CS60050
Machine Learning

Neural Networks - Backpropagation

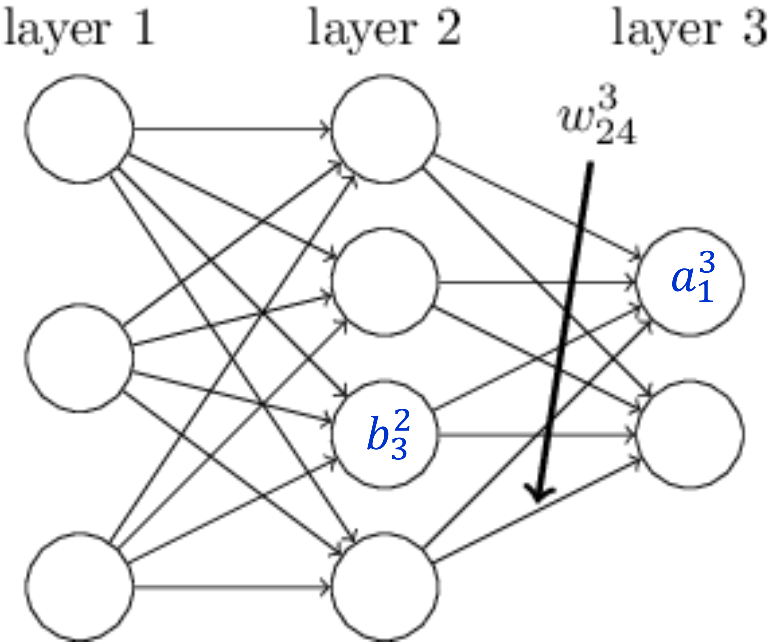
Somak Aditya
Assistant Professor

Sudeshna Sarkar

Department of CSE, IIT Kharagpur

Oct 12, 2023

Notations



w_{jk}^l is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

b_j^l : Bias for j^{th} neuron in l^{th} layer.

a_j^l : Activation of the j^{th} neuron in the l^{th} layer.

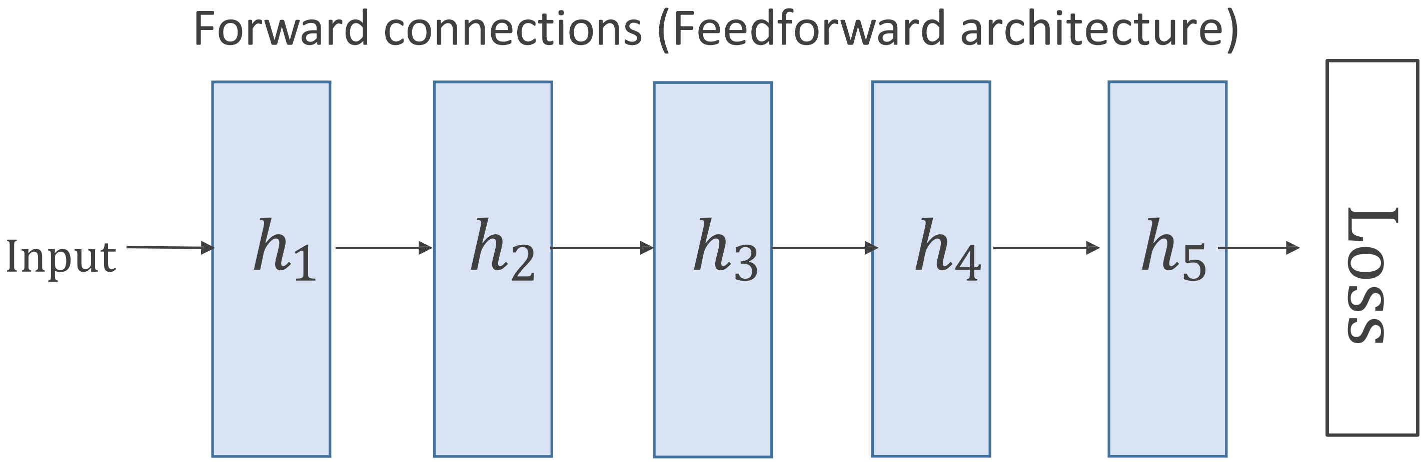
$$a_j^l = g\left(\sum_k w_{jk}^l \, a_k^{l-1} + b_j^l\right)$$

Vectorize

$$a^l = g\left(w^l a^{l-1} + b^l\right)$$
$$z^l = w^l a^{l-1} + b^l$$
$$a^l = g\left(z^l\right)$$

Neural networks in blocks

- We can visualize $a_L = h_L \circ h_{L-1} \circ \cdots \circ h_1(x)$ as a cascade of blocks.



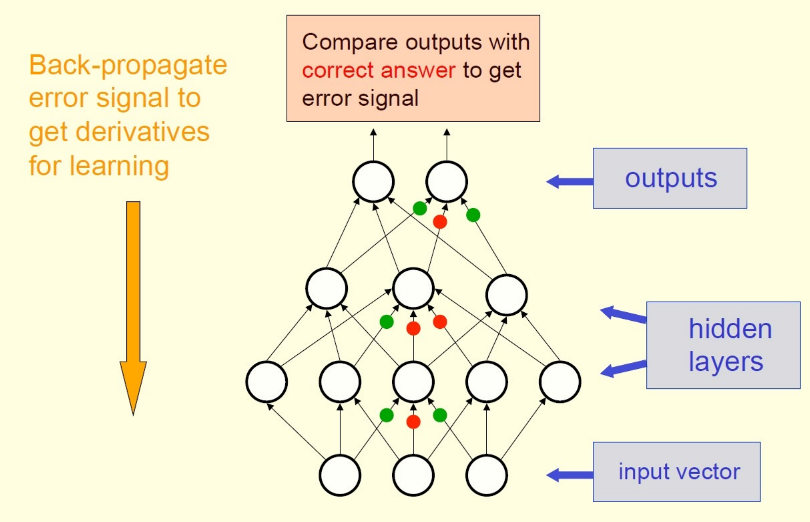
The activation functions must be **1st-order differentiable (almost) everywhere**

Backpropagation

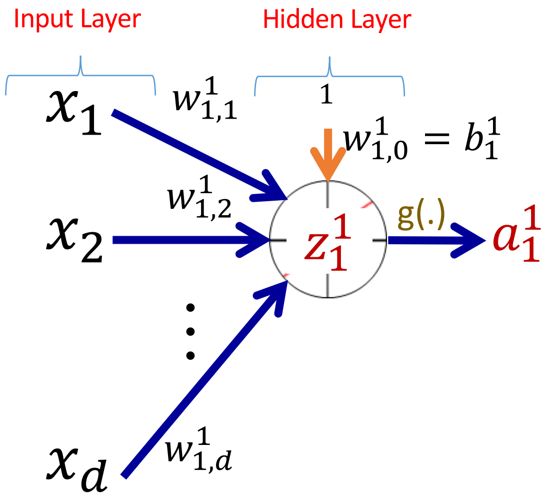
- Feedforward Propagation: Accept input $x^{(i)}$, pass through intermediate stages and obtain output $\hat{y}^{(i)}$
- During Training: Compute scalar cost J(θ)

$$J(\theta) = \sum_i L\left(NN\left(x^{(i)}; \theta\right), y^{(i)}\right)$$

- Backpropagation allows information to flow backwards from cost to compute the gradient



Multilayer Neural Network

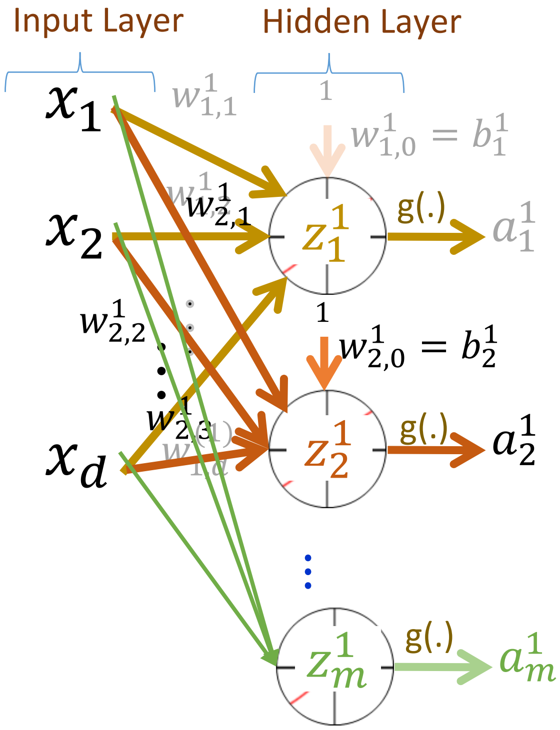


$$z_1^1 = b_1^1 + \sum_{i=1}^{d} w_{1,i}^1\, x_i = [\mathbf{w}_1^1 b_1^1] \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

$$[x_1\ x_2 \ldots x_d]^T$$

$$a_1^1 = g(z_1^1)$$

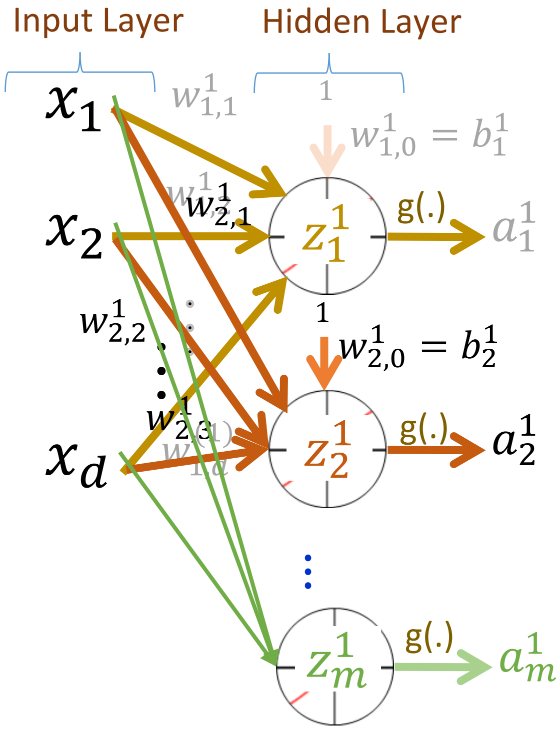
Multilayer Neural Network



$$a^{(0)} = x$$
$$z^{(1)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(0)}$$
$$a^{(1)} = g\left(\boldsymbol{z}^{(1)}\right)$$

W^1 : $m \times n$ matrix
b^1 : $m \times 1$ column vector
X : $d \times 1$ column vector
Z^1 : $m \times 1$ column vector
A^1 : $m \times 1$ column vector

Multilayer Neural Network



$$a^{(0)} = x$$
$$z^{(1)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(0)}$$
$$a^{(1)} = g\left(\boldsymbol{z}^{(1)}\right)$$

$$\begin{bmatrix} a^1_1 \\ a^1_2 \\ \vdots \\ a^1_m \end{bmatrix} = \begin{bmatrix} g(z^1_1) \\ g(z^1_2) \\ \vdots \\ g(z^1_m) \end{bmatrix}$$

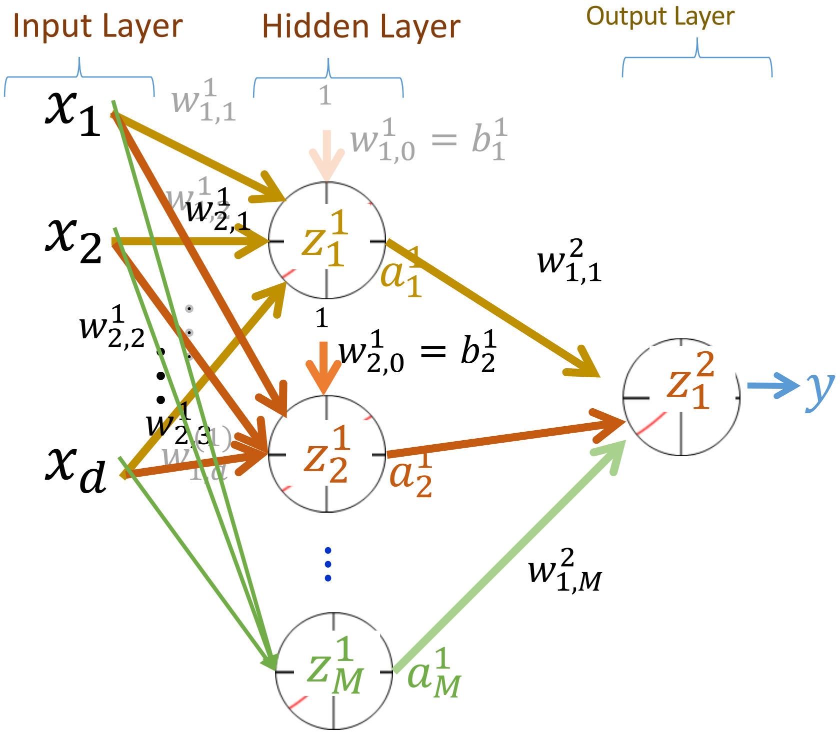
$$\boldsymbol{a}^1 = \boldsymbol{g}\left(\boldsymbol{z}^{(1)}\right)$$

$$\boldsymbol{z}^1 = [\boldsymbol{W}^1 \boldsymbol{b}^1]\begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

$$z^1_1 = [\boldsymbol{w}^1_1 b^1_1]\begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$
$$z^1_2 = [\boldsymbol{w}^1_2 b^1_2]\begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$
$$\vdots$$
$$z^1_M = [\boldsymbol{w}^1_m b^1_2]\begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} z^1_1 \\ z^1_2 \\ \vdots \\ z^1_m \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}^1_1 & b^1_1 \\ \boldsymbol{w}^1_2 & b^1_2 \\ \vdots & \vdots \\ \boldsymbol{w}^1_m & b^1_m \end{bmatrix}\begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

Multilayer Neural Network



Output Layer Pre-activation

$$z_1^{(2)} = \left[\boldsymbol{w}_1^{(2)}\ b_1^{(2)}\right]\begin{bmatrix}\boldsymbol{a}^{(1)} \\ 1\end{bmatrix}$$

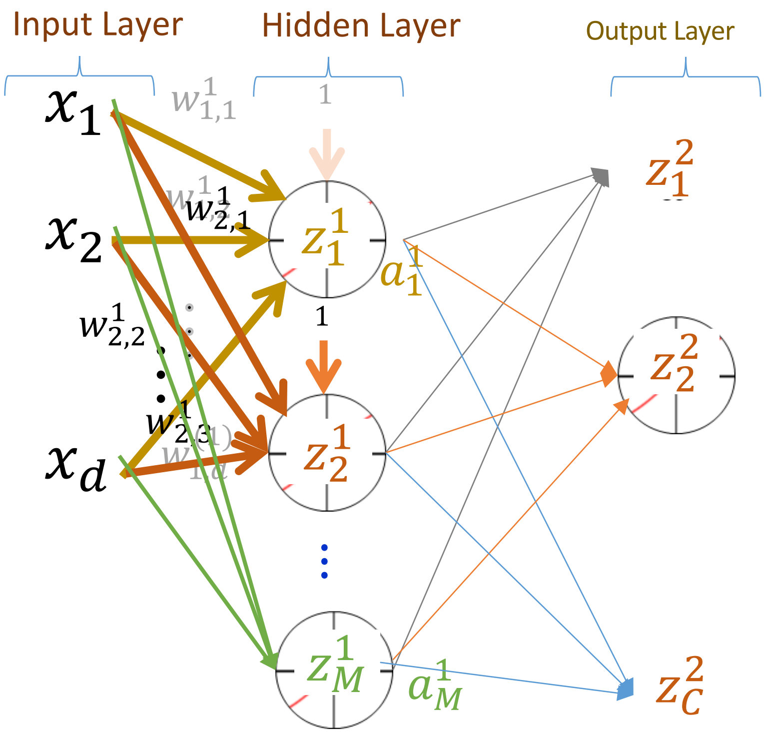
Output Layer Activation

$$y_1 = o\left(\boldsymbol{z}_1^{(2)}\right)$$

output

- Sigmoid for 2-class classification
- Softmax for multi-class classification
- Linear for regression

Multilayer Neural Network

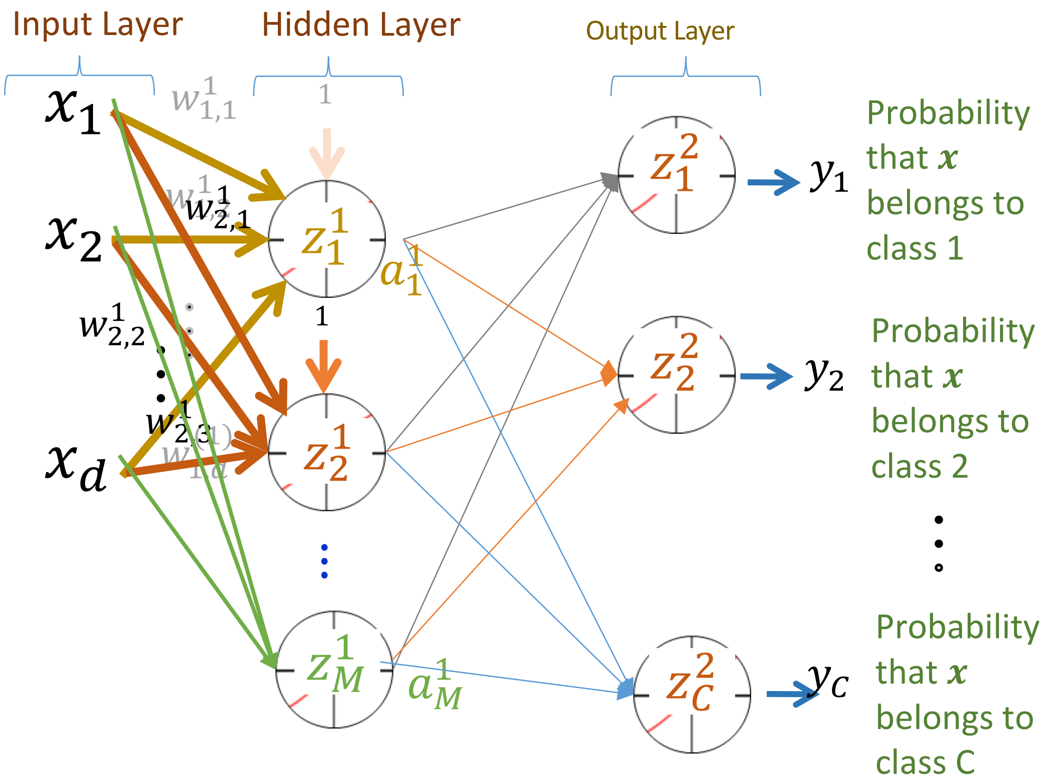


$$\rightarrow y_1 = o_1\left(z_1^{(2)}\right) = \frac{\exp(z_1^{(2)})}{\sum_c \exp(z_c^{(2)})}$$

$$\rightarrow y_2 = o_2\left(z_2^{(2)}\right) = \frac{\exp(z_2^{(2)})}{\sum_c \exp(z_c^{2})}$$

$$\rightarrow y_C = o_C\left(z_C^{(2)}\right) = \frac{\exp(z_C^{(2)})}{\sum_c \exp(z_c^{(2)})}$$

Training a Neural Network – Loss Function



Aim to maximize the probability corresponding to the correct class for any example $\boldsymbol{x}$

$$\max \boldsymbol{y}_c$$
$$\equiv \max \left(log\ \boldsymbol{y}_c\right)$$
$$\equiv \min \left(-log\ \boldsymbol{y}_c\right)$$

Can be equivalently expressed as

$$-\sum_i \prod_{i=c} \log(y_i)$$

known as <u>cross-entropy loss</u>

Forward Pass

$\boldsymbol{\theta}$ is the collection of all learnable parameters i.e., all $\boldsymbol{W}$ and $\boldsymbol{b}$

$$\boldsymbol{x} \xrightarrow{\boldsymbol{W}^1, \boldsymbol{b}^1} \boldsymbol{z}^1 \xrightarrow{g(.)} \boldsymbol{a}^1 \xrightarrow{\boldsymbol{W}^2, \boldsymbol{b}^2} \boldsymbol{z}^2 \xrightarrow{g(.)} \boldsymbol{a}^2 \rightarrow \cdots \xrightarrow{\boldsymbol{W}^L, \boldsymbol{b}^L} \boldsymbol{z}^L \xrightarrow{o(.)} \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$$

Hidden layer pre-activation:

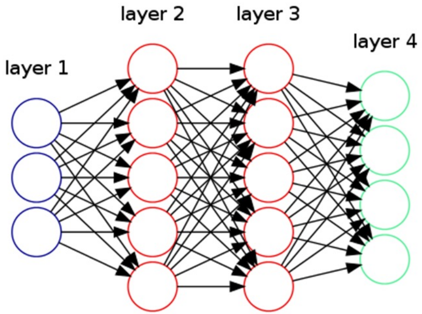
For $l = 1, \dots, L$; $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$

Hidden layer activation:

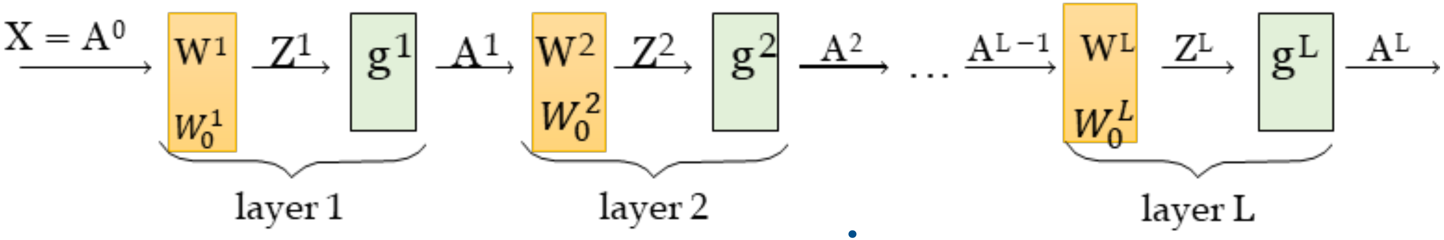
For $l = 1, \dots, L-1$; $\boldsymbol{a}^{(l)} = g(\boldsymbol{z}^{(l)})$

Output layer activation:

For $l = L$; $\boldsymbol{y} = \boldsymbol{a}^{(L)} = o(\boldsymbol{z}^{(L)}) = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$



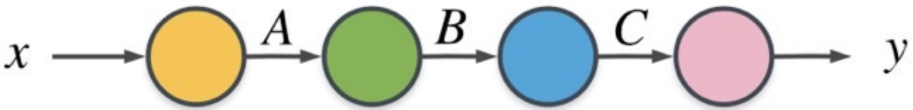
Error back-propagation



1. Compute Loss
2. Compute the derivative of the L w.r.t. the final output of the network A^L
3. Compute the derivative of L w.r.t. the pre-activation Z^L
4. Compute the derivative of L wrt W^L
5. Then compute the derivative of the L w.r.t. the final output of the network A^{L-1}
6. Then compute the derivative of L w.r.t. the pre-activation Z^{L-1}
7. Compute the derivative of L wrt W^{L-1}

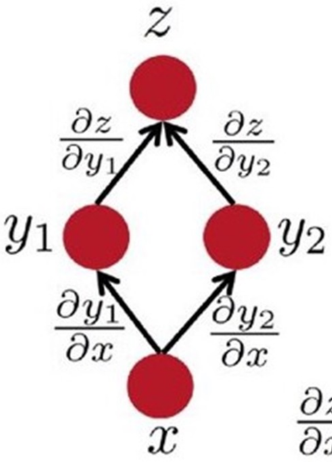
…. ….

Chain Rule



$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial C} \times \frac{\partial C}{\partial B} \times \frac{\partial B}{\partial A} \times \frac{\partial A}{\partial x}$$

Multiple Path

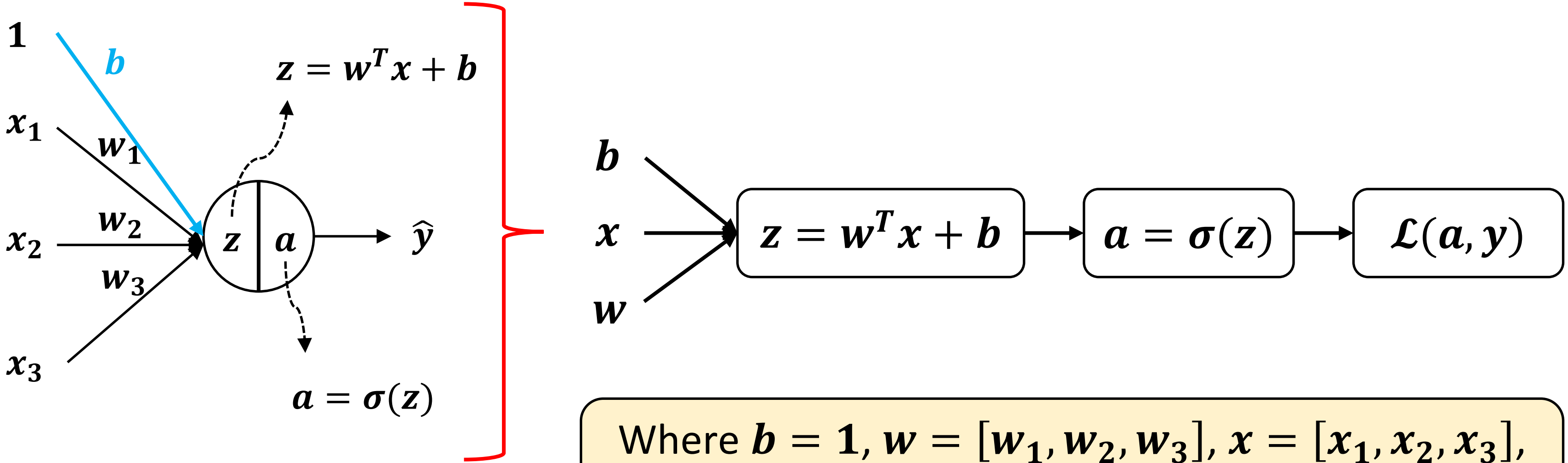


$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

Backpropagation is just repeated application of the chain rule.

The Computation Graph of Logistic Regression

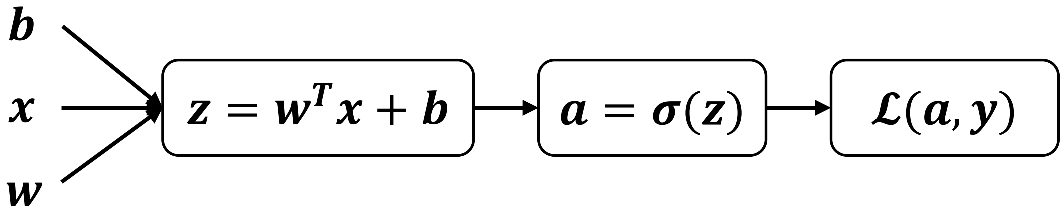
Let us translate logistic regression into a computation graph



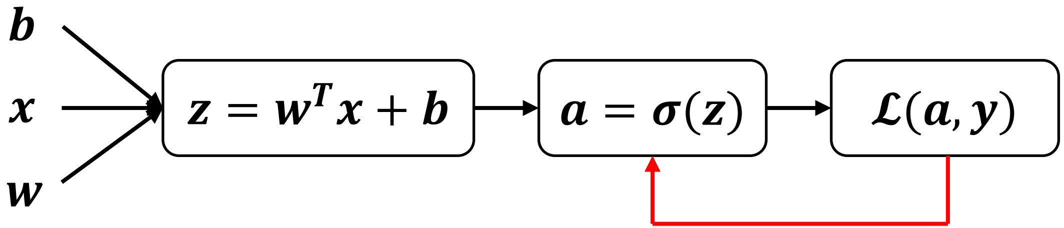
Where $b = 1$, $w = [w_1, w_2, w_3]$, $x = [x_1, x_2, x_3]$,
and $\mathcal{L}(a, y)$ is the cost (or ***loss***) function

Forward Propagation

- The loss function can be computed by moving from left to right



Backward Propagation

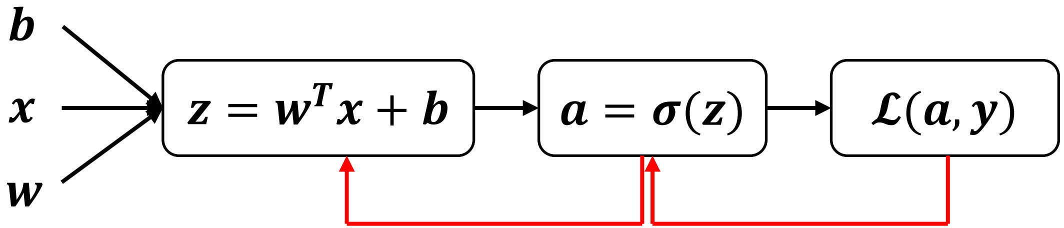


Partial derivative of $\mathcal{L}$ with respect to a

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial}{\partial a}\big(-y log(a) - (1 - y) log(1 - a)\big)$$

$$= \frac{-y}{a} + \frac{(1 - y)}{(1 - a)}$$

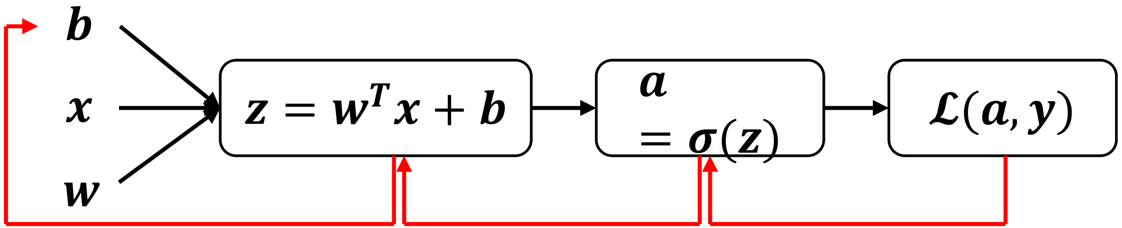
Backward Propagation



$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} = \left(\frac{-y}{a} + \frac{(1-y)}{(1-a)}\right) \times \frac{\partial a}{\partial z} = \left(\frac{-y}{a} + \frac{(1-y)}{(1-a)}\right) \times a(1-a)$$

$$= a - y$$

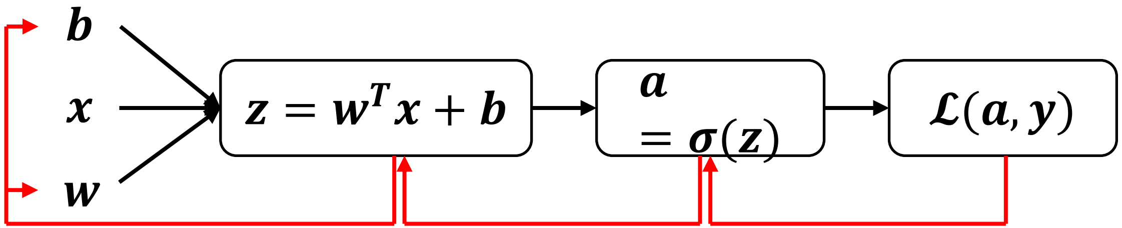
Backward Propagation



$$\frac{\partial \mathcal{L}}{\partial b} = \text{Partial derivative of } \mathcal{L} \text{ with respect to } b$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial b} = (a - y) \times \frac{\partial z}{\partial b} = (a - y) \times 1 = (a - y)$$

Backward Propagation



$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w} = (a - y) \times \frac{\partial z}{\partial w} = (a - y)x$$

Backward Propagation: Summary

- Here is the summary of the gradients in logistic regression:

$$dz = \frac{\partial \mathcal{L}}{\partial z} = a - y$$

$$db = \frac{\partial \mathcal{L}}{\partial b} = a - y$$

$$dw = \frac{\partial \mathcal{L}}{\partial w} = (a - y)x$$

Backward Pass MLP

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

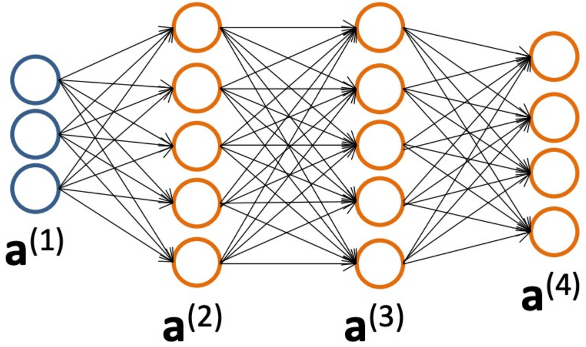
> $J(\Theta)$ is not convex, so GD on a neural net yields a local optimum
> - But, tends to work well in practice

Need code to compute:

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

Forward Propagation

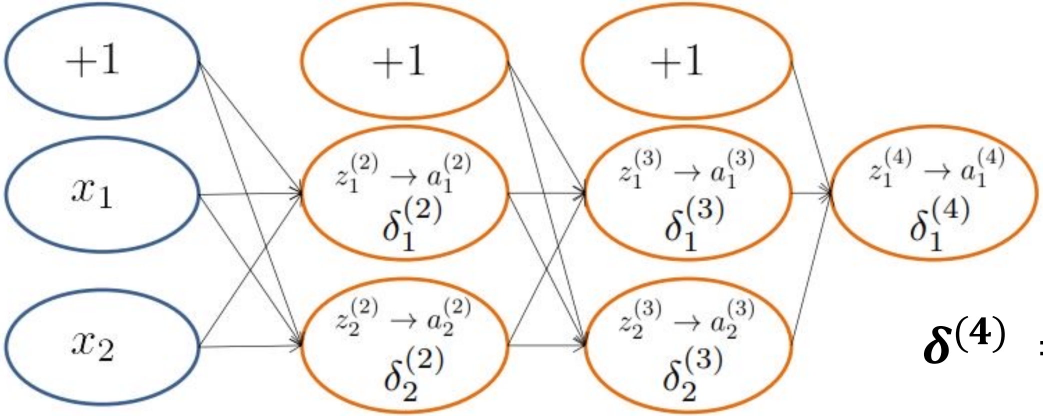
- Given one labeled training instance ($\mathbf{x}$, y):



<u>Forward Propagation</u>

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

Backpropagation



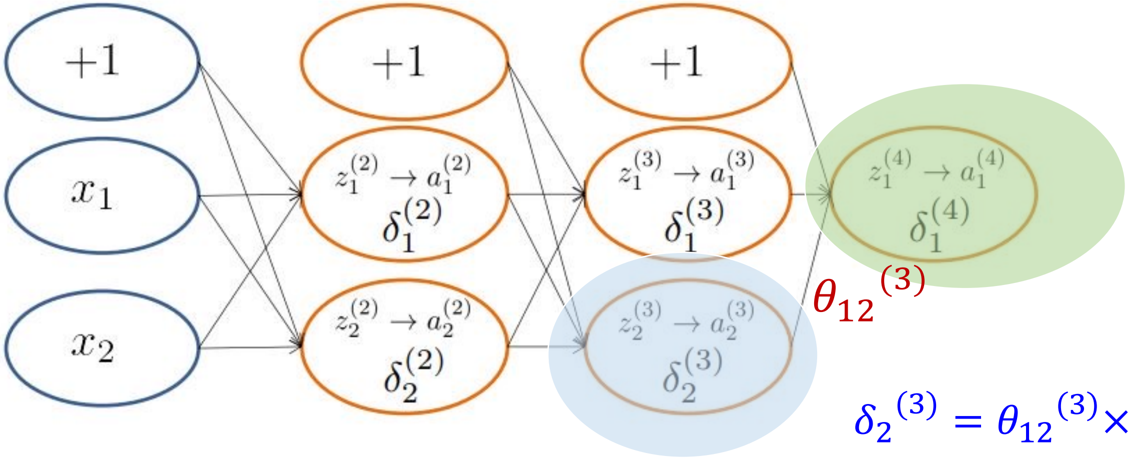
$$\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \boldsymbol{y}$$

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

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

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

Backpropagation



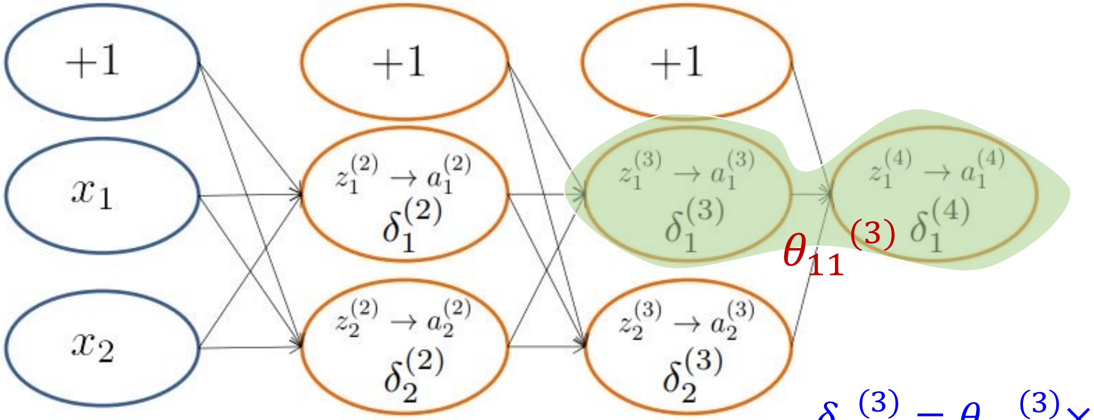
$$\delta_2{}^{(3)} = \theta_{12}{}^{(3)} \times \delta_1{}^{(4)} \times a'^{(3)}_2$$

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

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

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

Backpropagation



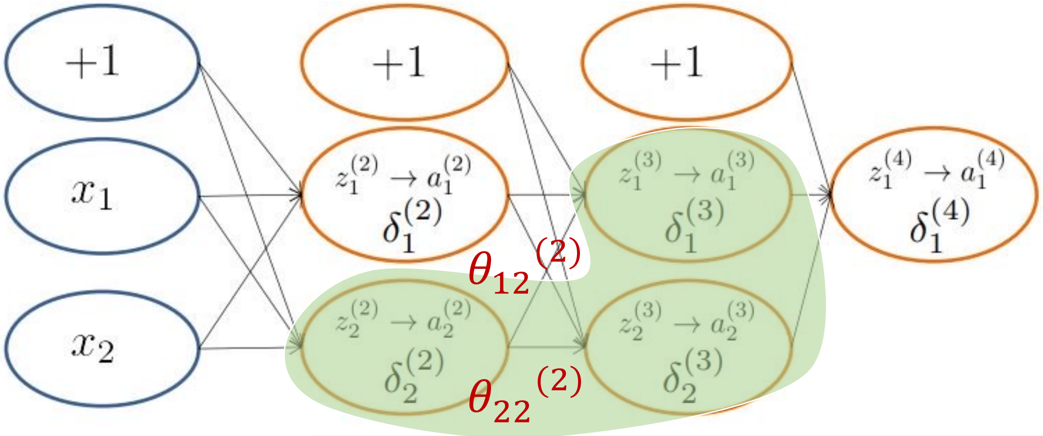
$$\delta_2{}^{(3)} = \theta_{12}{}^{(3)} \times \delta_1{}^{(4)} \times a'^{(3)}_2$$
$$\delta_1{}^{(3)} = \theta_{11}^{(3)} \times \delta_1{}^{(4)} \times a'^{(3)}_1$$

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

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

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

Backpropagation



$$\boxed{\delta_2{}^{(2)} = \theta_{12}{}^{(2)} \times \delta_1{}^{(3)} \times a'^{(2)}_2 + \theta_{22}{}^{(2)} \times \delta_2{}^{(3)} \times a'^{(2)}_2}$$

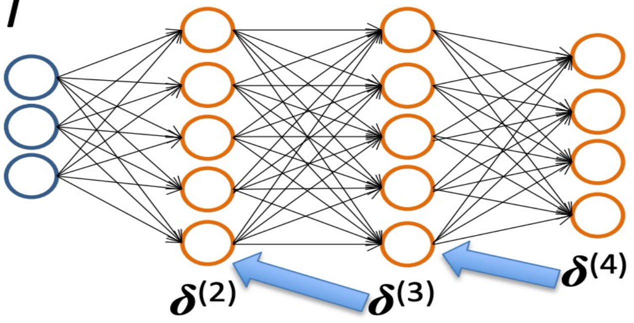
$\delta_j^{(l)}$ = "error" of node *j* in layer *l*

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

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

Gradient Computation

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



Element-wise product .*

<u>Backpropagation</u>

- $\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$
- $\boldsymbol{\delta}^{(3)} = (\Theta^{(3)})^T \boldsymbol{\delta}^{(4)}$.* $g'(\mathbf{z}^{(3)})$
- $\boldsymbol{\delta}^{(2)} = (\Theta^{(2)})^T \boldsymbol{\delta}^{(3)}$.* $g'(\mathbf{z}^{(2)})$
- (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 λ = 0)

Backpropagation

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance $(\mathbf{x}_i, y_i)$:

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

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 $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

$\boldsymbol{D}^{(l)}$ is the matrix of partial derivatives of $J(\Theta)$

Note: Can vectorize $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ as $\boldsymbol{\Delta}^{(l)} = \boldsymbol{\Delta}^{(l)} + \boldsymbol{\delta}^{(l+1)} \mathbf{a}^{(l)\mathsf{T}}$

Training a NN 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 $(\mathbf{x}_i, y_i)$:
Set $\mathbf{a}^{(1)} = \mathbf{x}_i$
Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation
Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$
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 $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Backpropagation

Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$
Until weights converge or max #epochs is reached

Training a Neural Network

- Pick a network architecture (connectivity pattern between nodes)
- # input units = # of features in dataset
- # output units = # classes
- Choose number of hidden layers and number of units in each layer.

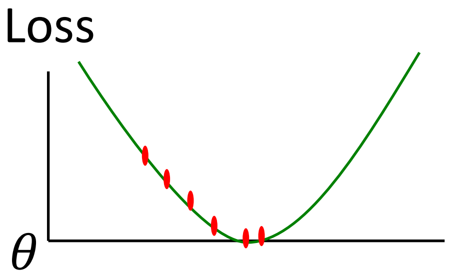
Gradient descent

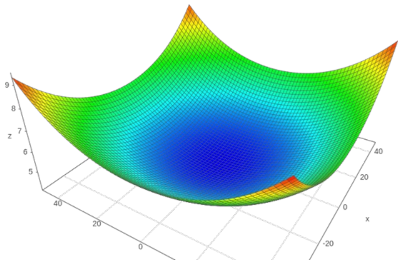
Partial derivatives give us the slope (i.e. direction to move) in that dimension

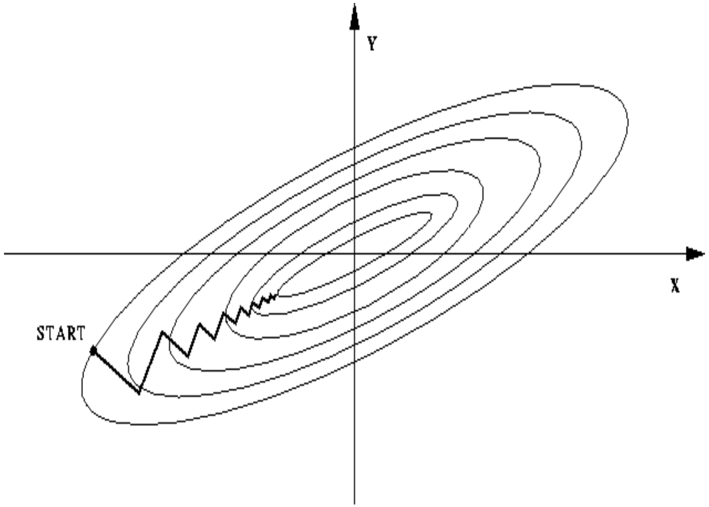
Approach:

- pick a starting point (θ)
- repeat:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

$$\theta_j = \theta_j - \eta \frac{d}{d\theta_j} \mathrm{Loss}(\theta)$$







Batch, Stochastic and Minibatch

- Optimization algorithms that use the entire training set to compute the gradient are called batch or deterministic gradient methods. Ones that use a single training example for that task are called stochastic or online gradient methods
- Most of the algorithms we use for deep learning fall somewhere in between!
- These are called minibatch or minibatch stochastic methods

Batch, Stochastic and Mini-batch Stochastic Gradient Descent

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ

while stopping criteria not met **do**
  Compute gradient estimate over $\boxed{N \text{ examples:}}$
  $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
  Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
end while

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ

while stopping criteria not met **do**
  $\boxed{\text{Sample example } (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \text{ from training set}}$
  Compute gradient estimate:
  $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
  Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
end while

Mini-batch

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$

while stopping criterion not met **do**
  Sample $\boxed{\text{a minibatch of } m \text{ examples}}$ from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
end while

Images courtesy: Shubhendu Trivedi et. Al, Goodfellow et. al..

Batch and Stochastic Gradient Descent