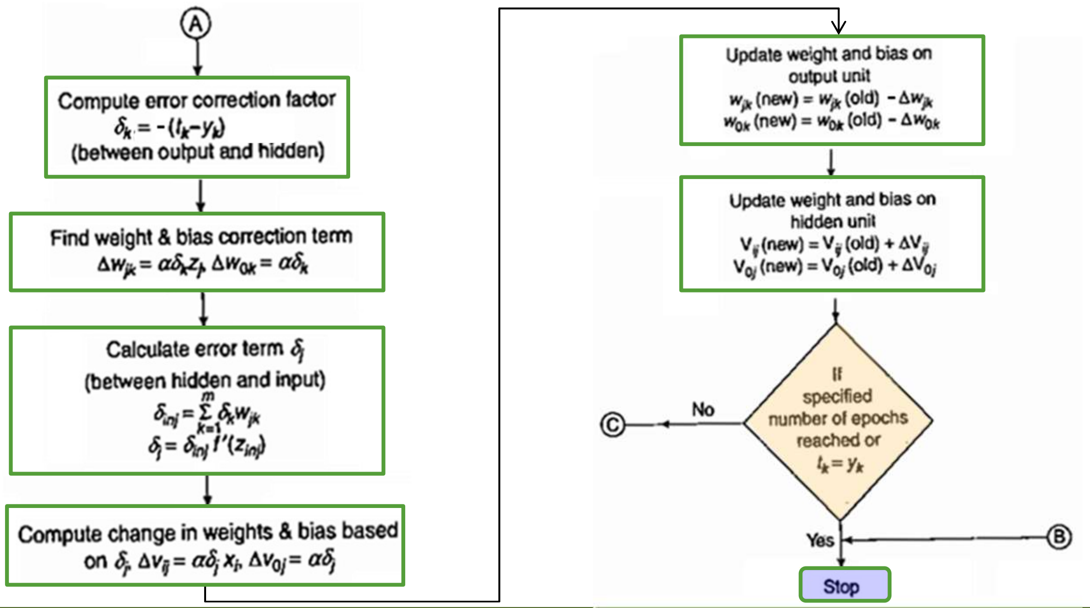
- Architecture



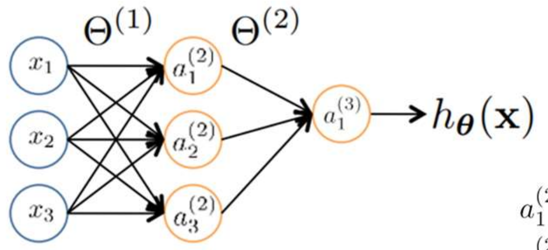
BPN Training


Start
Initialize the weights to some random values
C
For each training pair x, t
No
B
Yes
Receive input signal x_i & transmit to hidden unit
In hidden unit, calculate o/p,
$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$
$z_j = f(z_{inj}), \quad j = 1$ to p, $i = 1$ to n
Send z_j to the output layer units
Calculate output signal from output layer,
$y_{ink} = w_{0k} + \sum_{y=1}^{p} z_j w_{jk}$
$y_k = f(y_{ink}), \quad k = 1$ to m
Target pair t_k enters
A

BPN Training


A
Compute error correction factor $\delta_k = -(t_k - y_k)$ (between output and hidden)
Find weight & bias correction term $\Delta w_{jk} = \alpha \delta_k z_j$, $\Delta w_{0k} = \alpha \delta_k$
Calculate error term δ_j (between hidden and input)
$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$
$\delta_j = \delta_{inj} f'(z_{inj})$
Compute change in weights & bias based on δ_j, $\Delta v_{ij} = \alpha \delta_j x_i$, $\Delta v_{0j} = \alpha \delta_j$
Update weight and bias on output unit
w_{jk} (new) = w_{jk} (old) $- \Delta w_{jk}$
w_{0k} (new) = w_{0k} (old) $- \Delta w_{0k}$
Update weight and bias on hidden unit
V_{ij} (new) = V_{ij} (old) $+ \Delta V_{ij}$
V_{0j} (new) = V_{0j} (old) $+ \Delta V_{0j}$
If specified number of epochs reached or $t_k = y_k$
No
C
Yes
B
Stop

Neural Network



$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ weight matrix controlling function mapping from layer j to layer $j+1$

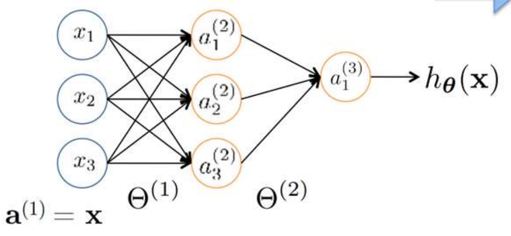
$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times4}$$

Vectorization

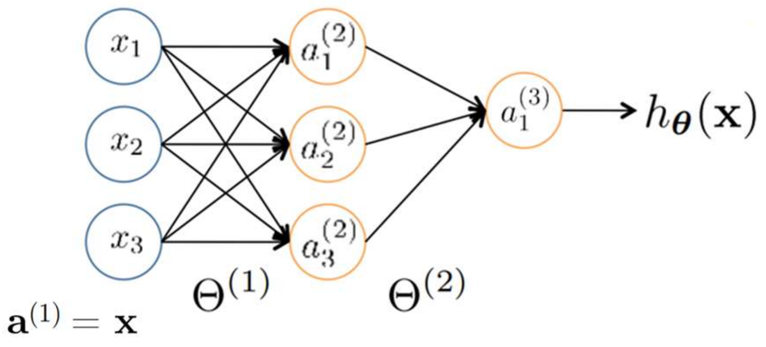
$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$
$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$
$$\text{Add } a_0^{(2)} = 1$$
$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$
$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Forward Propagation



Feed-Forward Steps:

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

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

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

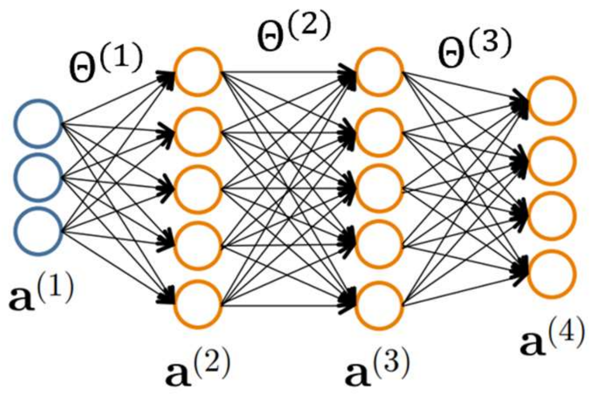
$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Forward Propagation

- Given one labeled training instance $(\mathbf{x},\ y)$:

Forward Propagation

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$



Cost Function

Neural Network:

$h_\Theta \in \mathbb{R}^K$ $\qquad (h_\Theta(\mathbf{x}_i))_k = k^{th}\text{output}$ of the i-th sample

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log\left(h_\Theta(\mathbf{x}_i)\right)_k + (1-y_{ik})\log\left(1-(h_\Theta(\mathbf{x}_i))_k\right)\right]$$

k^{th} class: true, predicted
not k^{th} class: true, predicted

Optimizing the Neural Network

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

Solve via: $\min_{\Theta} J(\Theta)$

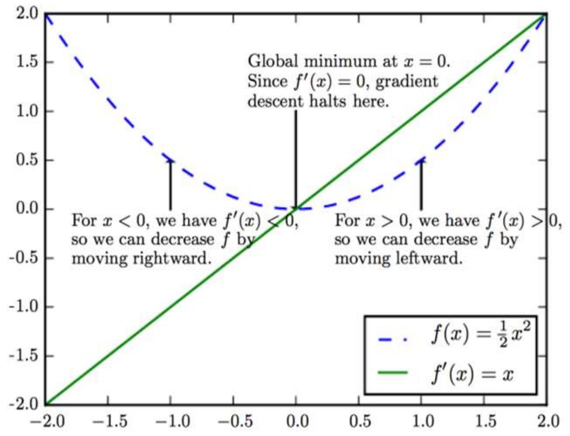
We can use Gradient Descent (GD)

Need code to compute:
- $J(\Theta)$
- $\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)$

How Gradient Descent (GD) works using derivatives

- Criterion $f(x)$ minimized by moving from current solution in direction of the negative of gradient

- For $x>0$, $f(x)$ increases with x and $f'(x)>0$
- For $x<0$, $f(x)$ decreases with x and $f'(x)<0$
- Use $f'(x)$ to follow function downhill
- Reduce $f(x)$ by going in direction opposite sign of derivative $f'(x)$



Method of Gradient Descent

- Decrease f by moving in the direction of the negative gradient
  - This is known as the method of ***gradient descent***
- The method proposes a new point $\boldsymbol{x}' = \boldsymbol{x} - \alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
  - where α is the ***learning rate***, a positive scalar. Set to a small constant.
- This converges when every element of the gradient is zero
  - In practice, very close to zero
- We may be able to avoid iterative algorithm and jump to the critical point by solving the equation $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$ for x

Gradient Descent Algorithm

Gradient Descent: Algorithmic representation

$$
\begin{aligned}
&\text{repeat until convergence } \{\\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \text{(simultaneously update } j = 0 \text{ and } j = 1)\\
&\}
\end{aligned}
$$

Parameters

Cost function, which is to be minimized

Correct updating of parameters

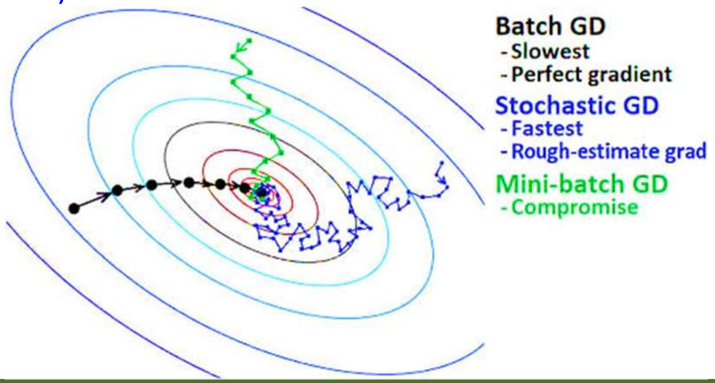
$$
\begin{aligned}
&\text{temp0} := \theta_0 - \alpha \tfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)\\
&\text{temp1} := \theta_1 - \alpha \tfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)\\
&\theta_0 := \text{temp0}\\
&\theta_1 := \text{temp1}
\end{aligned}
$$

Incorrect updating of parameters

$$
\begin{aligned}
&\text{temp0} := \theta_0 - \alpha \tfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)\\
&\theta_0 := \text{temp0}\\
&\text{temp1} := \theta_1 - \alpha \tfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)\\
&\theta_1 := \text{temp1}
\end{aligned}
$$

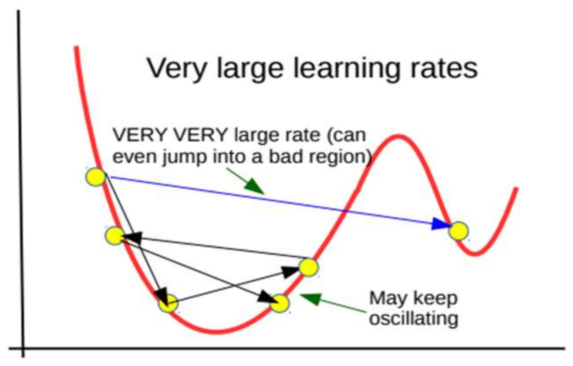
Types of Gradient Descent

- There are **three major types** of Gradient Descent Algorithms:
  - Batch Gradient Descent (BGD)
  - Stochastic Gradient Descent (SGD)
  - Mini-Batch Gradient Descent (MBGD)

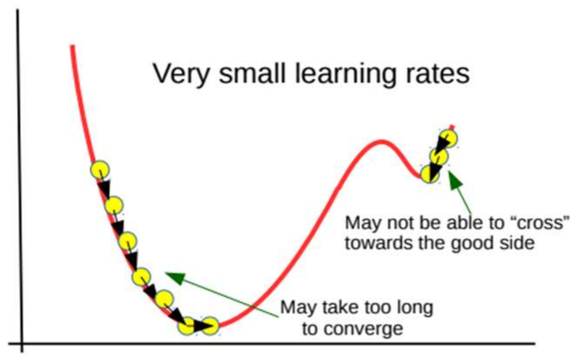


Learning rate is very important

15


Very large learning rates
VERY VERY large rate (can even jump into a bad region)
May keep oscillating

Too large, the next point will perpetually bounce haphazardly across the bottom of the well

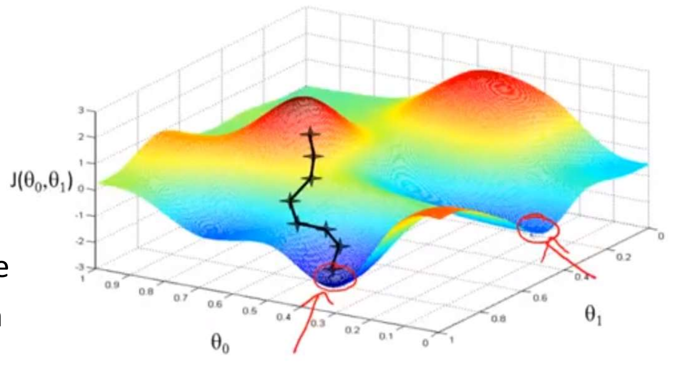

Very small learning rates
May not be able to "cross" towards the good side
May take too long to converge

Too small learning rate will take too long

GD: Advantages and Disadvantages

16

- Advantages
  - Ease of implementation
  - Convergence guarantee
  - Scalability
- Disadvantages
  - Sensitivity to learning rate
  - Sensitivity to initialization
  - Convergence speed



Courtesy: Andrew Ng

Optimizing the Neural Network

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

Solve via: $\min_{\Theta} J(\Theta)$

$J(\Theta)$ is not convex, so GD on a neural net yields a local optimum
- But, tends to work well in practice

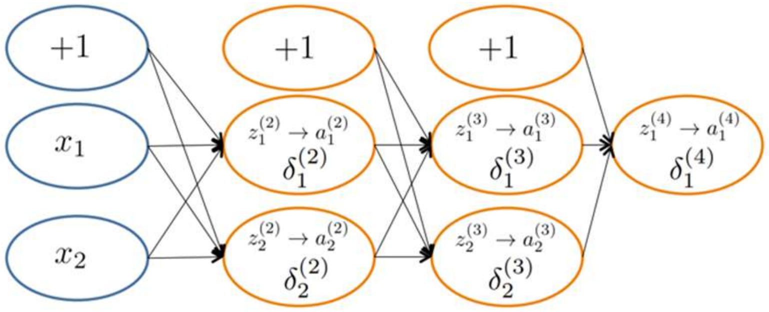
Need code to compute:
- $J(\Theta)$
- $\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)$

Backpropagation

- Similar to the perceptron learning algorithm, we cycle through our examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error

- The trick is to assess the blame for the error and divide it among the contributing weights

Backpropagation Intuition

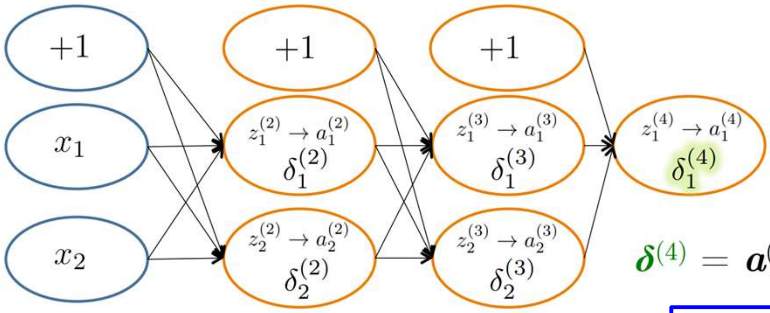

+1
x_1
x_2
+1
$z_1^{(2)} \to a_1^{(2)}$
$\delta_1^{(2)}$
$z_2^{(2)} \to a_2^{(2)}$
$\delta_2^{(2)}$
+1
$z_1^{(3)} \to a_1^{(3)}$
$\delta_1^{(3)}$
$z_2^{(3)} \to a_2^{(3)}$
$\delta_2^{(3)}$
$z_1^{(4)} \to a_1^{(4)}$
$\delta_1^{(4)}$

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

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -\left[y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))\right]$

Backpropagation Intuition


+1
x_1
x_2
+1
$z_1^{(2)} \to a_1^{(2)}$
$\delta_1^{(2)}$
$z_2^{(2)} \to a_2^{(2)}$
$\delta_2^{(2)}$
+1
$z_1^{(3)} \to a_1^{(3)}$
$\delta_1^{(3)}$
$z_2^{(3)} \to a_2^{(3)}$
$\delta_2^{(3)}$
$z_1^{(4)} \to a_1^{(4)}$
$\delta_1^{(4)}$

$$\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$$

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

$$\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{z}} = g'(\boldsymbol{z}) = g(\boldsymbol{z})(1 - g(\boldsymbol{z})) = \boldsymbol{a}(1 - \boldsymbol{a})$$

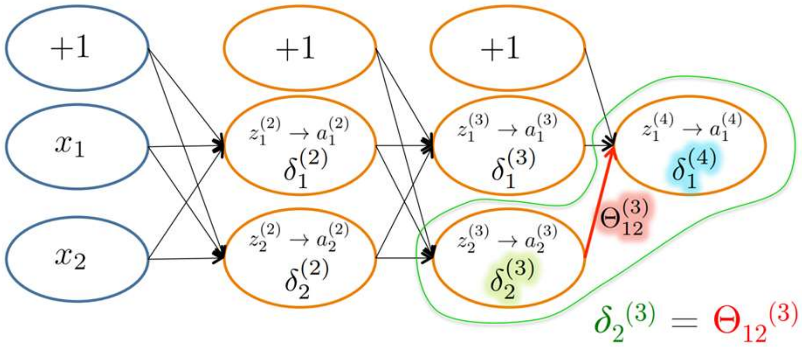
When g is logistic sigmoid

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

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -\left[y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))\right]$

Backpropagation Intuition



$+1$
x_1
x_2
$+1$
$z_1^{(2)} \to a_1^{(2)}$
$\delta_1^{(2)}$
$z_2^{(2)} \to a_2^{(2)}$
$\delta_2^{(2)}$
$+1$
$z_1^{(3)} \to a_1^{(3)}$
$\delta_1^{(3)}$
$z_2^{(3)} \to a_2^{(3)}$
$\delta_2^{(3)}$
$z_1^{(4)} \to a_1^{(4)}$
$\delta_1^{(4)}$
$\Theta_{12}^{(3)}$

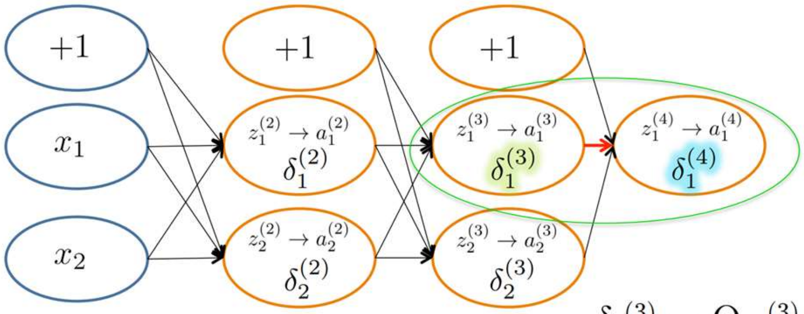
$$\delta_2{}^{(3)} = \Theta_{12}{}^{(3)} \times \delta_1{}^{(4)} \times (a_2^{(3)}(1 - a_2^{(3)}))$$

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

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -\left[y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i)\log(1 - h_\Theta(\mathbf{x}_i))\right]$

Backpropagation Intuition



$+1$
x_1
x_2
$+1$
$z_1^{(2)} \to a_1^{(2)}$
$\delta_1^{(2)}$
$z_2^{(2)} \to a_2^{(2)}$
$\delta_2^{(2)}$
$+1$
$z_1^{(3)} \to a_1^{(3)}$
$\delta_1^{(3)}$
$z_2^{(3)} \to a_2^{(3)}$
$\delta_2^{(3)}$
$z_1^{(4)} \to a_1^{(4)}$
$\delta_1^{(4)}$

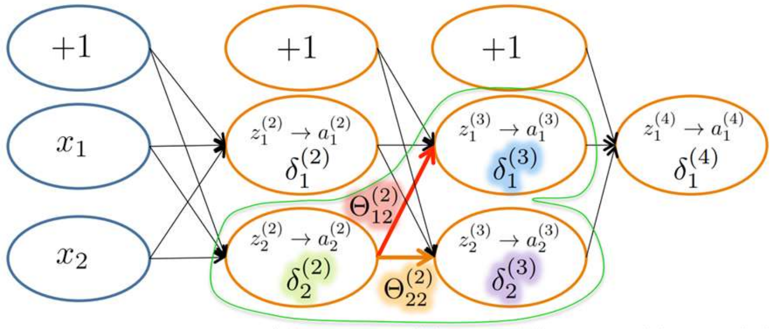
$$\delta_2{}^{(3)} = \Theta_{12}{}^{(3)} \times \delta_1{}^{(4)} \times (a_2^{(3)}(1 - a_2^{(3)}))$$
$$\delta_1{}^{(3)} = \Theta_{11}{}^{(3)} \times \delta_1{}^{(4)} \times (a_1^{(3)}(1 - a_1^{(3)}))$$

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

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -\left[y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i)\log(1 - h_\Theta(\mathbf{x}_i))\right]$

Backpropagation Intuition



$+1$
x_1
x_2
$z_1^{(2)} \to a_1^{(2)}$ $\delta_1^{(2)}$
$z_2^{(2)} \to a_2^{(2)}$ $\delta_2^{(2)}$
$z_1^{(3)} \to a_1^{(3)}$ $\delta_1^{(3)}$
$z_2^{(3)} \to a_2^{(3)}$ $\delta_2^{(3)}$
$z_1^{(4)} \to a_1^{(4)}$ $\delta_1^{(4)}$
$\Theta_{12}^{(2)}$
$\Theta_{22}^{(2)}$

$\delta_2{}^{(2)} = (\Theta_{12}{}^{(2)} \times \delta_1{}^{(3)} + \Theta_{22}{}^{(2)} \times \delta_2{}^{(3)}) \times (a_2^{(2)}(1 - a_2^{(2)}))$

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

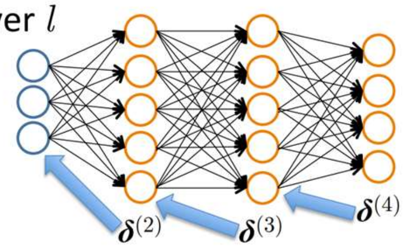
Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = -\left[y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))\right]$

Backpropagation Intuition: Gradient Computation

Let $\delta_j^{(l)} =$ “error” of node j in layer l

(#layers L = 4)



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

Element-wise product .*

Backpropagation

- $\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$
- $\boldsymbol{\delta}^{(3)} = (\Theta^{(3)})' \boldsymbol{\delta}^{(4)} .* g'(\mathbf{z}^{(3)})$
- $\boldsymbol{\delta}^{(2)} = (\Theta^{(2)})' \boldsymbol{\delta}^{(3)} .* g'(\mathbf{z}^{(2)})$
- (No $\boldsymbol{\delta}^{(1)}$)

$g'(\mathbf{z}^{(3)}) = \mathbf{a}^{(3)} .* (1 - \mathbf{a}^{(3)})$

$g'(\mathbf{z}^{(2)}) = \mathbf{a}^{(2)} .* (1 - \mathbf{a}^{(2)})$

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

Training a Neural Network via Gradient Descent with Backprop

```
Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch
    Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$                (Used to accumulate gradient)
    For each training instance $(\mathbf{x}^{(s)}, y^{(s)})$:
        Set $\mathbf{a}^{(1)} = \mathbf{x}^{(s)}$
        Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation
        Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y^{(s)}$
        Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$
        Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
    Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$
    Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$
Until weights converge or max #epochs is reached
```

Backpropagation

Questions?