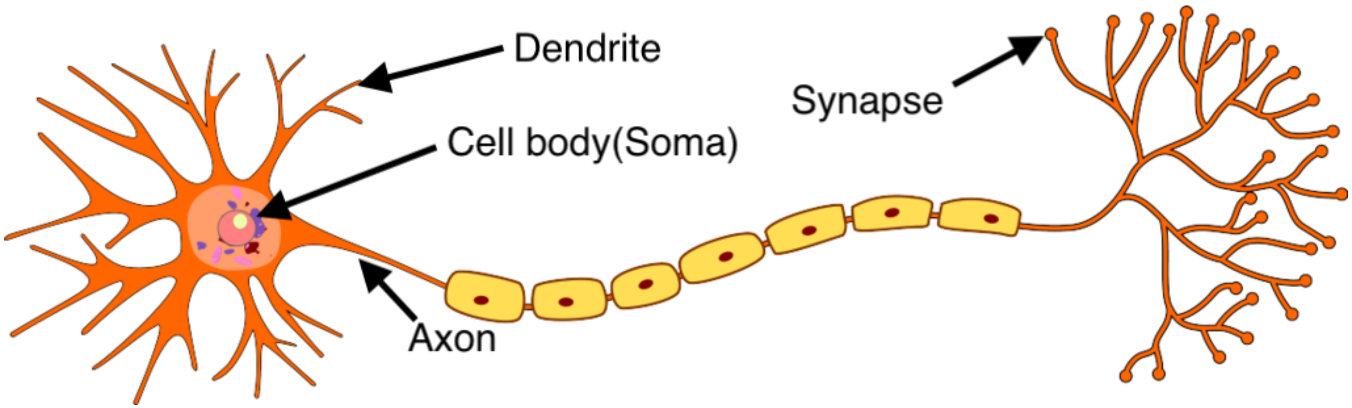
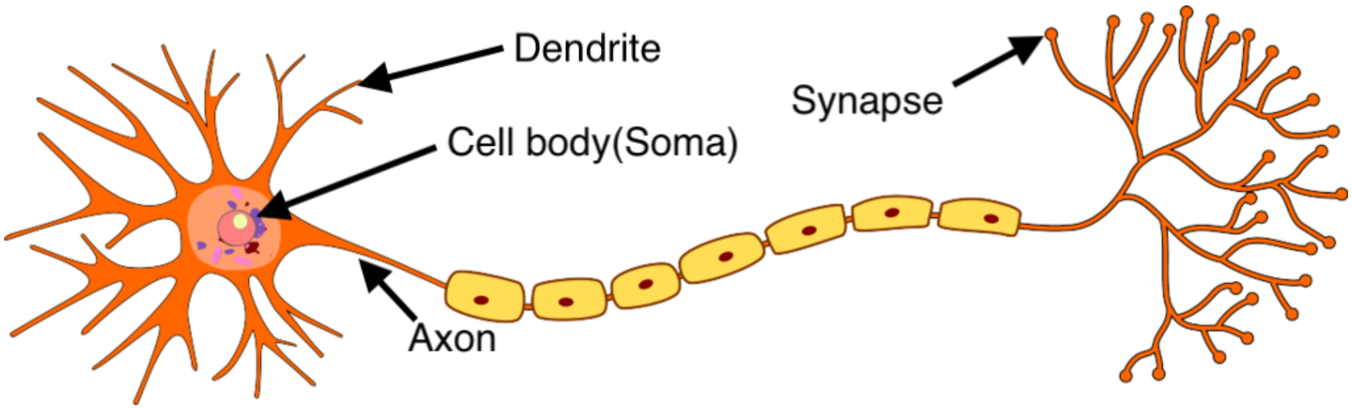
Deep Learning

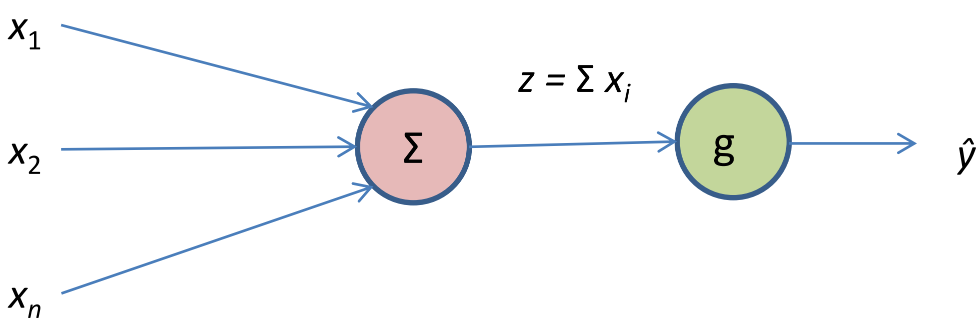
Biological Neuron



- Massively parallel interconnected network of neurons
- Sense organs relay information to lowest layer of neurons
  - Fired neurons relay information to other connected neurons
  - Division of work – respond to certain stimulus
- Average human brain has around 10^{11} neurons

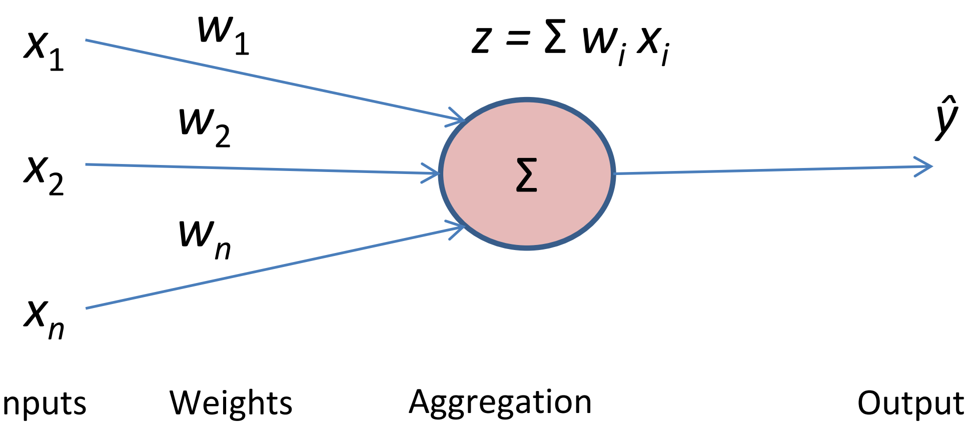
Basic Building Block





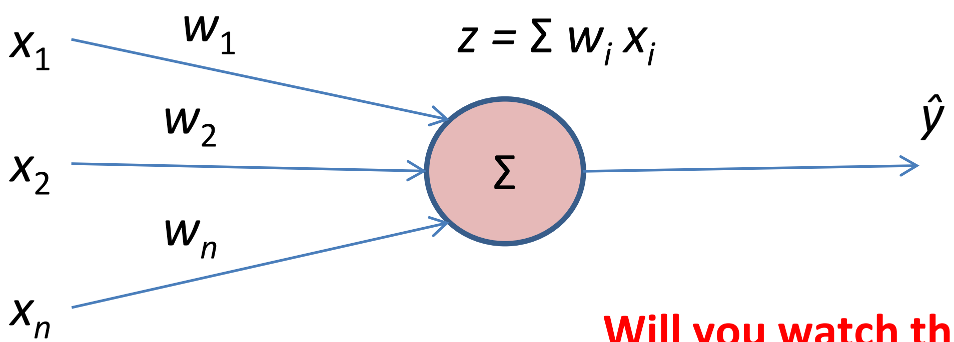
Classical Perceptron

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{pmatrix} \qquad \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \dots \\ w_n \end{pmatrix}$$



$$\hat{y} = (\Sigma\, w_i x_i) \qquad\qquad \forall\, i = 1 \dots n$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

Perceptron – Basic Building Block



Will you watch this movie???

$\hat{y} = (\Sigma\, w_i x_i)$ $\quad\forall\, i = 1\ldots n$

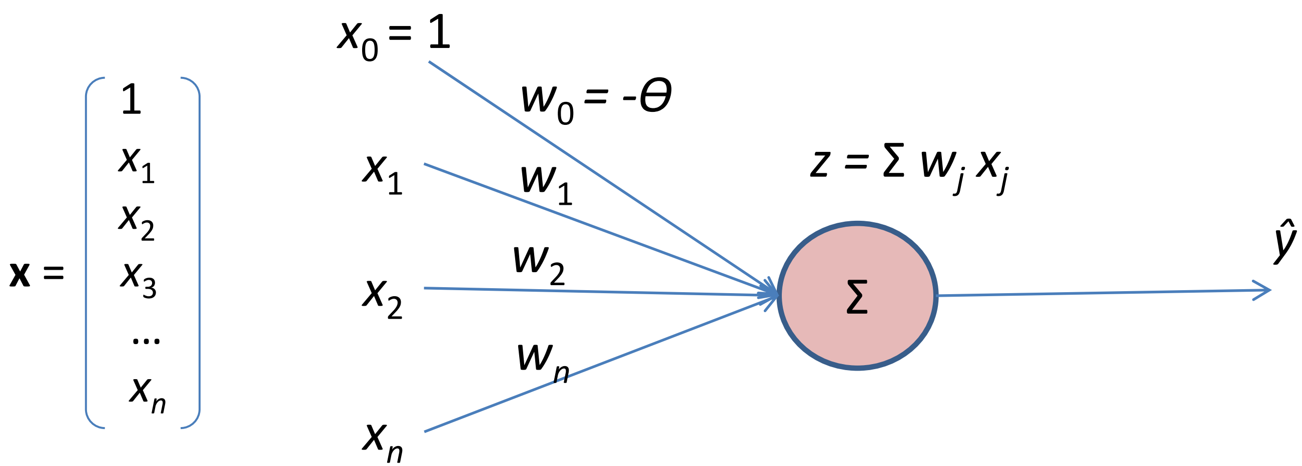
$\hat{y} = 1$ $\quad$ *if* $\Sigma\, w_i x_i >= \Theta$

$\quad = 0$ $\quad$ *if* $\Sigma\, w_i x_i < \Theta$

$\hat{y} = 1$ $\quad$ *if* $\Sigma\, w_i x_i - \Theta \;>= 0$

$\quad = 0$ $\quad$ *if* $\Sigma\, w_i x_i - \Theta \;< 0$

Perceptron – Basic Building Block



$\hat{y} = 1 \qquad \text{if } \Sigma\, w_i x_i - \Theta \;>= 0 \qquad \forall\, i = 1 \ldots n$

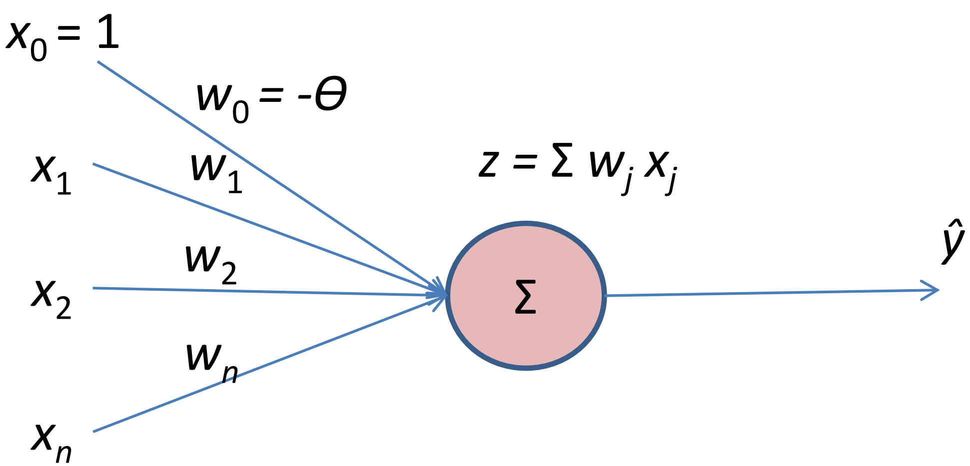
$\quad = 0 \qquad \text{if } \Sigma\, w_i x_i - \Theta \;< 0$

$\hat{y} = 1 \qquad \text{if } \Sigma\, w_i x_i >= 0 \qquad \forall\, i = 0 \ldots n,\ w_0 = -\,\Theta,\ x_0 = 1$

$\quad = 0 \qquad \text{if } \Sigma\, w_i x_i \;< 0$

w_0 / *b:* bias, represents the prior or prejudice

Perceptron – Basic Building Block



$\hat{y} = 1 \quad$ if $\Sigma\, w_i x_i >= 0$

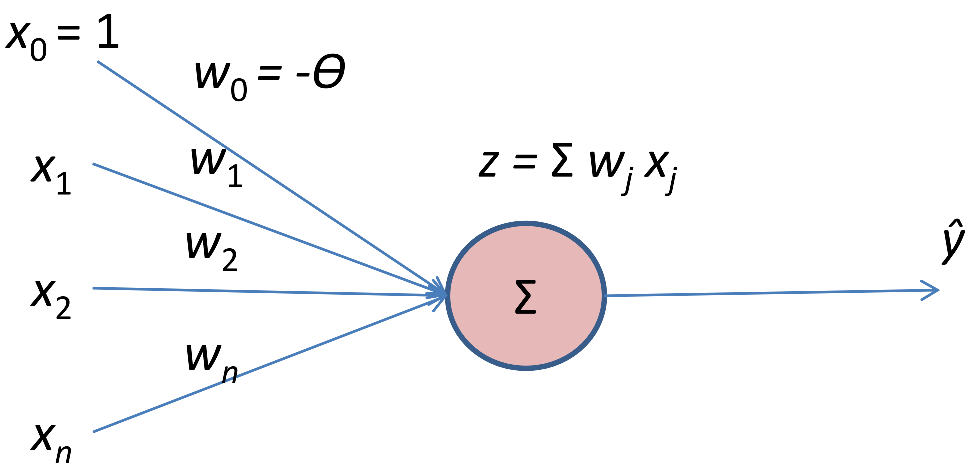
$\quad = 0 \quad$ if $\Sigma\, w_i x_i < 0$

$\forall\, i = 0 \ldots n,\ w_0 = -\,\Theta,\ x_0 = 1$

w_0 : bias

- A movie buff may have a very low threshold and may watch any movie [Θ = 0]
- A selective movie watcher may watch only a thriller, starring Matt Damon and directed by Nolan [Θ = 3]

Perceptron – Basic Building Block



$\hat{y} = 1 \quad \text{if } \Sigma\, w_i x_i >= 0 \qquad \forall\, i = 0 \ldots n,\ w_0 = -\Theta,\ x_0 = 1$

$\quad = 0 \quad \text{if } \Sigma\, w_i x_i < 0$

Linearly separable decision boundary: $\mathbf{w}^T\mathbf{x} = 0$

Perceptron separates input space into two, but:

- Weights (and threshold) can be learned
- Inputs are real-valued

Perceptron Learning Algorithm

$P \longleftarrow$ *inputs with label* 1;

$N \longleftarrow$ *inputs with label* 0;

Initialize $\mathbf{w} = [w_0, w_1, w_2, \ldots w_n]^T$ randomly;

while *!convergence* **do**

Pick random $\mathbf{x} \in P \cup N$;

if $\mathbf{x} \in P$ *and* $\sum w_i * x_i < 0$ **then**

$\mathbf{w} = \mathbf{w} + \mathbf{x}$;

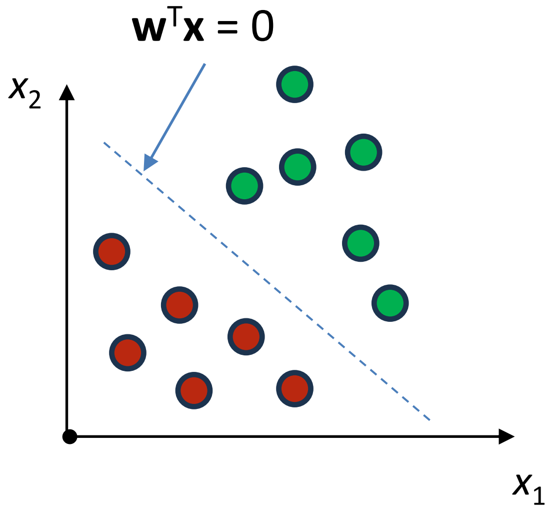
end

if $\mathbf{x} \in N$ *and* $\sum w_i * x_i \geq 0$ **then**

$\mathbf{w} = \mathbf{w} - \mathbf{x}$;

end

end



Convergence- not making any more errors on training data or predictions are not changing

Perceptron Learning Algorithm

- Consider vectors **w** and **x**

$$\mathbf{w} = [w_0, w_1, w_2, \ldots w_n]^T$$

$$\mathbf{x} = [1, x_1, x_2, \ldots x_n]^T$$

Perceptron rule:

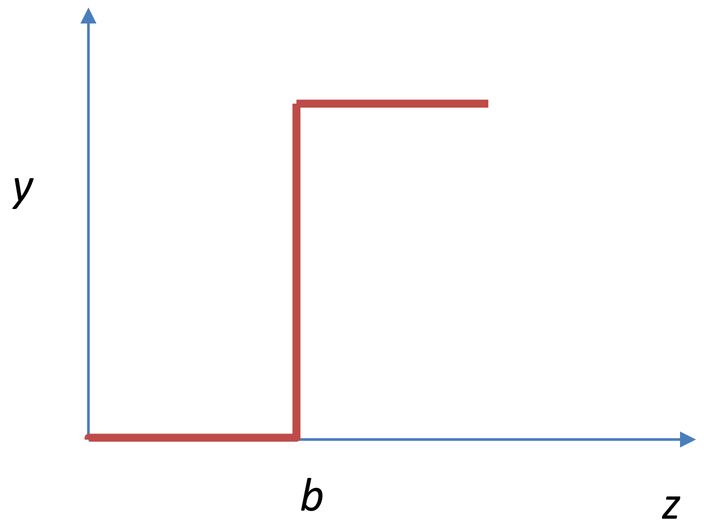
$$\hat{y} = 1 \qquad \text{if } \mathbf{w}^T\mathbf{x} \geq 0$$

$$= 0 \qquad \text{if } \mathbf{w}^T\mathbf{x} < 0$$

- Find line $\mathbf{w}^T\mathbf{x} = 0$ which divides input space into two
- Every point **x** on this line satisfies $\mathbf{w}^T\mathbf{x} = 0$

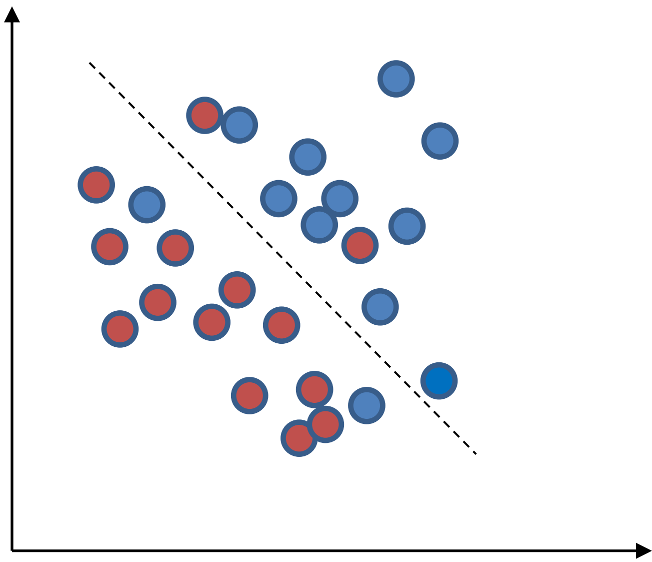
Perceptron Logic

- A perceptron will fire if weighted sum of inputs is greater than threshold (b= $-w_0$)



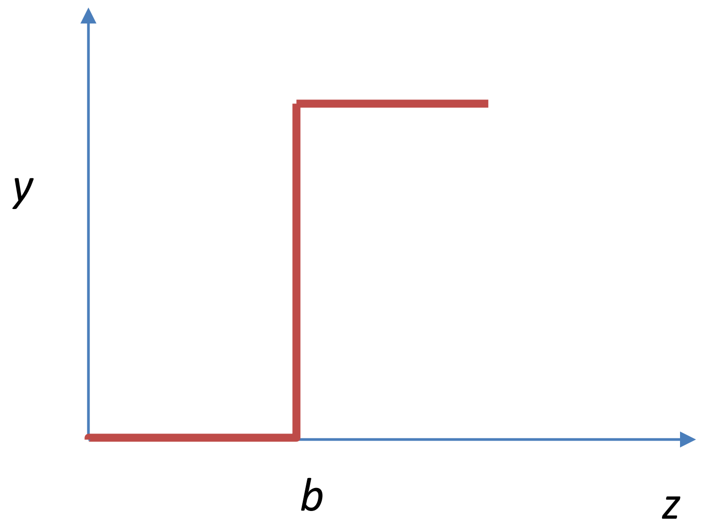
Single perceptron

- Single perceptron cannot deal with data which is not linearly separable
  - Have to be flexible with convergence statement while determining weights…
  - Till almost (say 90%) points are satisfying condition
  - Leads to few errors
  - May not be acceptable in critical real-world applications



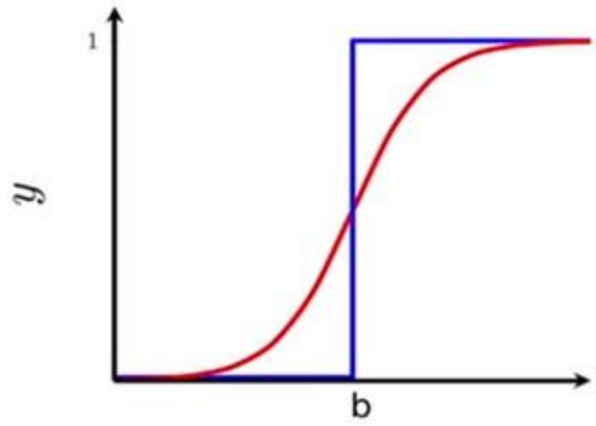
Perceptron Logic

- A perceptron will fire if weighted sum of inputs is greater than threshold (b= -w_0)
  - Thresholding logic is harsh



If x=0.51, watch movie, if x=0.49, do not watch??
For real world problems, we need a smoother decision function

Sigmoid neuron



$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

$z = \sum w_i \, x_i$

If $z = (w^{T}x + b) \longrightarrow \infty$, then $y = 1$

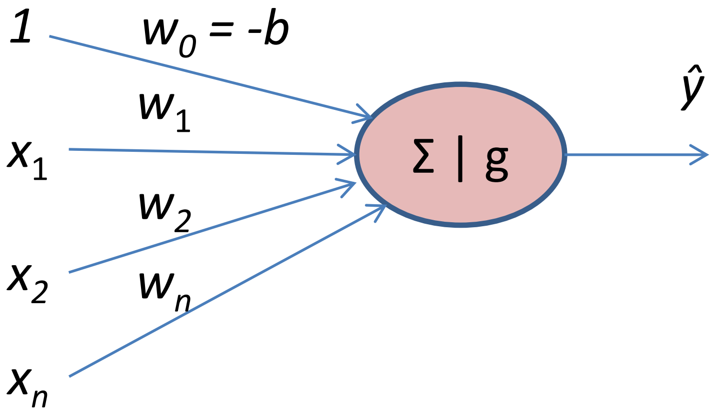
If $z = (w^{T}x + b) \longrightarrow -\infty$, then $y = 0$

If $z = (w^{T}x + b) = 0$, then $y = 0.5$

Range of sigmoid function: 0 to 1

Can be interpreted as probability

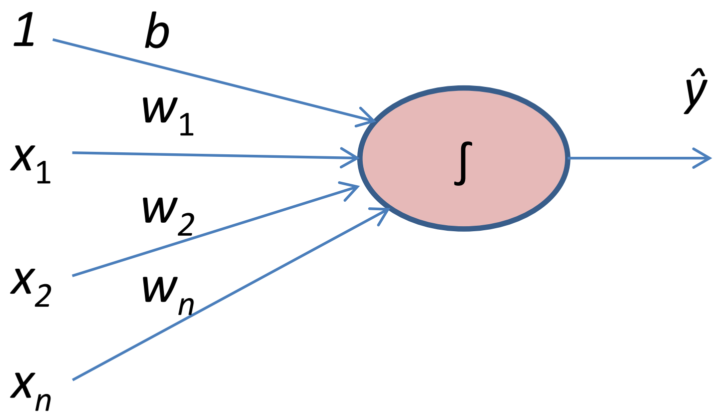
Sigmoid Neuron

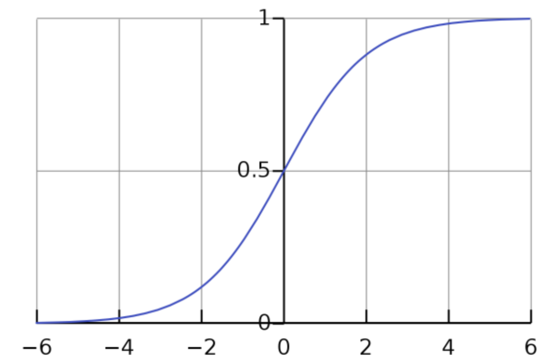


$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

- Smooth
- Continuous
- Differentiable

Perceptron – Forward propagation



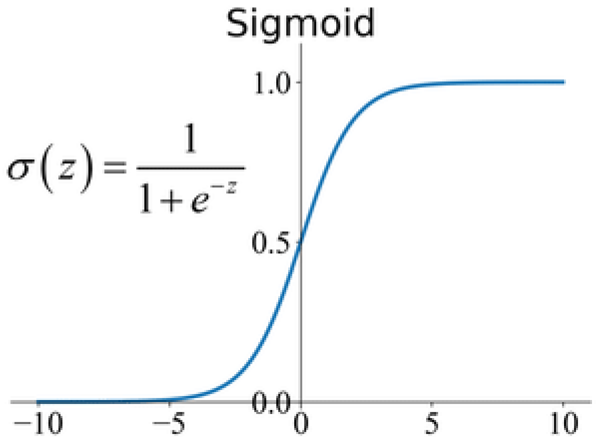


$$\hat{y} = g(z) = g(\Sigma\, w_i x_i) \qquad \forall\, i = 0 \ldots n$$
$$= g(b + \Sigma\, w_i x_i) \qquad \forall\, i = 1 \ldots n$$
$$= g(b + \boldsymbol{W}^T \boldsymbol{X})$$

where, $\mathbf{X} = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}$ $\quad \mathbf{W} = \begin{pmatrix} w_1 \\ w_2 \\ \ldots \\ w_n \end{pmatrix}$ $\quad$ b: bias

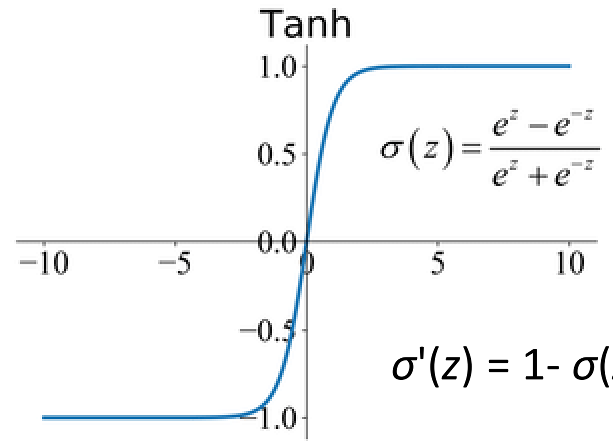
https://en.wikipedia.org/wiki/Sigmoid_function

Activation functions



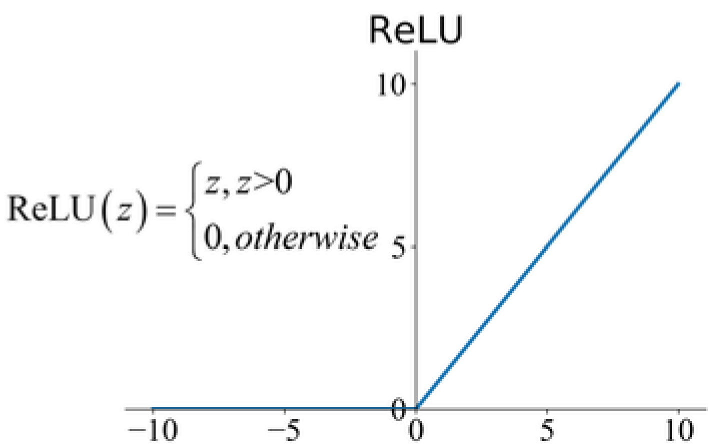
(a)

$\sigma'(z) = \sigma(z)(1- \sigma(z))$



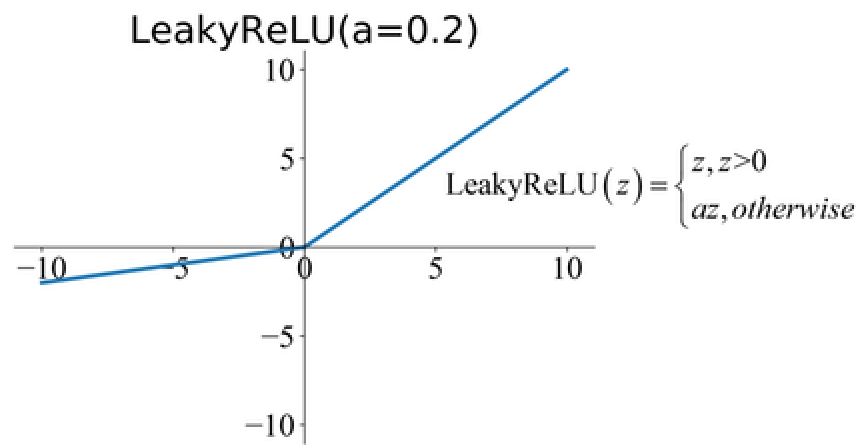
$\sigma'(z) = 1- \sigma(z)^2$

(b)



(c)

$\sigma'(z) = 1 \quad z>=0$
$\quad\quad = 0 \quad z<0$

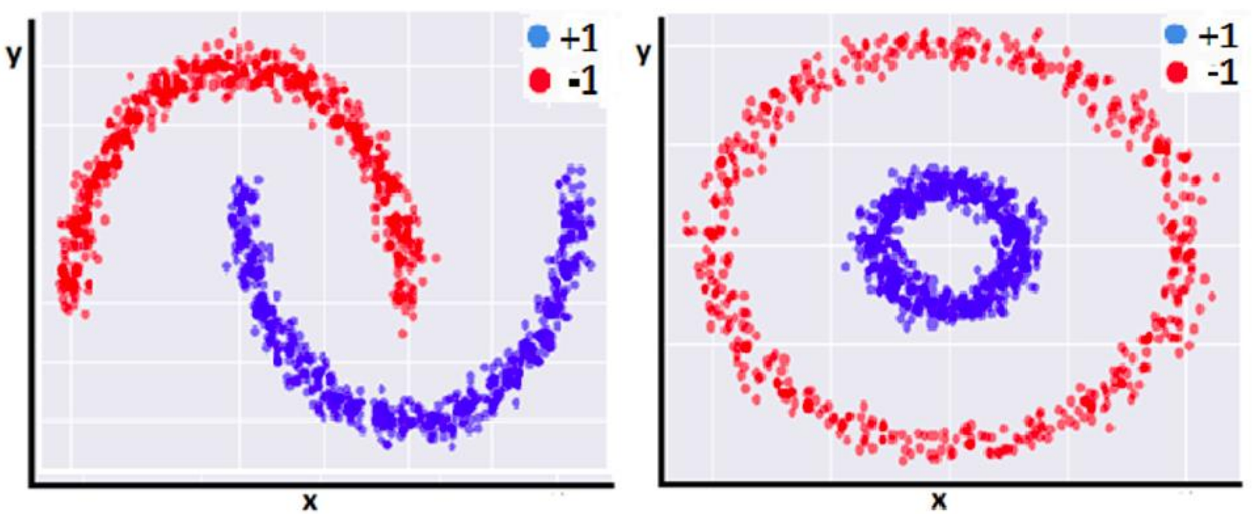


(d)

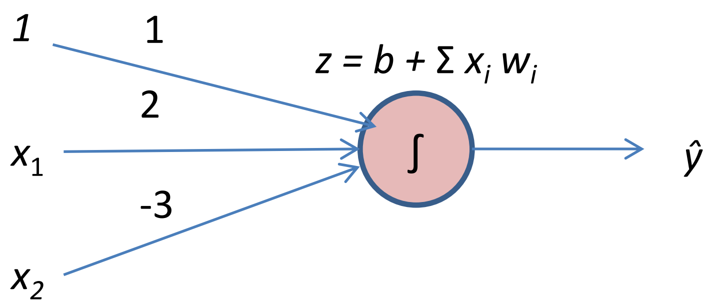
$\sigma'(z) = 1 \quad z>=0$
$\quad\quad = a \quad z<0$

Importance - Activation functions

- Introduce non-linearity in the network
  - Allows to deal with non-linear data
  - Allows to approximate complex functions



Perceptron – Example

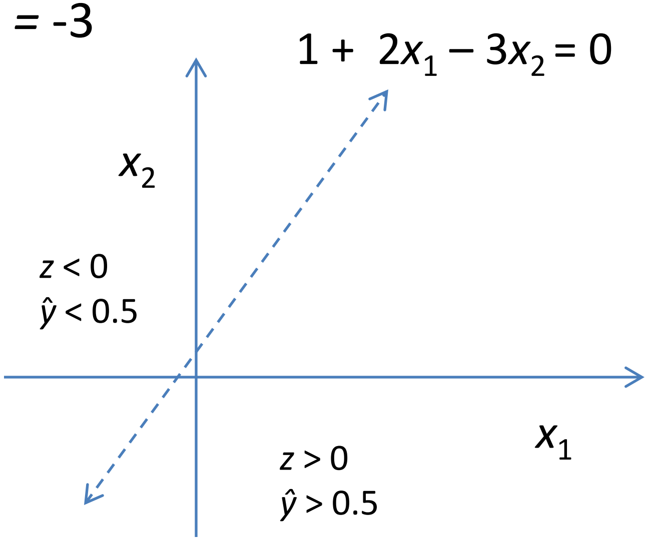


$$\boldsymbol{X} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad \boldsymbol{W} = \begin{pmatrix} 2 \\ -3 \end{pmatrix} \qquad b = 1,\ w_1 = 2,\ w_2 = -3$$

$$\hat{y} = g(b + \boldsymbol{X}^T\boldsymbol{W})$$

$$\hat{y} = g\left(1 + \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} 2 \\ -3 \end{pmatrix}\right)$$

$$\hat{y} = g(1 + 2x_1 - 3x_2)$$



Example

- Ex. data points: (0.5,0.2), (2.5, 0.9)
- At end of training, expect to find *w**, *b** such that

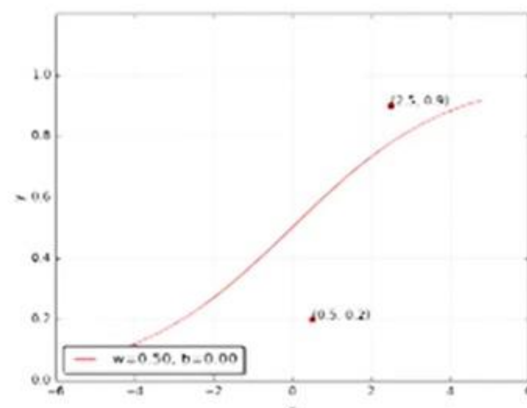
  f(0.5) -> 0.2,   f(2.5) -> 0.9

- Loss = $(1/2)*\sum(y - \hat{y})^2$ given: $w = 0.5$, $b = 0$

  $= (1/2)*[(0.2 - f(0.5))^2 + (0.9 - f(2.5)^2]$

  = 0.073

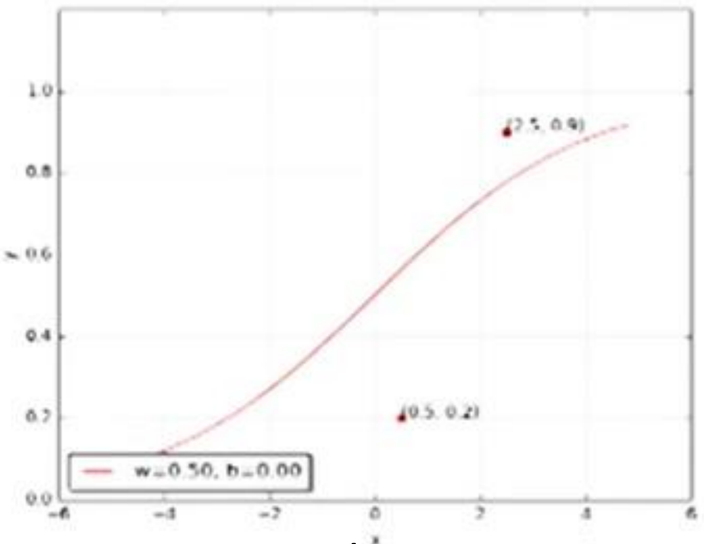
$$y = \frac{1}{1+e^{-(w^T x+b)}}$$



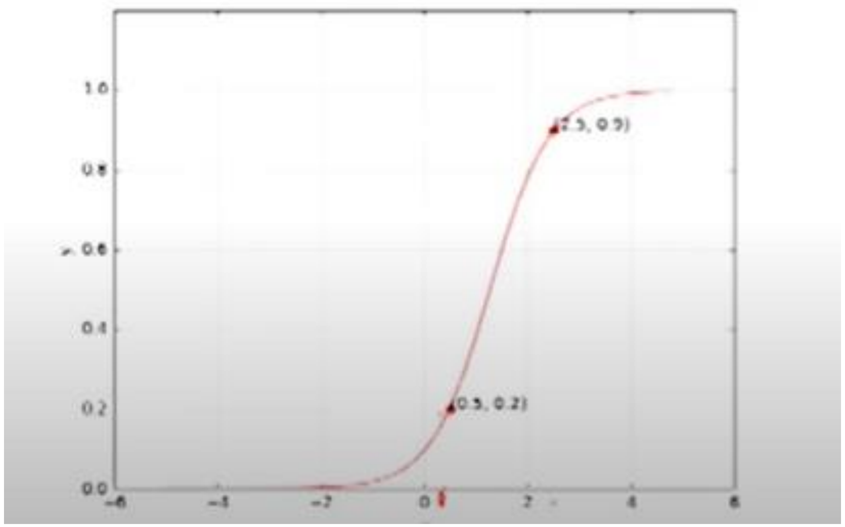
$w = 0.5$, $b = 0$

Example

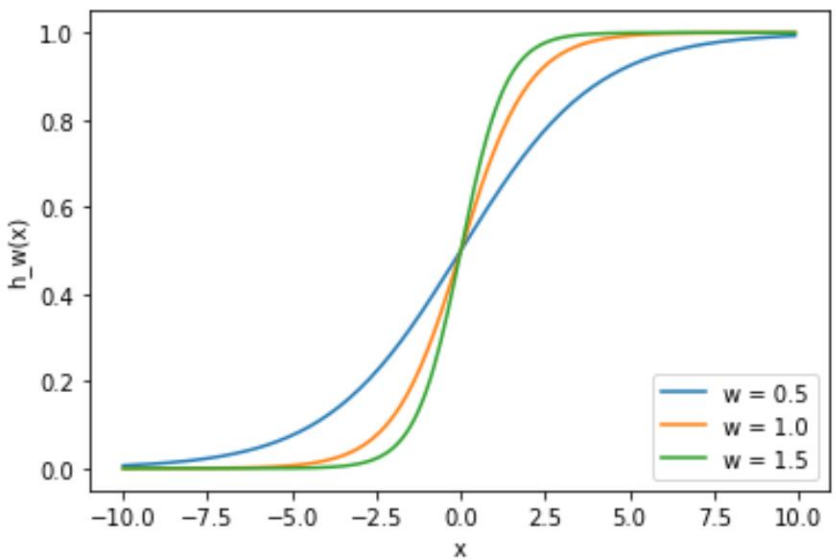
- Ex. data points: (0.5,0.2), (2.5, 0.9)
- Hope to find a sigmoid function that the data points lie on the function



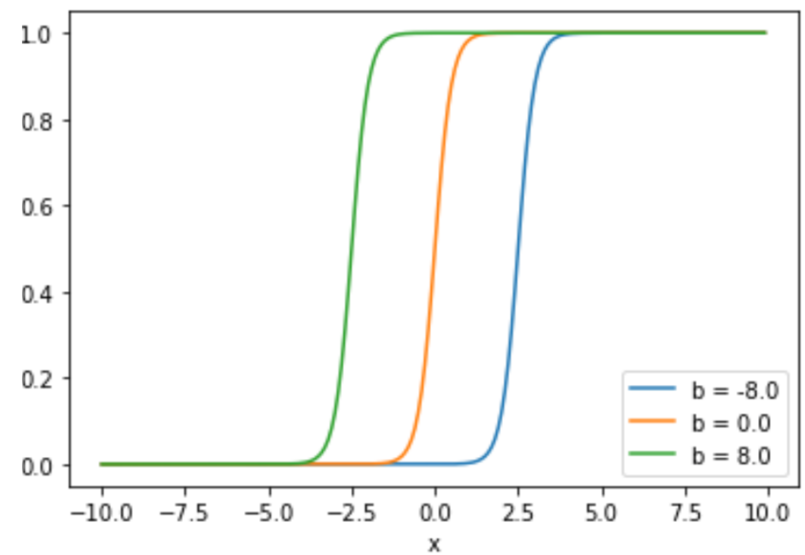
w = 0.5, b = 0



Sigmoid function



Changes w.r.t *w*



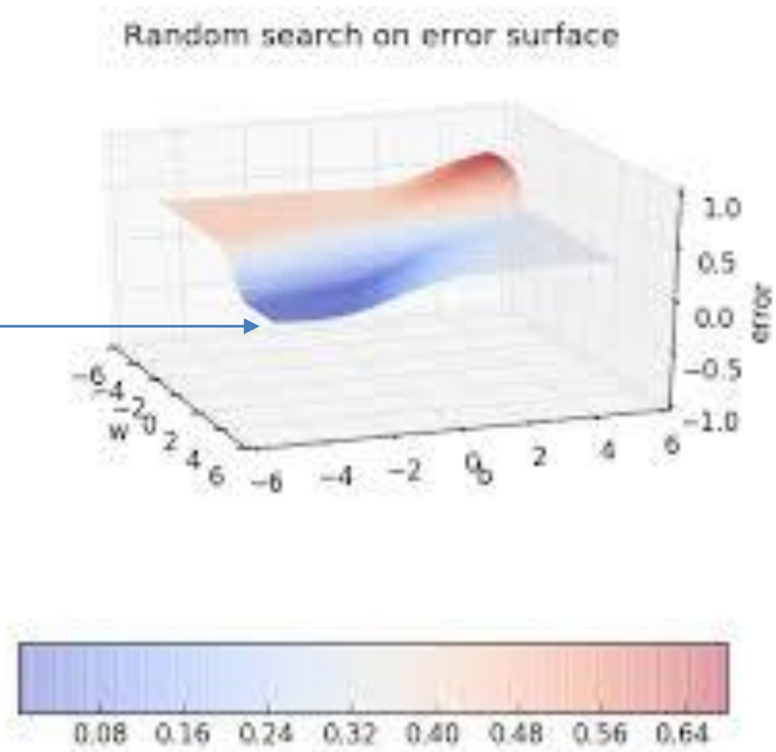
Changes w.r.t *b*

Guesses!!

w	b	Loss
0.5	0	0.073
-0.1	0	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003
1.78	-2.27	0.0000

Error surface

- 2 data points, 2 parameters (*w, b*)



Lowest error

Other *w, b* values
can be checked

- How to handle more data points and parameters??

Gradient Descent

- Find a way to traverse error surface to reach minimum value quickly
- Parameters: $\theta = [w, b]$
- Change in parameters: $\Delta\theta = [\Delta w, \Delta b]$
- New parameters: $\theta_{\text{new}} = \theta + \Delta\theta = [w_{new}, b_{new}]$
  - Change with a small stride η

    $w_{new} = w + \eta.\ \Delta w$

    $b_{new} = b + \eta.\ \Delta b$
- How to choose Δw and Δb?

Gradient Descent

- According to Taylor series:

$$L(\theta + \eta\Delta\theta) = L(\theta) + \eta^*(\Delta\theta)^T \frac{\partial L(\theta)}{\partial \theta} + (\eta^2/2!)^*(\Delta\theta)^T \frac{\partial^2 L(\theta)}{\partial \theta^2}(\Delta\theta) + \ldots$$

$$= L(\theta) + \eta(\Delta\theta)^T \frac{\partial L(\theta)}{\partial \theta} \qquad \text{(if } \eta \text{ is small)}$$

- New loss should be less than old loss
- $(\eta\Delta\theta)$ would be favourable only if:

$$L(\theta + \eta\Delta\theta) < L(\theta)$$

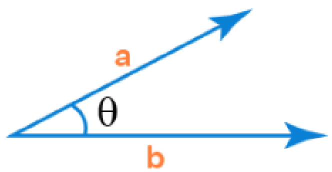
$$L(\theta + \eta\Delta\theta) - L(\theta) < 0$$

- Implies: $(\Delta\theta)^T \frac{\partial L(\theta)}{\partial \theta} < 0$ (η is positive constant)

Gradient Descent

- Desired: $(\Delta\theta)^T \frac{\partial L(\theta)}{\partial \theta} < 0$
  - Want Δθ to be as negative as possible
  - Dot product of two vectors: product of their magnitudes multiplied by the cosine of the angle between them
- Let β be the angle between Δθ and $\frac{\partial L(\theta)}{\partial \theta}$

Then: $-1 \leq \cos(\beta) = \dfrac{(\Delta\theta)^T \frac{\partial L(\theta)}{\partial \theta}}{||\Delta\theta|| * ||\frac{\partial L(\theta)}{\partial \theta}||} \leq 1$



$\cos\theta = \frac{a.b}{|a|\,|b|}$

Most negative: $\cos(\beta) = -1$ when $\beta = 180^\circ$

-> Δθ should be such that it is at 180◦ to the gradient $\frac{\partial L(\theta)}{\partial \theta}$

Gradient Descent

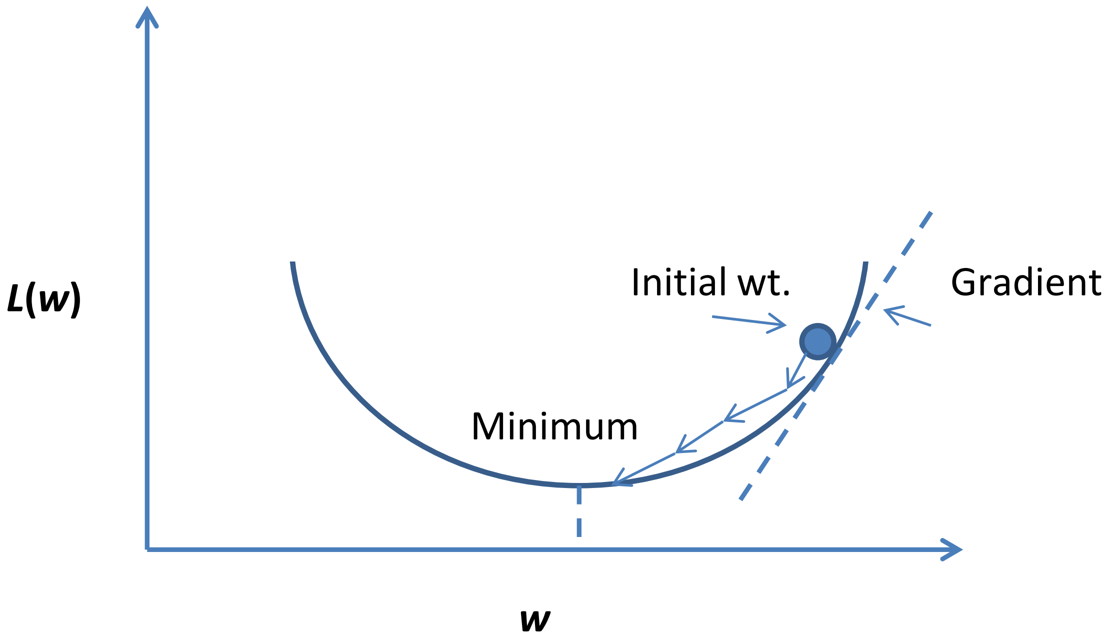
- $(\Delta\theta)^T \frac{\partial L(\theta)}{\partial \theta} < 0$
- Move in direction opposite to gradient (180° w.r.t. the gradient)

$$w_{t+1} = w_t - \eta \frac{\partial L(w,b)}{\partial w} \qquad \text{at } w = w_t \text{ and } b = b_t$$

$$b_{t+1} = b_t - \eta \frac{\partial L(w,b)}{\partial b} \qquad \text{at } w = w_t \text{ and } b = b_t$$

- Repeat till convergence

Gradient Descent



Randomly pick (w_0, w_1)

Compute *L*

Compute gradient $\partial L(w)/\partial w$ - gives ascent at that point

Take small step in opposite direction of gradient

Repeat until convergence

$$w := w - \eta * \partial L(w)/\partial w$$

Gradient Descent

Assuming, $L(\mathrm{w}, \mathrm{b}) = (1/2)*\sum(\hat{y} - y)^2$

$\hat{y} = f(x) = 1/(1+ \mathrm{e}^{-(wx+b)})$

Assuming one data point only:

$$\frac{\partial L(w,b)}{\partial w} = (1/2)*[2*(\hat{y} - y) * \frac{\partial(\hat{y}-y)}{\partial w}]$$

$$= (\hat{y} - y) * \frac{\partial}{\partial w}[\frac{1}{1+e^{-(wx+b)}}]$$

$$\frac{\partial L(w,b)}{\partial w} = (\hat{y} - y) * \hat{y} *(1- \hat{y})*x$$

$$\frac{\partial L(w,b)}{\partial b} = (\hat{y} - y) * \frac{\partial}{\partial b}[\frac{1}{1+e^{-(wx+b)}}]$$

$$= (\hat{y} - y) * \hat{y} *(1- \hat{y})$$

$\partial/\partial w\,[1/(1+ \mathrm{e}^{-(wx+b)})] = f(x)*(1 - f(x))*x$

Gradient Descent

$\frac{\partial L(w,b)}{\partial w}$ = $\sum[(\hat{y}_i - y_i) * \hat{y}_i*(1 - \hat{y}_i)*x_i]$ for all points

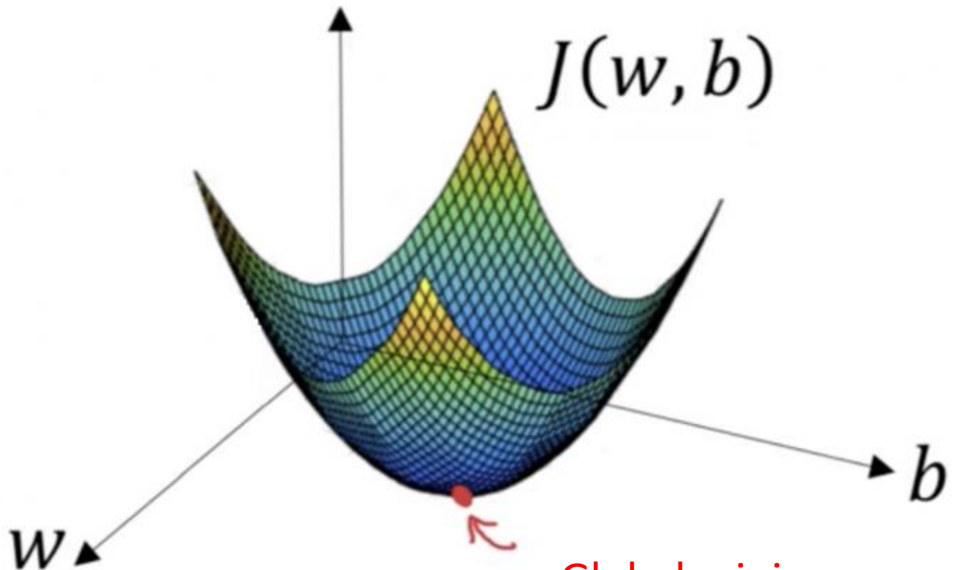
$\frac{\partial L(w,b)}{\partial b}$ = $\sum [(\hat{y}_i - y_i) * \hat{y}_i*(1 - \hat{y}_i)]$ for all points

- Algorithm:
  1. Initialize weights randomly
  2. Loop until convergence:
     1. Compute gradient $\frac{\partial L(w,b)}{\partial w}$, $\frac{\partial L(w,b)}{\partial b}$
     2. Update weights $\boldsymbol{w := w - \eta * \frac{\partial L(w,b)}{\partial w}}$

        $\boldsymbol{b := b - \eta * \frac{\partial L(w,b)}{\partial b}}$

     Return weights

Gradient Descent



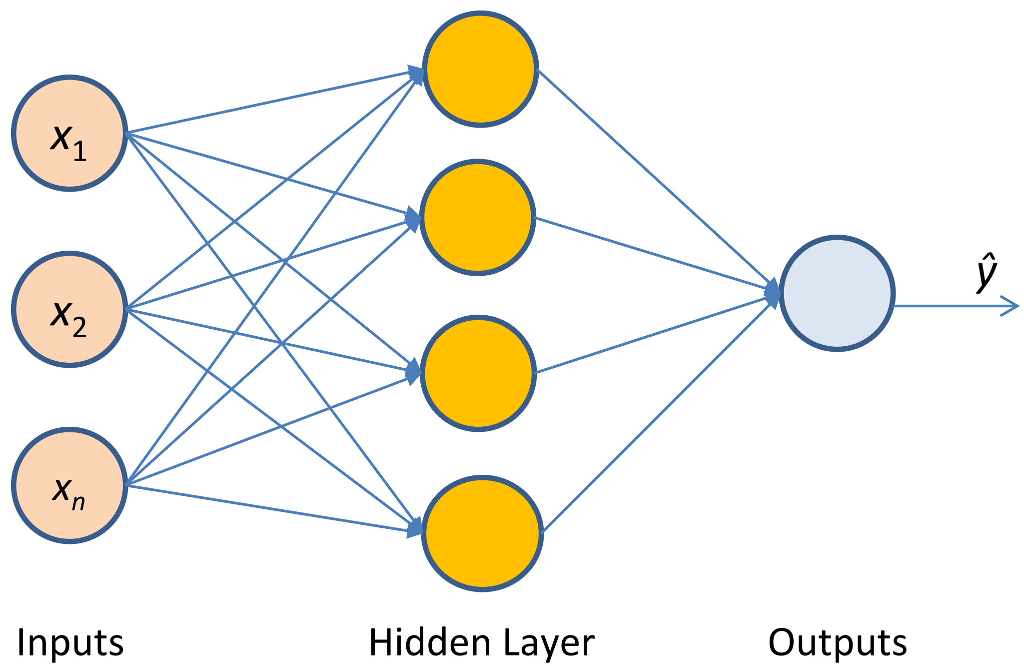
Global minimum
Assuming $J(W)$ to be a convex function

https://builtin.com/data-science/gradient-descent

Representative Power of Multilayer Networks

- A multilayer network of sigmoid neurons with a single hidden layer can be used to approximate any continuous function to any desired precision

Single Hidden Layer Neural Network

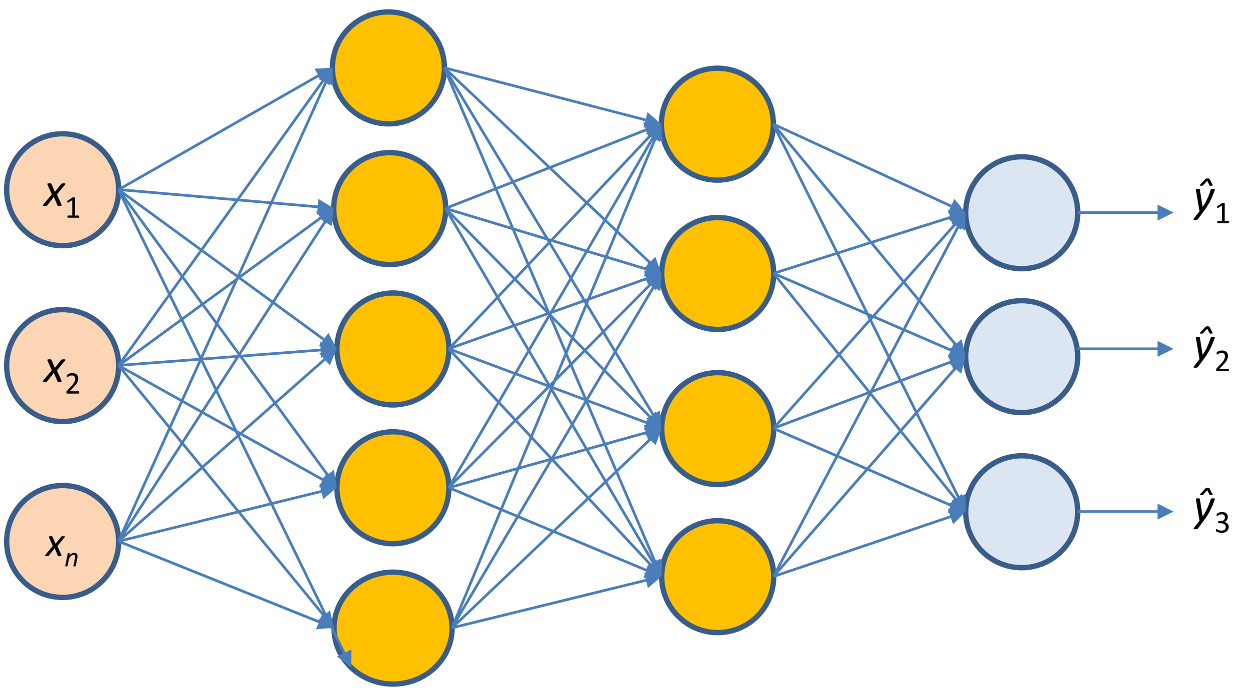


Hidden Layer: States of nodes are unobserved

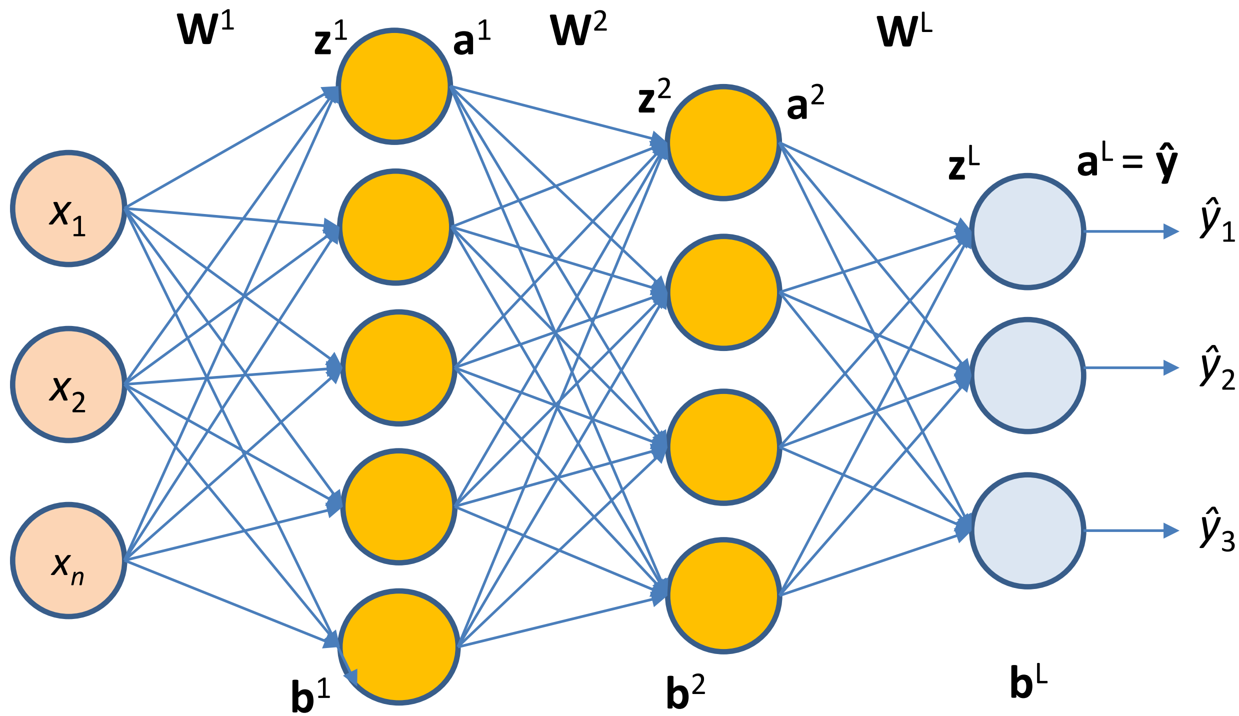
Inputs are densely connected to perceptrons, hence they are called **Dense** layers or **Fully Connected** layers

Feedforward Neural Network

- Input is an *n*-dimensional vector (0^{th} layer) $\in R^n$
- Network has *L*-1 hidden layers
- 1 output layer containing *k* neurons (ex. for *k* classes)
- Each neuron – aggregation and activation

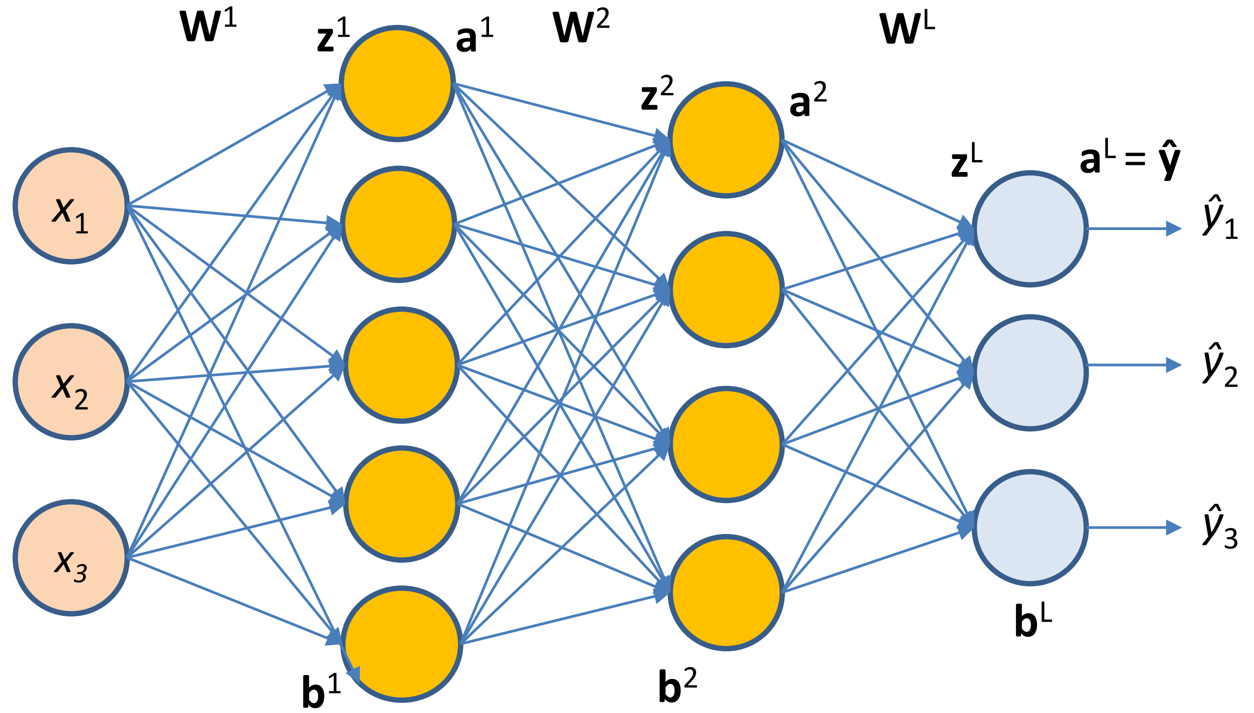


Feedforward Neural Network



Assuming n^i neurons in hidden layer h^i, $sizeof(W^i)= n^i * n^{i-1}$ and $sizeof(b^i)= n^i$ between layers i -1 and i for $0<i<L$

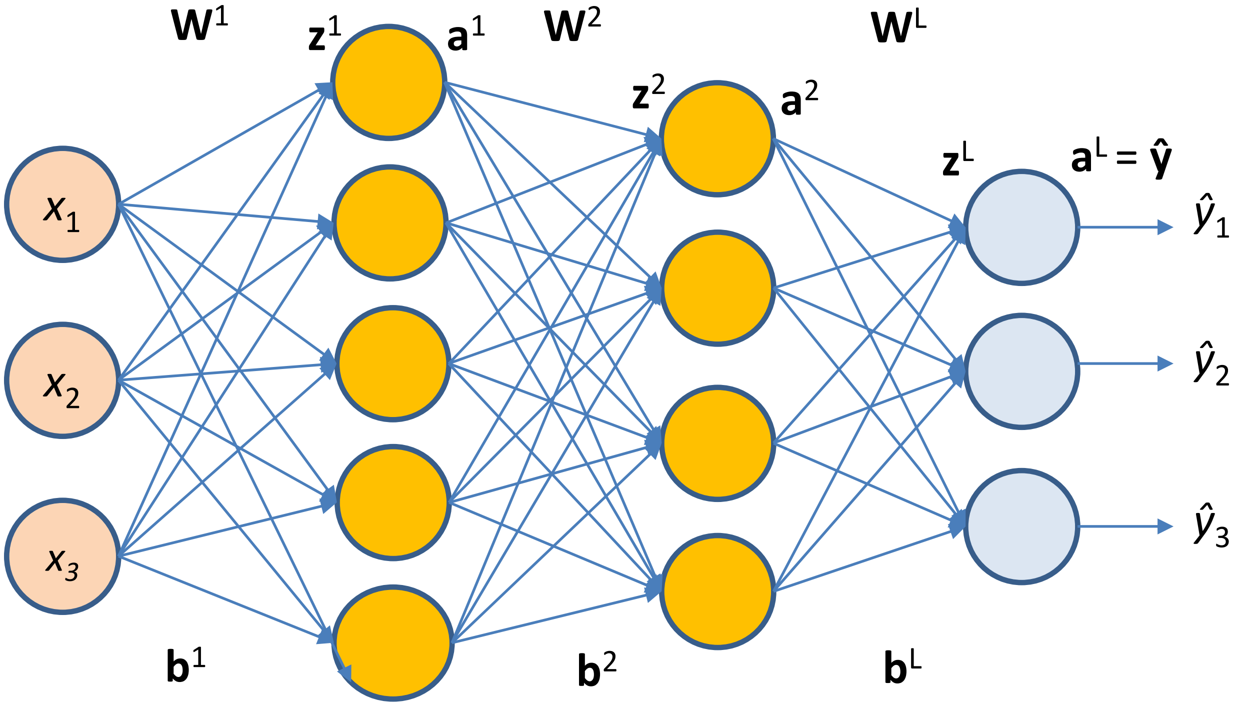
$sizeof(W^L)= n^k * n^{i-1}$ and $sizeof(b^L)= n^k$ between last hidden layer and output layer



Aggregation at layer i : $\mathbf{z}^{\mathbf{i}} = \mathbf{W}^{\mathbf{i}}\,\mathbf{a}^{\mathbf{i-1}} + \mathbf{b}^{\mathbf{i}}$

For first hidden layer: $\mathbf{z}^{1} = \mathbf{W}^{1}\,\mathbf{a}^{0} + \mathbf{b}^{1}$

$$\begin{pmatrix} {z_1}^1 \\ {z_2}^1 \\ \dots \\ {z_5}^1 \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ \dots. & \dots. & \dots. \\ W_{51} & W_{52} & W_{53} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \dots.. \\ b_5 \end{pmatrix} = \begin{pmatrix} \sum W_{1i} x_i + b_1 \\ \sum W_{2i} x_i + b_2 \\ \dots.. \\ \sum W_{5i} \mathrm{x_i} + b_5 \end{pmatrix}$$



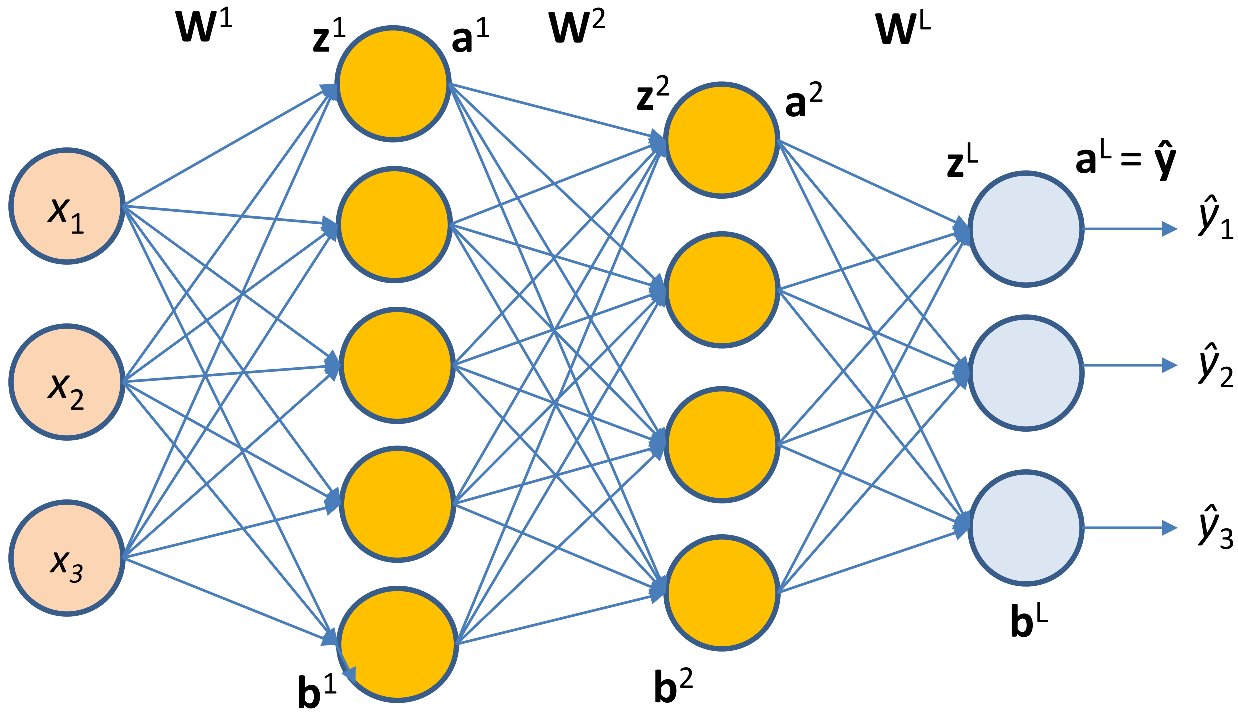
Activation at layer i = $g(\mathbf{z}^i) = g(\mathbf{b}^i + \mathbf{W}^i\, \mathbf{a}^{i-1})$

For first hidden layer: $g(\mathbf{z}^1) = g(\mathbf{b}^1 + \mathbf{W}^1\, \mathbf{a}^0)$

$$\begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_5 \end{pmatrix} = \begin{pmatrix} g(z_1) \\ g(z_2) \\ \dots \\ g(z_5) \end{pmatrix}$$

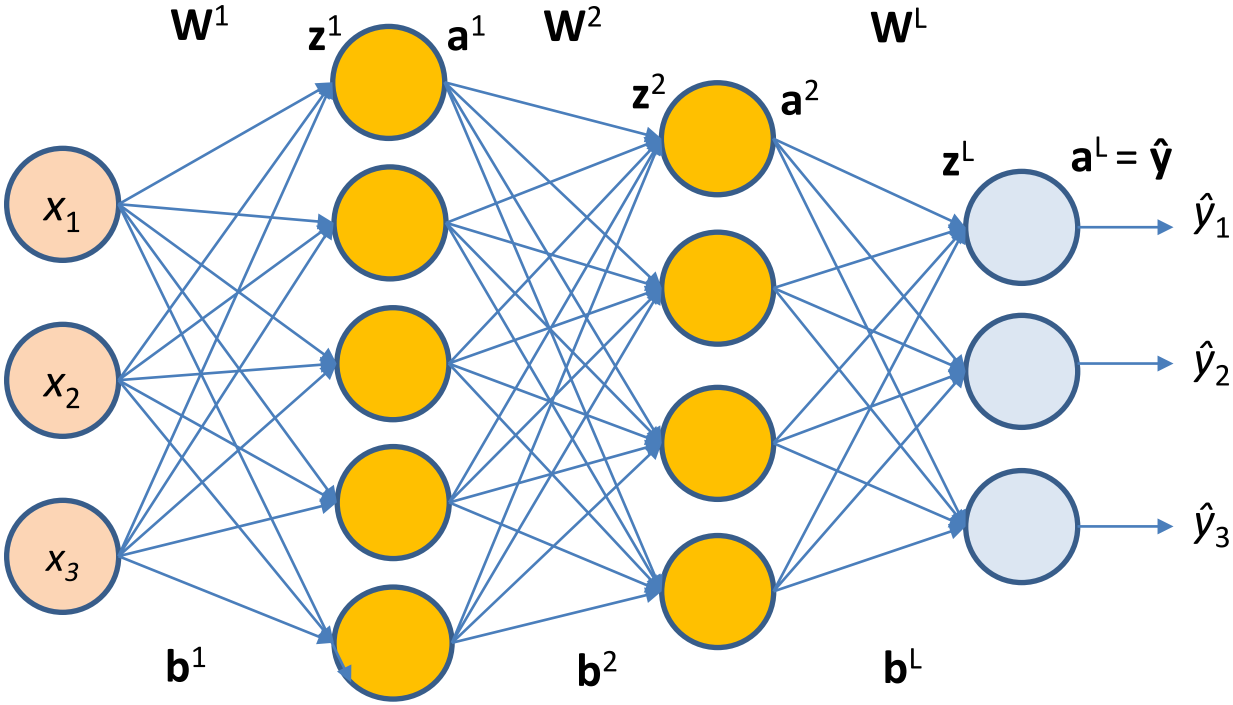
Eg. $g(z_1) = \sigma(z_1) = 1 / (1 + e^{-z_1})$

g: activation function (logistic, tanh, linear etc.)



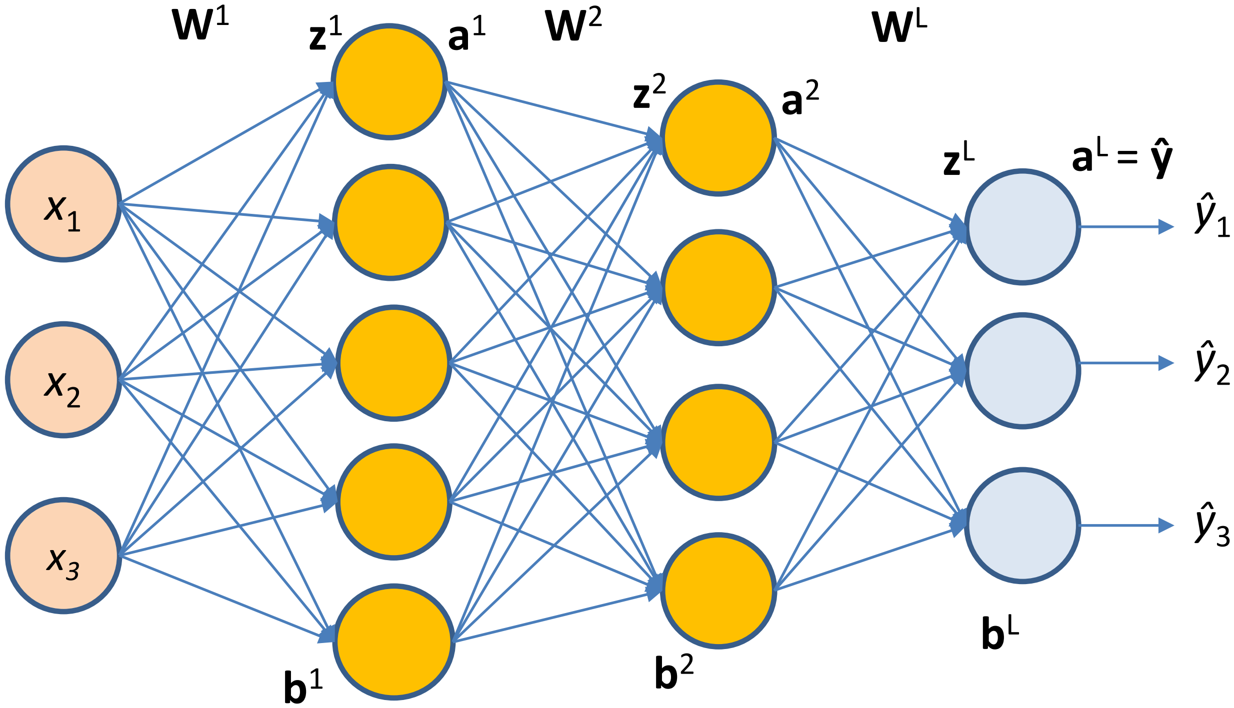
For second hidden layer: $\mathbf{z}^2 = \mathbf{W}^2\,\mathbf{a}^1 + \mathbf{b}^2$

$$\begin{pmatrix} z_1{}^2 \\ z_2{}^2 \\ z_3{}^2 \\ z_4{}^2 \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & W_{13} & W_{14} & W_{15} \\ W_{21} & W_{22} & W_{23} & W_{24} & W_{25} \\ W_{31} & W_{32} & W_{33} & W_{34} & W_{35} \\ W_{41} & W_{42} & W_{43} & W_{44} & W_{55} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} \sum W_{1i}\,a_i + b_1 \\ \sum W_{2i}\,a_i + b_2 \\ \sum W_{3i}\,a_{\mathrm{i}} + b_3 \\ \sum W_{4i}\,a_{\mathrm{i}} + b_4 \end{pmatrix}$$



Activation at layer 2 = $g(\mathbf{z}^2) = g(\mathbf{b}^2 + \mathbf{W}^2\ \mathbf{a}^1)$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} g(z_1) \\ g(z_2) \\ g(z_3) \\ g(z_4) \end{pmatrix}$$



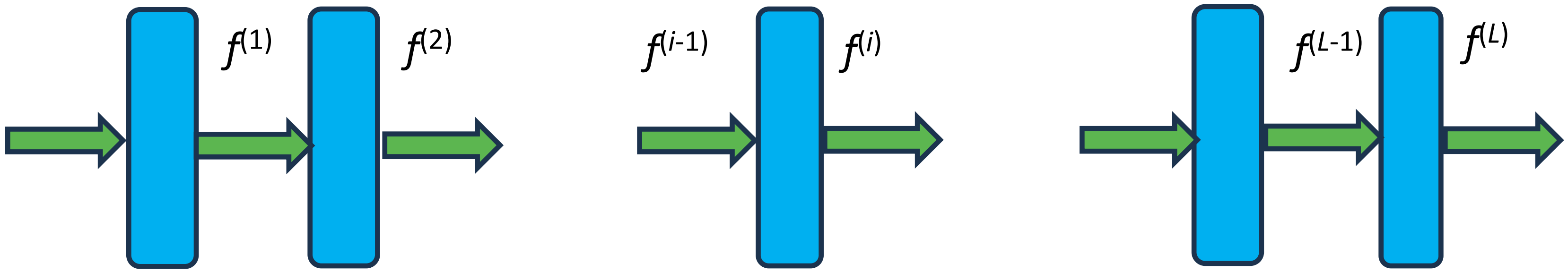
Aggregation at output layer L = z^L = $\mathbf{W}^{\mathbf{L}}\,\mathbf{a}^{\mathbf{L-1}} + \mathbf{b}^{\mathbf{L}}$

$$z_1 = w_{11}a_1 + w_{12}a_2 + w_{13}a_3 + w_{14}a_4 + b$$
$$z_2 = w_{21}a_1 + w_{22}a_2 + w_{23}a_3 + w_{24}a_4 + b$$
$$z_3 = w_{31}a_1 + w_{32}a_2 + w_{33}a_3 + w_{34}a_4 + b$$

Activation at output layer L = $\hat{\mathbf{y}}$ = $\mathrm{g}(z^L) = \mathrm{g}(\mathbf{W}^{\mathbf{L}}\,\mathbf{a}^{\mathbf{L-1}} + \mathbf{b}^{\mathbf{L}})$

$$\begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{pmatrix} = \begin{pmatrix} \mathrm{g}(z_1) \\ \mathrm{g}(z_2) \\ \mathrm{g}(z_3) \end{pmatrix}$$

Neural Network Function



$$\boldsymbol{f^{(L)}\,(\,f^{(L\text{-}1)}\,(\ldots.\,f^{(i)}\,\ldots.(f^{(2)}\,(f^{(1)}\,(X)))))}$$

Learning parameters

In given example, dimensions of parameters:

- $\mathbf{W^1}$: $n^1 * n$ $\mathbf{b^1}$: n^1
- $\mathbf{W^2}$: $n^2 * n^1$ $\mathbf{b^2}$: n^2
- $\mathbf{W^L}$: $k * n^2$ $\mathbf{b^L}$: k

- Assuming L layers and n^i neurons in hidden layer h^i and k neurons in output layer, no. of parameters to be learned:
  - Weights: $(L\text{-}1)*(n^i * n^{i-1}) + (n^2 * k)$ for $0<i<L$
  - Bias: $(L\text{-}1)*n^i + k$

Learning parameters

- **Data:** $\{x_i, y_i\}$ $\quad\quad\quad\quad$ $i = 1..m$
- **Model:**

$$\hat{\mathbf{y}} = f(\mathbf{x}) = g(\mathbf{W}^3 g(\mathbf{W}^2 g(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$$

$$\hat{\mathbf{y}} = [\hat{y}^1 \quad \hat{y}^2 \dots \hat{y}^k]$$

- **Algorithm:** Gradient Descent with back Propagation
- **Loss/Error function:** Sum of squared error loss

$$min \frac{1}{N} \sum_{i=1}^{m} \sum_{j=1}^{k} (\hat{y}_j^i - y_j^i)$$ for i^{th} sample for all classes j

Learning parameters

- Gradient Descent:

  t:=0;

  max_iterations:=1000;

  Initialize $\boldsymbol{\theta_0} := [\mathbf{W^1}_0, \ldots \mathbf{W^L}_0, \mathbf{b^1}_0 \ldots \mathbf{b^L}_0]$;

  while *t*++ < *max_iterations* do

  $\boldsymbol{\theta_{t+1}} := \boldsymbol{\theta_t} - \eta \boldsymbol{\nabla\theta_t}$;

  end

where, $\nabla\theta_t = [\ \frac{\partial L(\theta)}{\partial W_t}, \frac{\partial L(\theta)}{\partial b_t}\ ]^T$

$\nabla\theta$ composed of:

- $\nabla W^1, \nabla W^2, \ldots \nabla W^{L-1} \in R^{n(i-1)xni}$, $\nabla W^L \in R^{nxk}$
- $\nabla b^1, \nabla b^2, \ldots \nabla b^{L-1} \in R^{ni}$, $\nabla b^L \in R^k$

Loss function

- Loss function should capture how much $\hat{y}_i$ deviates from y_i
- $y_i \in R^n$ then squared error loss can be used:

  $$L(\theta) = (1/2m) * \sum (y_i - \hat{y}_i)^2$$

- Problems with squared error loss:

  $$\frac{\partial L(w,b)}{\partial w} = (\hat{y} - y) * \hat{y} * (1 - \hat{y}) * x$$

  – If $y_i = 1$ and $\hat{y}_i \sim 0$, $\frac{\partial L(w,b)}{\partial w} \sim 0$ Undesirable

  – If $y_i = 0$ and $\hat{y}_i \sim 1$, $\frac{\partial L(w,b)}{\partial w} \sim 0$ Undesirable

  – Weight updation becomes very slow

Loss function

- Cross-entropy: gives a measure on how close a predicted distribution is to a true distribution
  - True distribution p_i , Estimated distribution q_i
  - Estimated information content = $- \sum p_i \log_e(q_i)$
  - Capture difference between two probability distributions
  - If prediction is close to actual, cross entropy will be low

$$
\begin{array}{ll}
L(\theta) = -\sum y_c \log_e(\hat{y}_c) & \text{for all } k \text{ classes} \\
y_c = 1 & \text{if } c = t \text{ (true class)} \\
\quad = 0 & \text{otherwise}
\end{array}
$$

$$L(\theta) = -\log_e(\hat{y}_t)$$

Loss function

- Objective function for classification:
  - Cross-entropy Loss

    minimize: $L(\theta) = -\log_e(\hat{y}_t)$

$\hat{y}_t$: predicted probability of correct event

$\log_e(\hat{y}_t)$: probability that x belongs to t^{th} class, log-likelihood of data

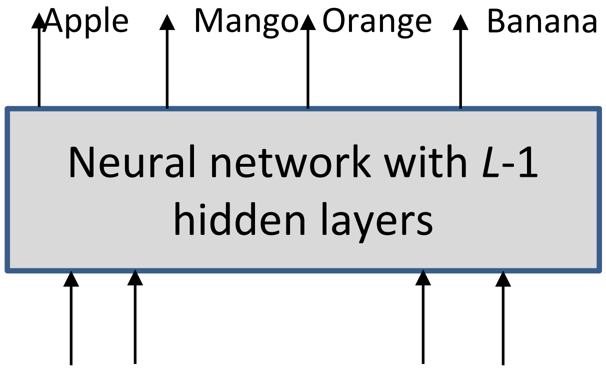
Output Activation Function

- Binary classification:
  - Single neuron in output layer
  - Sigmoid activation function
  - Output between 0-1
  - Above threshold → One class
  - Below threshold → Another class

Output Activation Function

- Output activation function for multi-classification:
  - Sum of outputs should be 1
  - $\hat{y}$ should be a probability distribution
  - Sigmoid – probabilities will be $0<p<1$ but sum not equal to 1

$y_i = \{1 \quad 0 \quad 0 \quad 0\}$



apple

Classification problem

Output Activation Function

- Softmax function

$$z^L = b^L + W^L a^{L-1}$$

$$\hat{y} = g(z^L{}_j) = e^{z}{}_j \,/\, \sum e^{z}{}_j \qquad \text{for } j = 1..k$$

$z^L{}_j$ is j^{th} element of z^L

- Example: $z^L = [10 \quad 20 \quad -30]$

$\hat{y} = [e^{10}/(e^{10} + e^{20} + e^{-30}) \quad e^{20}/(e^{10} + e^{20} + e^{-30}) \quad e^{-30}/(e^{10} + e^{20} + e^{-30})\ ]$

NOTE: Exponent converts –ve values to +ve values

Loss function

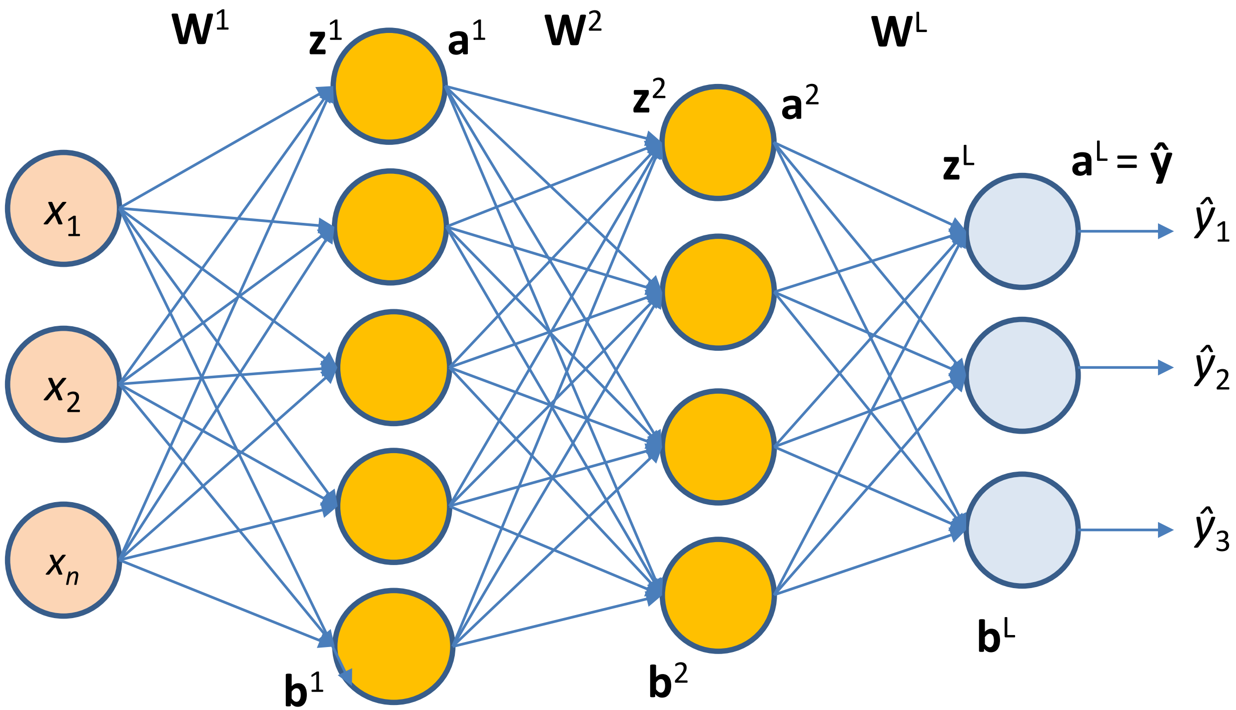
	Outputs	
	Real values	**Probabilities**
Output activation	Linear	Softmax
Loss function	Squared error	Cross-entropy

Backpropagation

How to compute $\nabla\theta$ composed of:

$\nabla W^1, \nabla W^2, \ldots \nabla W^{L-1} \in R^{nxn}$, $\nabla W^L \in R^{nxk}$

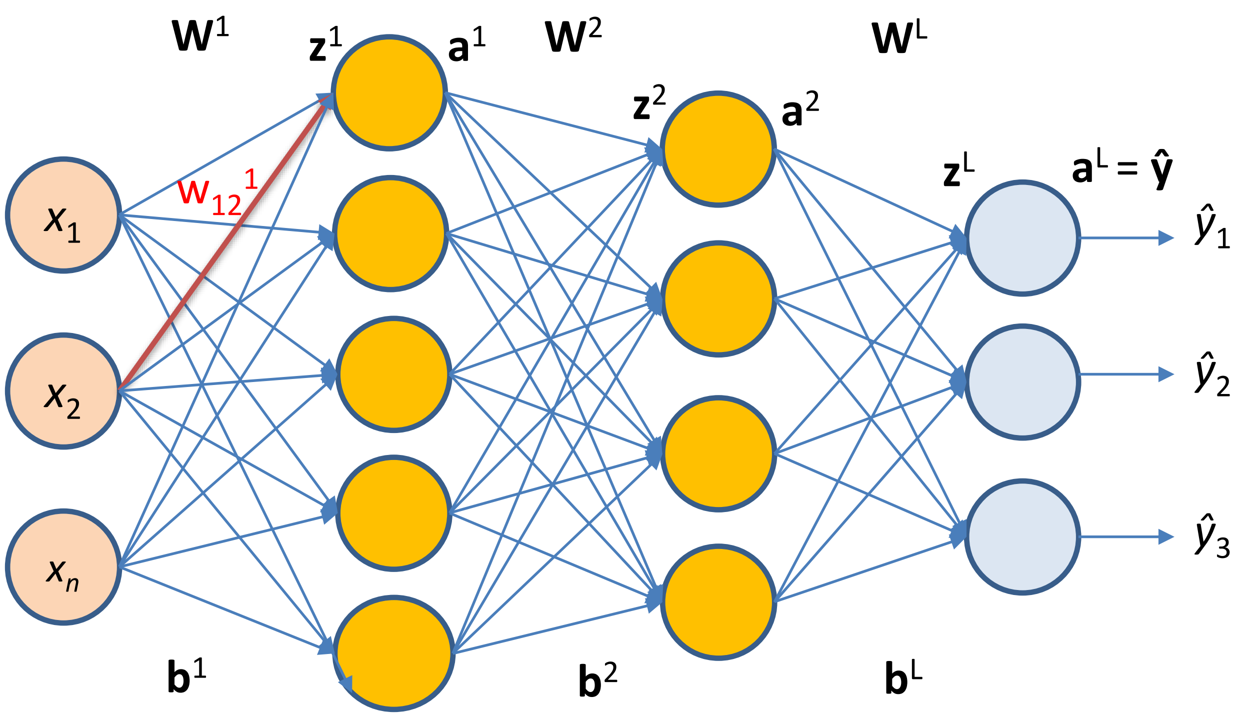
$\nabla b^1, \nabla b^2, \ldots \nabla b^{L-1} \in R^n$, $\nabla b^L \in R^k$



Backpropagation

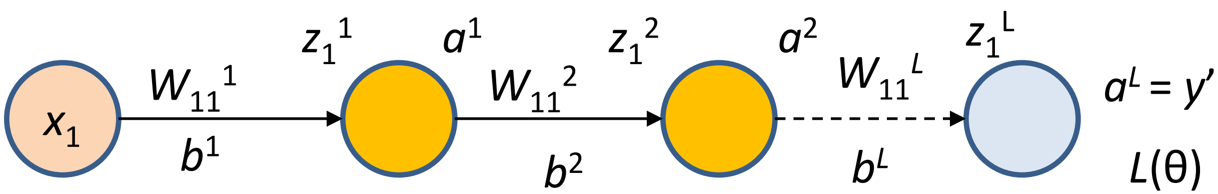
Assuming classification problem, $L(\theta) = -\log_2(\hat{y}_t)$

- To learn weight w_{12}^1 use SGD and compute $\frac{\partial L(w,b)}{\partial W_{12}}$



Backpropagation

Assume a deep thin network, who is responsible for the loss??



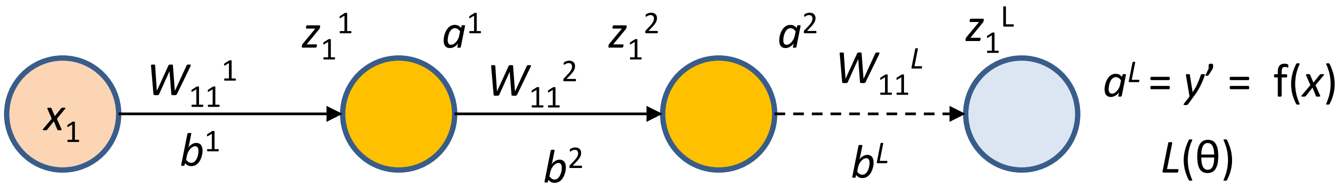
Find derivative by chain rule:

$$\frac{\partial L(\theta)}{\partial W_{11}^1} = \underbrace{\frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1^L}}_{\text{Output layer}} * \underbrace{\frac{\partial z_1^L}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2}}_{\text{Previous hidden layer}} * \underbrace{\frac{\partial z_1^2}{\partial a_1^1} * \frac{\partial a_1^1}{\partial z_1^1}}_{\text{Previous hidden layer}} * \underbrace{\frac{\partial z_1^1}{\partial W_{11}^1}}_{\text{Weights}}$$

If we change W_{11}, how much does the loss change

Backpropagation

Assume a deep thin network



Find derivative by chain rule:

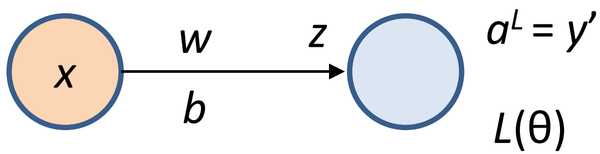
$$\frac{\partial L(\theta)}{\partial W_{11}^1} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1^L} * \frac{\partial z_1^L}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2} * \frac{\partial z_1^2}{\partial a_1^1} * \frac{\partial a_1^1}{\partial z_1^1} * \frac{\partial z_1^1}{\partial W_{11}^1}$$

$$\frac{\partial L(\theta)}{\partial W_{11}^2} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1^L} * \frac{\partial z_1^L}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2} * \frac{\partial z_1^2}{\partial W_{11}^2}$$

$$\frac{\partial L(\theta)}{\partial W_{11}^L} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1^L} * \frac{\partial z_1^L}{\partial W_{11}^L}$$

BACKPROPAGATION WITH SIGMOID OUTPUT ACTIVATION & BINARY CROSS-ENTROPY LOSS

Backpropagation



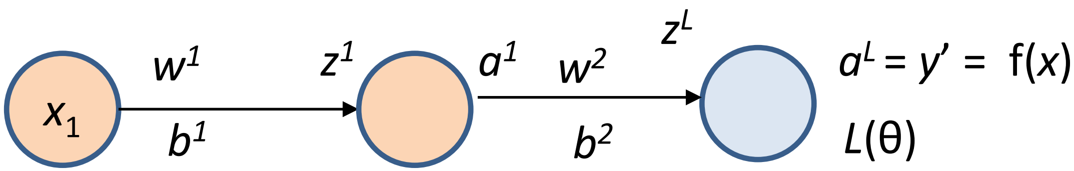
Assuming binary cross-entropy function

L = -y log ŷ - (1 - y) log (1-$\hat{y}$)

$$\frac{\partial L(\theta)}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial L(\theta)}{\partial z} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} = \left(\frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) * \hat{y}(1-\hat{y}) = \hat{y} - y$$

$$\frac{\partial L(\theta)}{\partial w} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial \hat{z}}{\partial w} = (\hat{y} - y) * x$$



$$\frac{\partial L(\theta)}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial L(\theta)}{\partial z^2} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z^2} = \left(\frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) * \hat{y}(1-\hat{y}) = \hat{y} - y \qquad :\delta^L = \frac{\partial L}{\partial \hat{y}} * \sigma'(z^L)$$

$$\frac{\partial L(\theta)}{\partial w^2} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z^2} * \frac{\partial z^2}{\partial w^2} = (\hat{y} - y) * a^1 \qquad :\delta^L * a^1$$

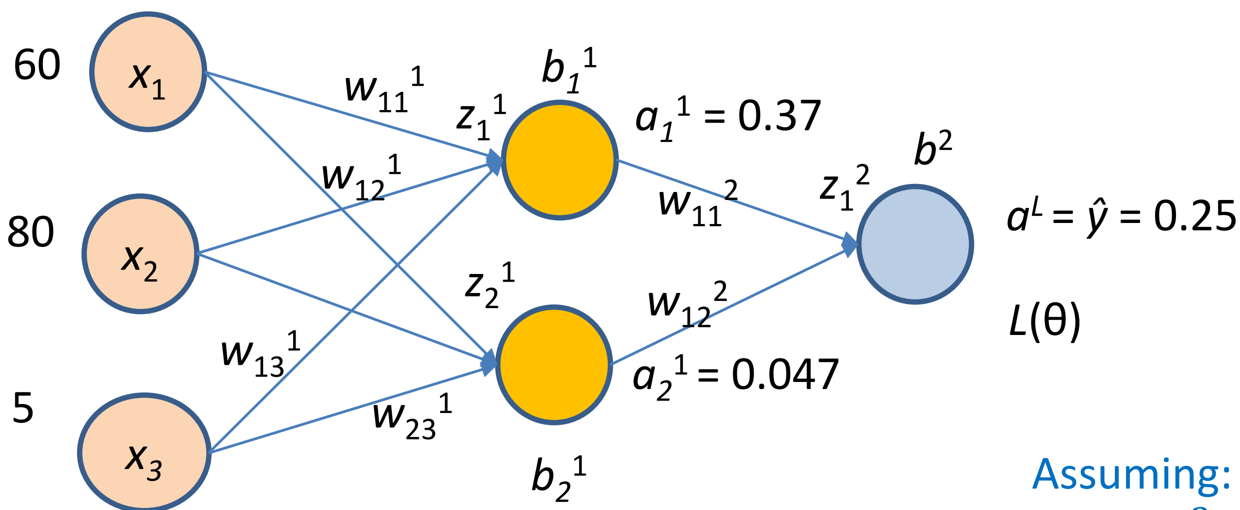
$$\frac{\partial L(\theta)}{\partial a^1} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z^2} * \frac{\partial z^2}{\partial a^1} = (\hat{y} - y) * w^2 \qquad : \delta^L * w^2$$

$$\frac{\partial L(\theta)}{\partial z^1} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z^2} * \frac{\partial z^2}{\partial a^1} * \frac{\partial a^1}{\partial z^1} = (\hat{y} - y) * w^2 * a^1(1 - a^1) \qquad :\delta^1 = \delta^L w^2 * \sigma'(z^1)$$

$$\frac{\partial L(\theta)}{\partial w^1} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z^2} * \frac{\partial z^2}{\partial a^1} * \frac{\partial a^1}{\partial z^1} * \frac{\partial z^1}{\partial w^1} = (\hat{y} - y) * w^2 * a^1(1 - a^1) * x \qquad :\delta^1 * x$$

Backpropagation Equations

- $\delta^L = \frac{\partial L}{\partial \hat{y}} \odot \sigma'(z^L)$
- $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
- $\frac{\partial L}{\partial b_j^l} = \delta_j^l$
- $\frac{\partial L}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$



$y=1$

Assuming:
Initial w_{11}^2=12, w_{11}^1=0.1
η = 0.01

During forward propagation:
a_1^1 = 0.37, z_1^1 = 0.5

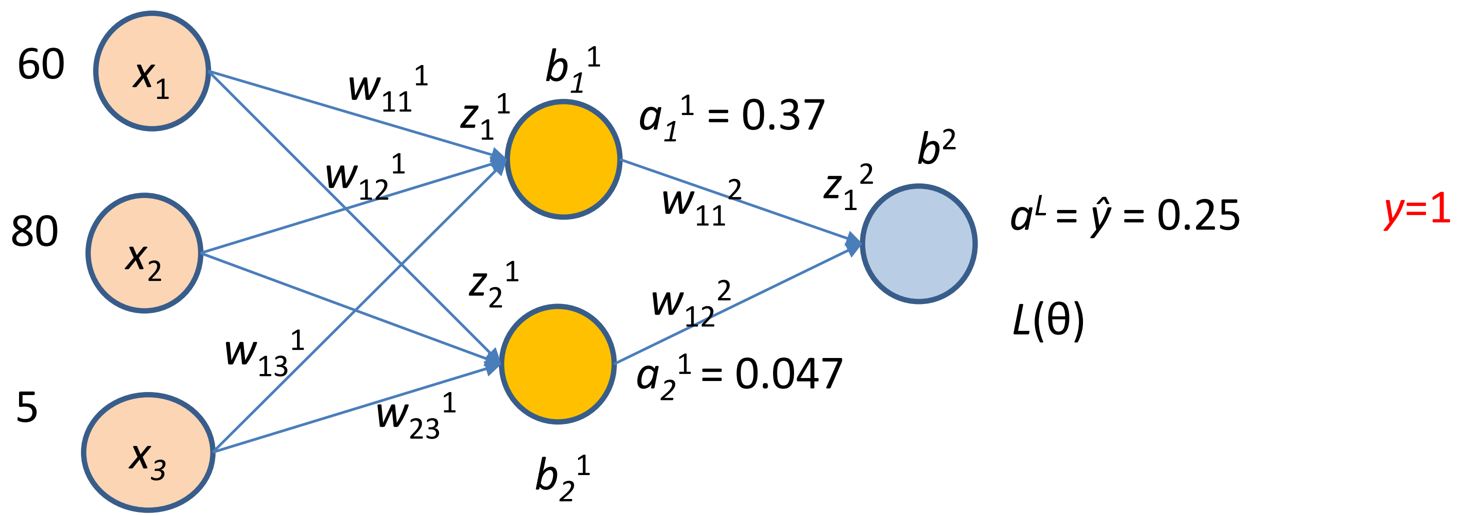
$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1^2} * \frac{\partial z_1^2}{\partial w_{11}^2}$$

$$\frac{\partial L(\theta)}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}} = \frac{-1}{0.25} = -4$$

$$\frac{\partial L(\theta)}{\partial z_1^2} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1^2} = \hat{y} - y = 0.25 - 1 = -0.75$$

$$\frac{\partial L(\theta)}{\partial w_{11}^2} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1^2} * \frac{\partial z}{\partial w_{11}^2} = -0.75 * a_1^1 = -0.75 * 0.37 = -0.2775$$

$$w_{11}^{2*} = w_{11}^2 - \eta * \frac{\partial L}{\partial w_{11}^2} = 12 - 0.01 * (-0.2775) = 12.0028$$



$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1^2} * \frac{\partial z_1^2}{\partial a_1^1} * \frac{\partial a_1^1}{\partial z_1^1} * \frac{\partial z_1^1}{\partial w_{11}^1}$$

$$= -0.75 * w_{11}^2 * a_1^1(1 - a_1^1) * x_1$$

= -0.75 * 12 * 0.37 (1- 0.37)*60

= -125.874

$$w_{11}^{1*} = w_{11}^1 - \eta * \frac{\partial L}{\partial w_{11}^1} = 0.1 - 0.01 * (-125.874) = 1.35$$

Assignment: Compute $\frac{\partial L}{\partial w_{13}^1}$