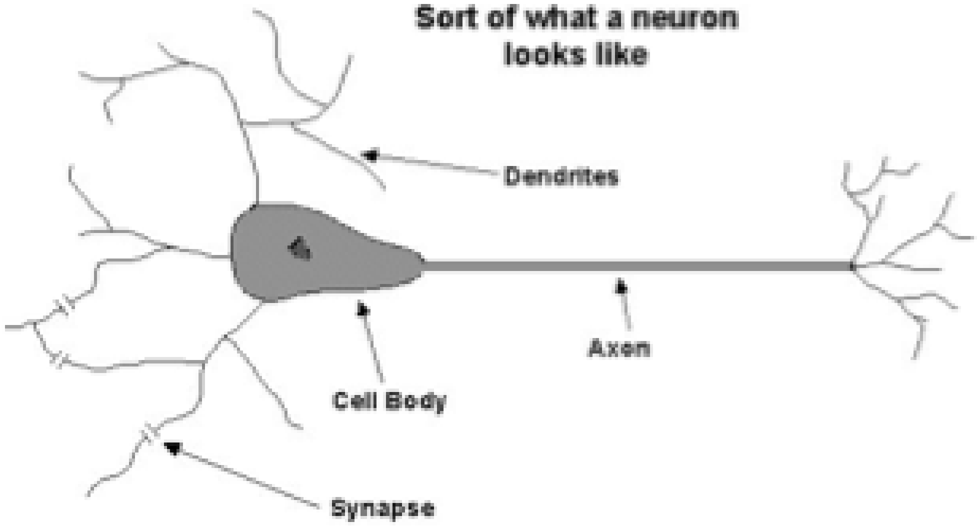
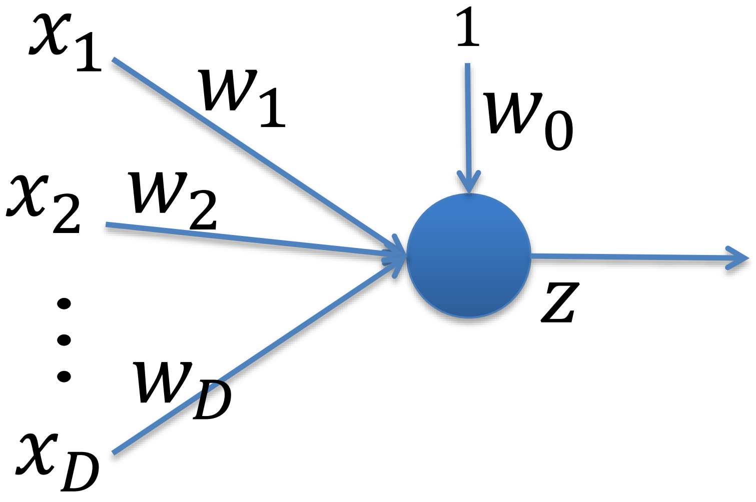
Artificial Neural Networks

by

Ahmet Sacan

Artificial Neuron (Perceptron)

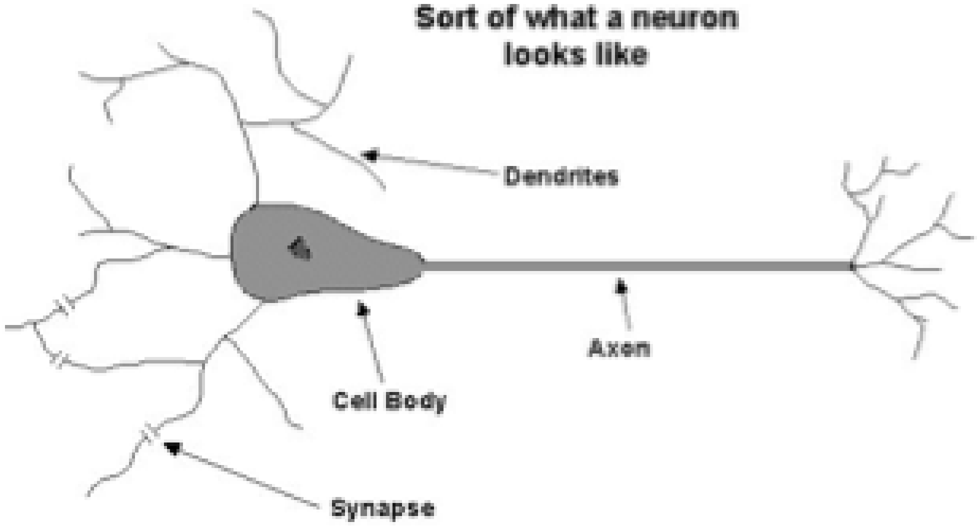


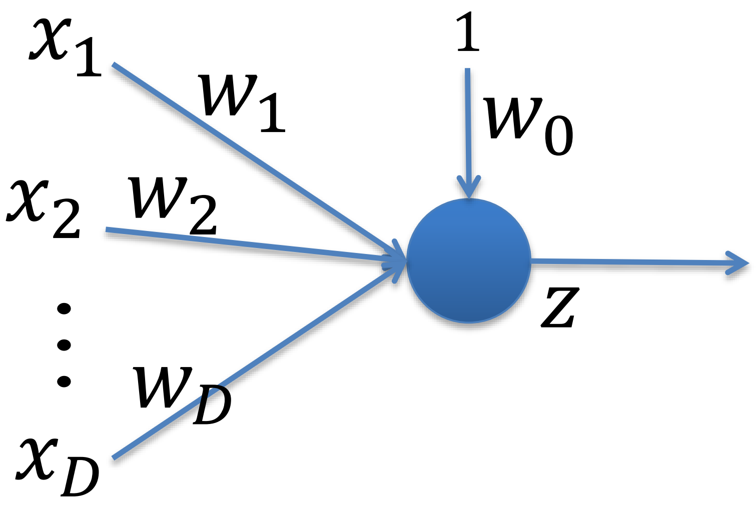


Total Input Signal:

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_D x_D \ + w_0$$

Artificial Neuron



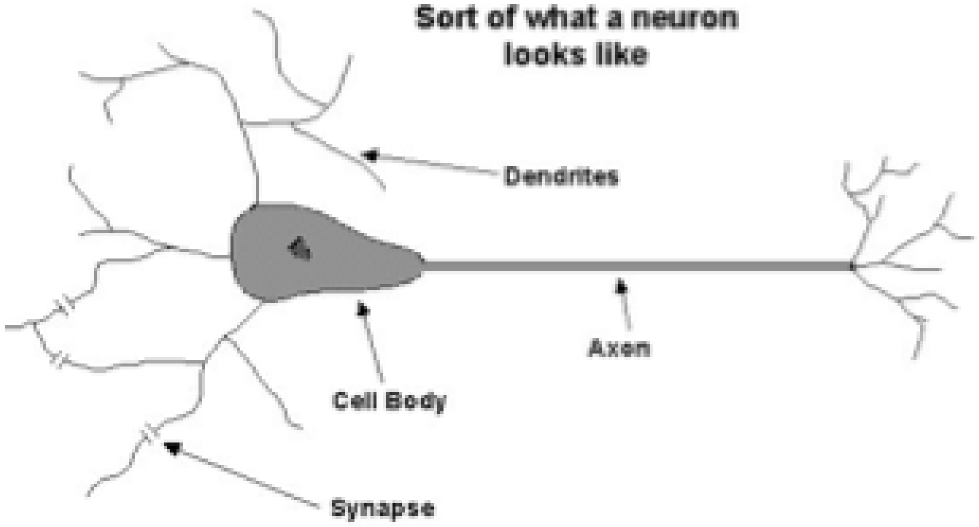


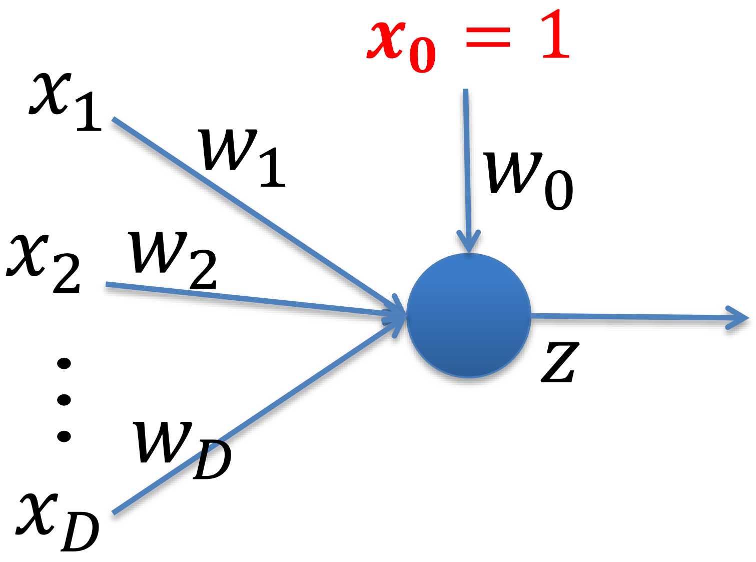
Total Input Signal:

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_D x_D + w_0$$

$$z = w_0 + \sum_{i=1}^{D} w_i x_i$$

Artificial Neuron





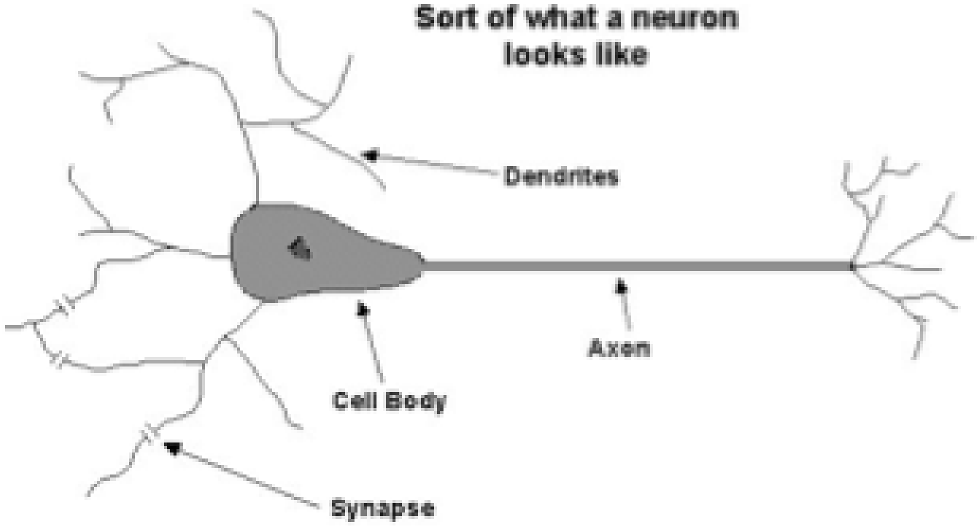
Total Input Signal:

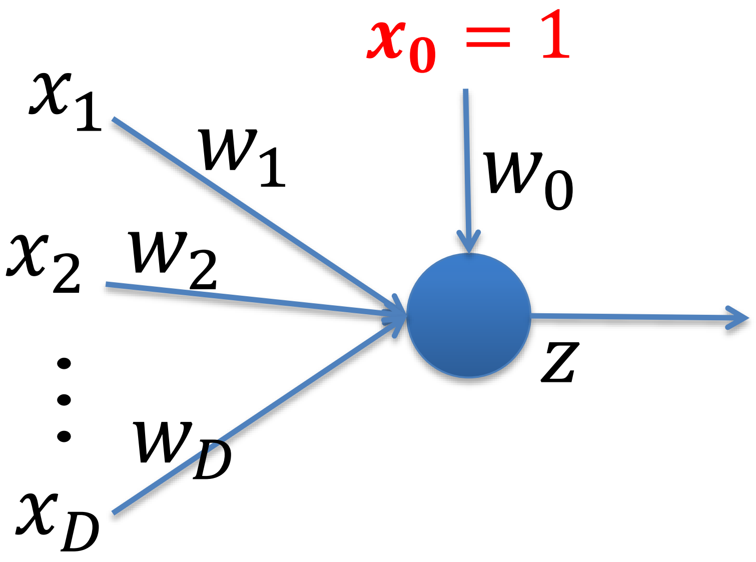
$$z = w_1 x_1 + w_2 x_2 + \cdots + w_D x_D + w_0$$

$$z = w_0 + \sum_{i=1}^{D} w_i x_i$$

$$z = \sum_{i=\mathbf{0}}^{D} w_i x_i$$

Artificial Neuron





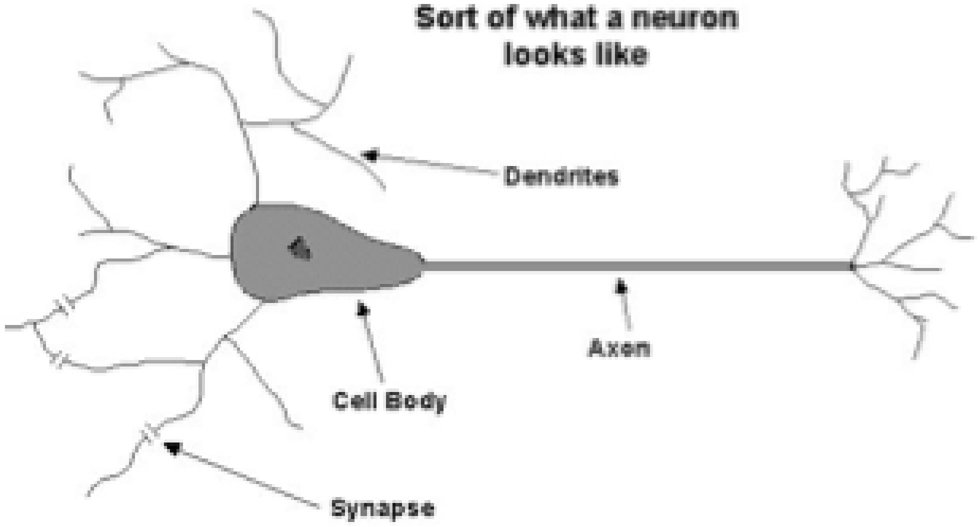
Total Input Signal:

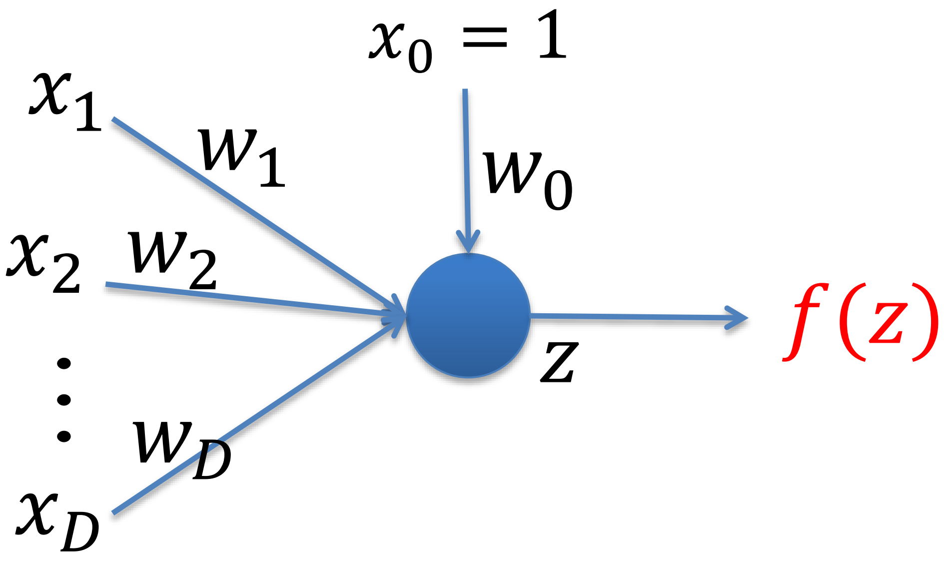
$$z = w_1 x_1 + w_2 x_2 + \cdots + w_D x_D + w_0$$

$$z = w_0 + \sum_{i=1}^{D} w_i x_i$$

$$z = \sum_{i=\mathbf{0}}^{D} w_i x_i = X \cdot W$$

Artificial Neuron



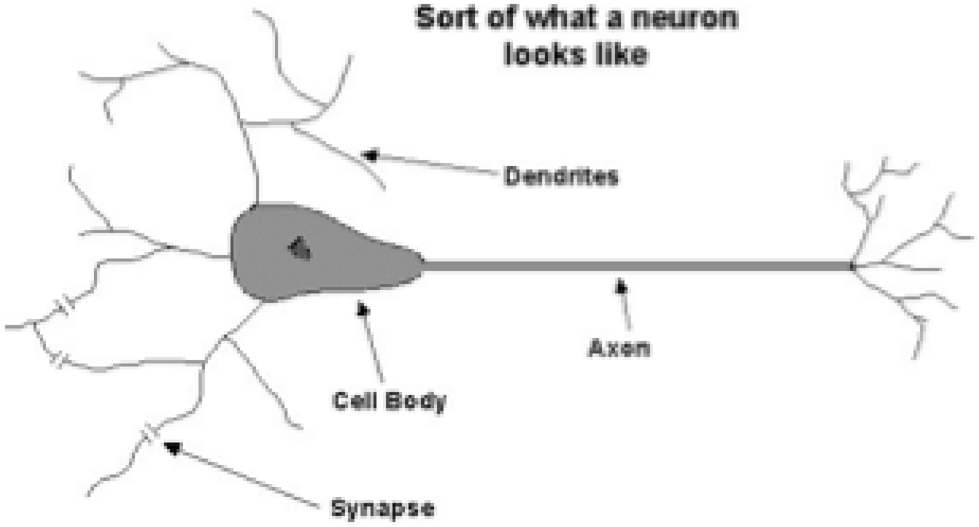


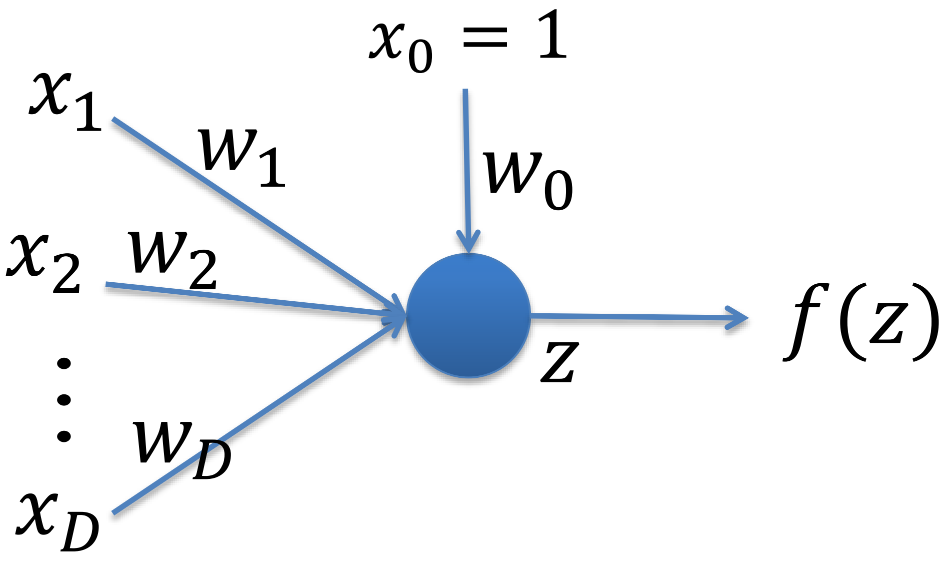
Total Input Signal: $z = \sum_{i=\mathbf{0}}^{D} w_i x_i$

Neuron Output (Activation Function):

$f(z), \quad f(X, W)$

Artificial Neuron





Total Input Signal: $z = \sum_{i=\mathbf{0}}^{D} w_i x_i$

Activation Function:

Linear: $f(z) = z$

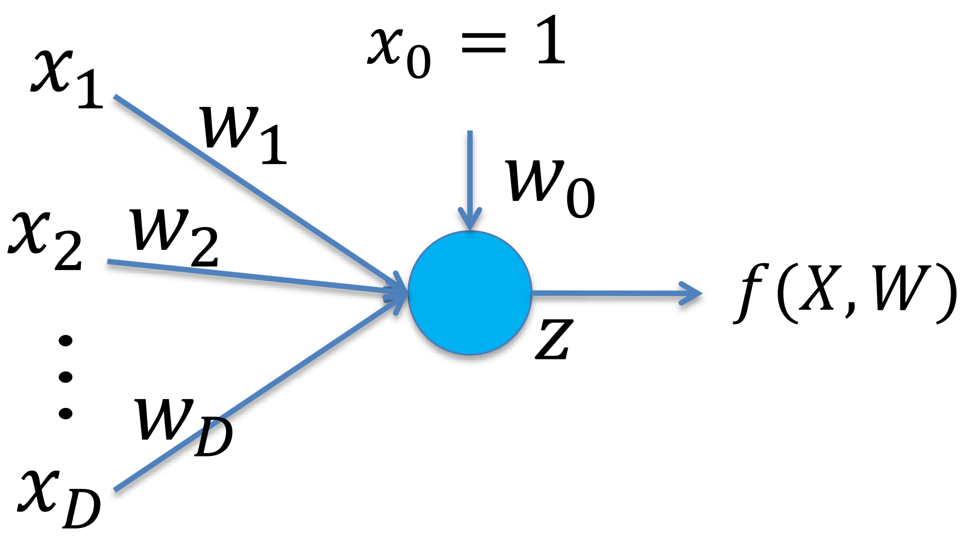
Sigmoid: $f(z) = 1/(1 + e^{-z})$

"Training" a Neuron

- Training Data:

	x_0	x_1	x_2	...	x_D
X^1	1	2.3	5.6	...	7.9
X^2	1	6.6	0.4	...	4.3
	1			...	
	1			...	
	1			...	
	1			...	
X^N	1			...	

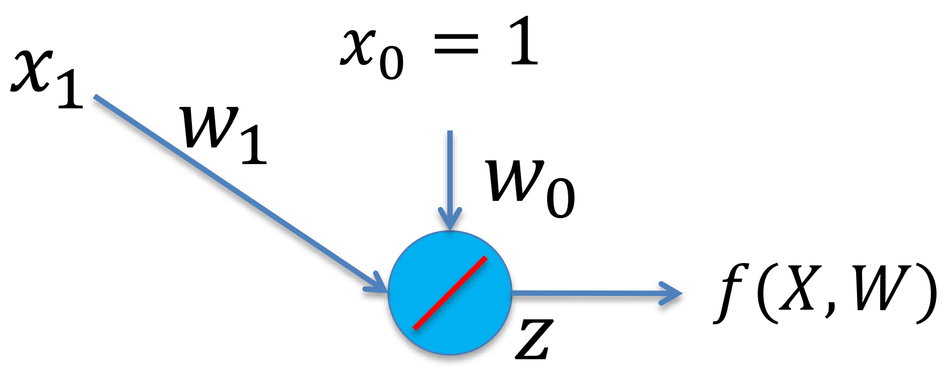
T
2.7
3.2
...
T^N



"Training" a Linear Neuron

- Consider a simpler dataset:

	x_0	x_1	x_2	...	x_D		T
X^1	1	2.3	5.6	...	7.9		2.7



- And a single dimension:

	x_0	x_1		T
X^1	1	2.3		2.7

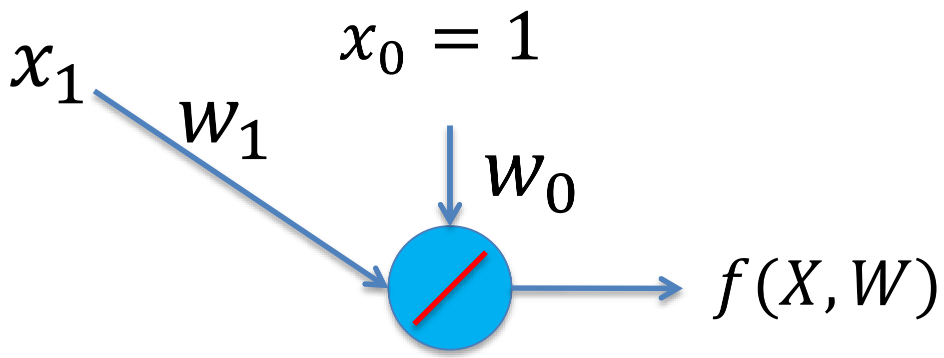
- Linear Activation Function:

$$f(X,W) = \sum_{i=0}^{D} w_i x_i$$
$$= w_0 x_0 + w_1 x_1$$

"Training" a Linear Neuron

	x_0	x_1
X^1	1	2.3

T
2.7



$$Y = f(X,W) = w_0 x_0 + w_1 x_1$$

- e.g., $w_0 = 5$, $w_1 = 7$
- $f(X,W) = 5 * 1 + 7 * 2.3 = \mathbf{21.1}$

	x_0	x_1
X^1	1	2.3

w_0	w_1
5	**7**

Y	T
21.1	2.7

- Perceptron output doesn't match target 2.7. We need to "adjust" w_0 and w_1 so perceptron output is more correct.

"Training" a Linear Neuron

- Define error: $E = \frac{1}{2}(Y - T)^2$

	x_0	x_1
X^1	1	2.3

w_0	w_1
5	**7**

Y	T	E
21.1	2.7	**169.3**

- Adjust w_0 and w_1 to minimize the error E.
- Many function optimization methods can be used to find the best w_0 and w_1. Let's use **Gradient Descent**

- $w_0^{next} = w_0 - \eta \frac{\partial E}{\partial w_0}$, $w_1^{next} = w_1 - \eta \frac{\partial E}{\partial w_1}$

"Training" a Linear Neuron

$$E = \frac{1}{2}(Y - T)^2$$

$$Y = f(X, W) = z = \sum_{i=0}^{D} w_i x_i = w_0 x_0 + w_1 x_1$$

$$E = \frac{1}{2}(w_0 x_0 + w_1 x_1 - T)^2$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w_0}} = x_0(w_0 x_0 + w_1 x_1 - T), \quad \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w_1}} = \mathrm{x}_1(w_0 x_0 + w_1 x_1 - T)$$

- $w_0^{next} = w_0 - \eta \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w_0}} \quad , \quad w_1^{next} = w_1 - \eta \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w_1}}$

"Training" a Linear Neuron

$$\frac{\partial E}{\partial w_0} = x_0(w_0x_0 + w_1x_1 - T), \quad \frac{\partial E}{\partial w_1} = x_1(w_0x_0 + w_1x_1 - T)$$

$$w_0^{next} = w_0 - \eta\frac{\partial E}{\partial w_0} \quad , \quad w_1^{next} = w_1 - \eta\frac{\partial E}{\partial w_1}$$

- e.g., $\eta = 0.1$:

	x_0	x_1
X^1	1	2.3

w_0	w_1
5	**7**

Y	T	E
21.1	2.7	**169.3**

$$w_0^{next} = 5 - 0.1 * 18.4 = 3.16$$

$$w_1^{next} = 7 - 0.1 * 42.3 = 2.77$$

	x_0	x_1
X^1	1	2.3

w_0	w_1
3.16	**2.77**

Y	T	E
	2.7	

Calculate next error & repeat...

After 1 iteration:

	x_0	x_1		w_0	w_1		Y	T	E
X^1	1	2.3		**3.16**	**2.77**		**9.53**	2.7	**23.3**

$$w_0^{next} = 3.16 - 0.1 * 6.83 = 2.48$$

$$w_1^{next} = 2.77 - 0.1 * 15.7 = 1.20$$

After 2 iterations:

	x_0	x_1		w_0	w_1		Y	T	E
X^1	1	2.3		**2.48**	**1.20**		**5.23**	2.7	**3.2**

After 3 iterations:

	x_0	x_1		w_0	w_1		Y	T	E
X^1	1	2.3		**2.22**	**0.62**		**3.64**	2.7	**0.44**

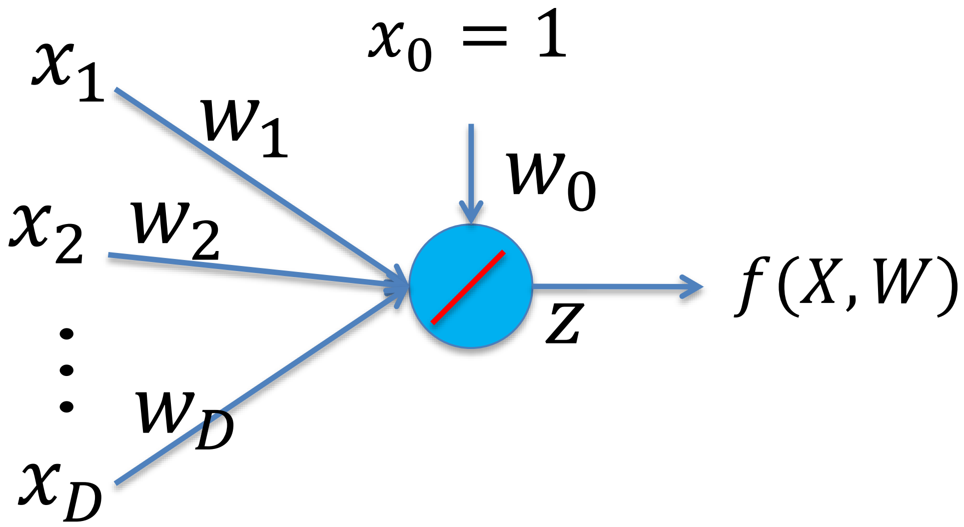
...

After 10 iterations:

	x_0	x_1		w_0	w_1		Y	T	E
X^1	1	2.3		**2.07**	**0.27**		**2.7009**	2.7	**4E-07**

What if more dimensions in the data?

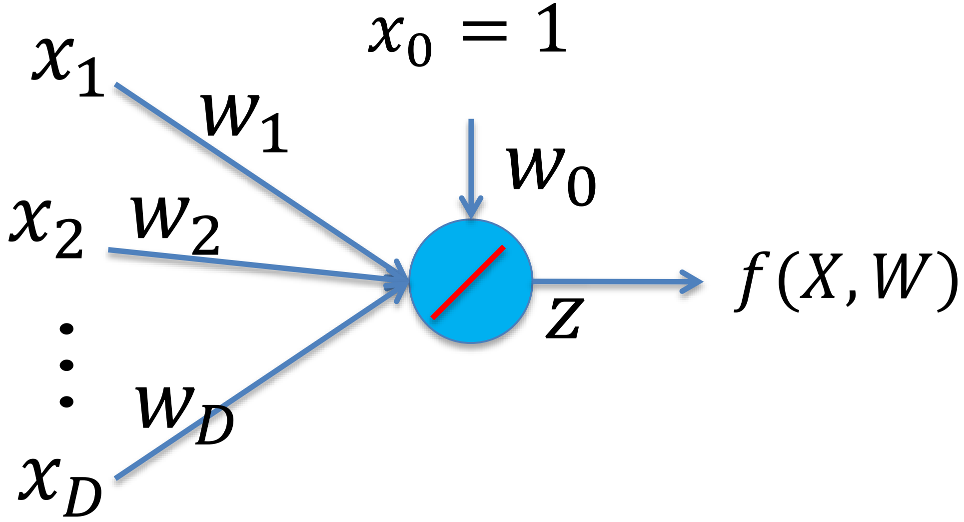
	x_0	x_1	x_2	...	x_D	T
X^1	1	2.3	5.6	...	7.9	2.7



- $w_i^{next} = w_i - \eta \frac{\partial E}{\partial w_i}$
- $E = \frac{1}{2}(Y - T)^2$
- $\frac{\partial E}{\partial w_i} = (Y - T)\frac{\partial Y}{\partial z}\frac{\partial z}{\partial w_i}$
- $Y = f(z) = z$
- $z = \sum_{i=0}^{D} w_i x_i = \mathrm{w_0 x_0} + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D$
- $\frac{\partial z}{\partial w_i} = x_i$
- $\frac{\partial E}{\partial w_i} = (Y - T)x_i$
- $w_i^{next} = w_i - \eta\,(Y - T)x_i$

What if different activation function?

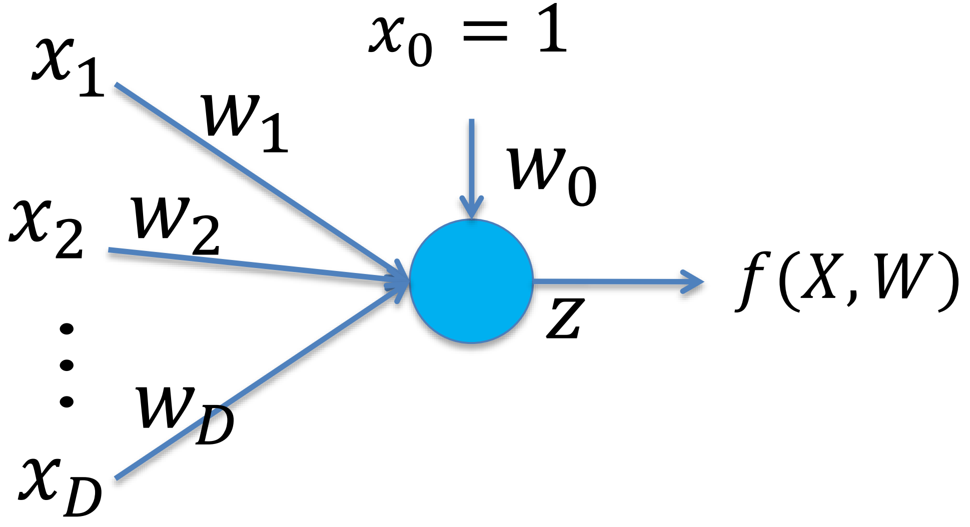
	x_0	x_1	x_2	...	x_D		T
X^1	1	2.3	5.6	...	7.9		2.7



- $w_i^{next} = w_i - \eta \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w_i}}$
- $E = \frac{1}{2}(Y - T)^2$
- $\frac{\partial E}{\partial w_i} = (Y - T)\frac{\partial Y}{\partial z}\frac{\partial z}{\partial w_i}$
- $Y = f(z) = z$
- $z = \sum_{i=0}^{D} w_i x_i = \mathrm{w}_0\mathrm{x}_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D$
- $\frac{\partial z}{\partial w_i} = x_i$
- $\frac{\partial E}{\partial w_i} = (Y - T)x_i$
- $w_i^{next} = w_i - \eta\,(Y - T)x_i$

What if different activation function?

	x_0	x_1	x_2	...	x_D	T
X^1	1	2.3	5.6	...	7.9	2.7

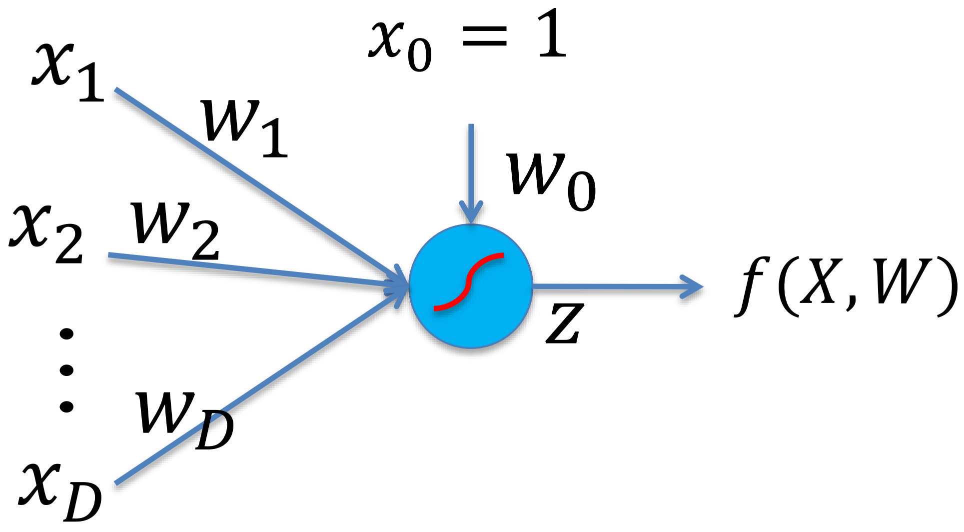


- $w_i^{next} = w_i - \eta \dfrac{\partial \boldsymbol{E}}{\partial \boldsymbol{w_i}}$
- $E = \dfrac{1}{2}(Y - T)^2$
- $\dfrac{\partial E}{\partial w_i} = (Y - T)\dfrac{\partial Y}{\partial z}\dfrac{\partial z}{\partial w_i}$
- $Y = f(z)$
- $z = \sum_{i=0}^{D} w_i x_i = \mathrm{w}_0\mathrm{x}_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D$
- $\dfrac{\partial z}{\partial w_i} = x_i$
- $\dfrac{\partial E}{\partial w_i} = (Y - T)f'(z)x_i$
- $w_i^{next} = w_i - \eta(Y - T)f'(z)x_i$

What if <u>sigmoid</u> activation function?

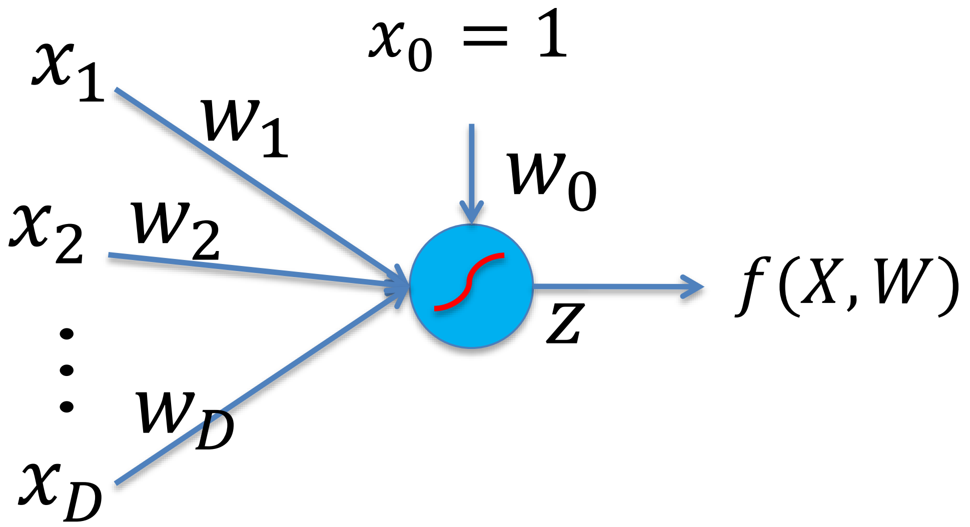
	x_0	x_1	x_2	...	x_D		T
X^1	1	2.3	5.6	...	7.9		2.7

- $\frac{\partial E}{\partial w_i} = (Y - T)f'(z)x_i$
- $Y = f(z) = \frac{1}{1+e^{-z}}$
- $f'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$



What if sigmoid activation function?

	x_0	x_1	x_2	...	x_D	T
X^1	1	2.3	5.6	...	7.9	2.7



- $\frac{\partial E}{\partial w_i} = (Y - T) f'(z) x_i$
- $Y = f(z) = \frac{1}{1+e^{-z}}$
- $f'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$

$$= \frac{1}{(1+e^{-z})}\left(\frac{1+e^{-z} \;-1}{(1+e^{-z})}\right)$$

$$= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

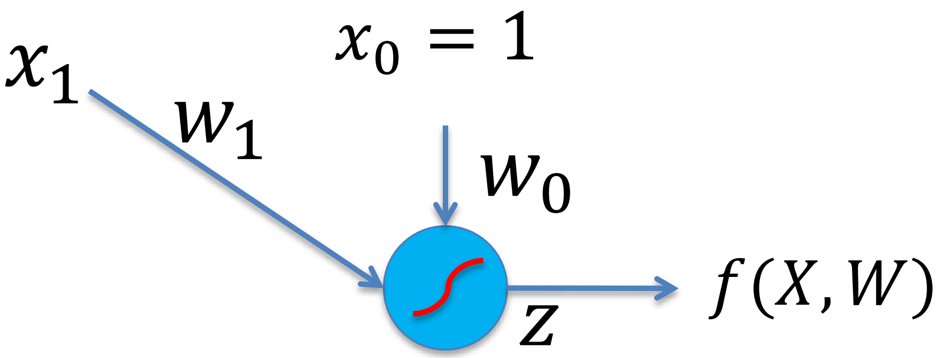
$$= Y(1 - Y)$$

- $\frac{\partial E}{\partial w_i} = (Y - T) Y (1 - Y) x_i$

"Training" a Sigmoid Neuron

- Consider single sample and single dimension:

	x_0	x_1		T
X^1	1	2.3		2.7



- Sigmoid Activation Function:

$$f(X,W) = 1/(1 + e^{-z})$$

Training a Sigmoid Neuron

- Initial $w_0 = 5, w_1 = 7$ and $\eta = 0.1$
- Feed X^1:
  - $z = 5 * 1 + 7 * 2.3 = \mathbf{21.1}$
  - $Y = \frac{1}{1+e^{-z}} = 0.999999999313901$
  - $E = \frac{1}{2}(Y - T)^2 = 1.445000001166368$

	x_0	x_1
X^1	1	2.3

w_0	w_1
5	7

Y	T	E
0.99999999931390	2.7	1.44500000116636

Training a Sigmoid Neuron

	x_0	x_1
X^1	1	2.3

w_0	w_1
5	7

Y	T	E
0.99999999931390	2.7	**1.44500000116636**

- $f'(z) = Y(1 - Y) = 7E^{-10}$
- $w_i^{next} = w_i - \eta(Y - T)f'(z)x_i$

  $= w_i + 1.16E^{-9}x_i$
- $w_0^{next} = 5.00000000012$
- $w_1^{next} = 7.00000000027$

	x_0	x_1
X^1	1	2.3

w_0	w_1
5.00000000012	**7.00000000027**

Y	T	E
0.99999999931390	2.7	**1.44500000116636**

Training a Sigmoid Neuron

	x_0	x_1
X^1	1	2.3

w_0	w_1
5	7

Y	T	E
0.999999999313901	2.7	1.445000001166368

After 1 iteration:

	x_0	x_1
X^1	1	2.3

w_0	w_1
5.000000000012	7.00000000027

Y	T	E
0.999999999313901	2.7	1.445000001166368

After 1000 iterations:

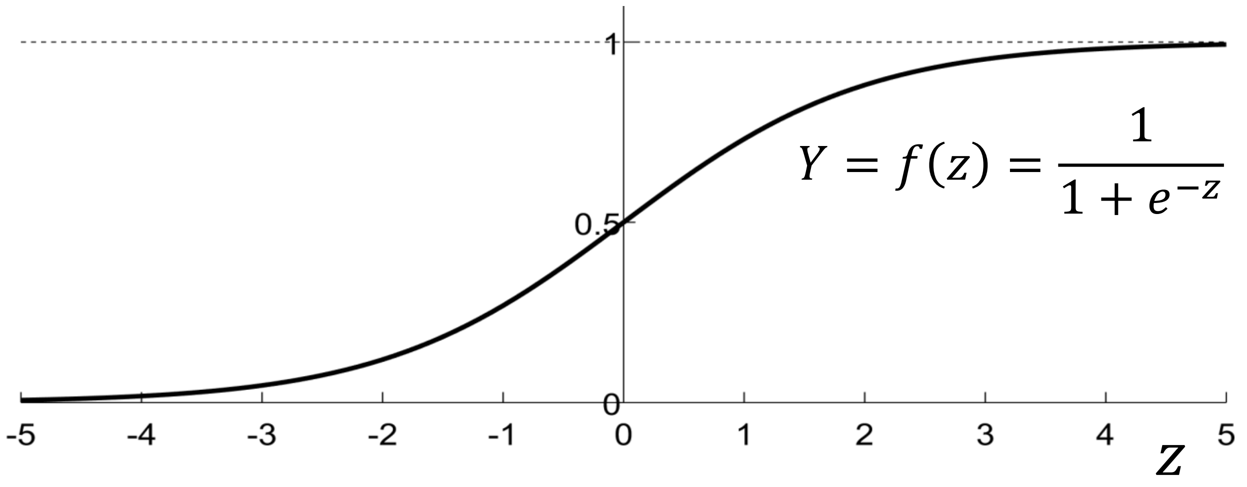
	x_0	x_1
X^1	1	2.3

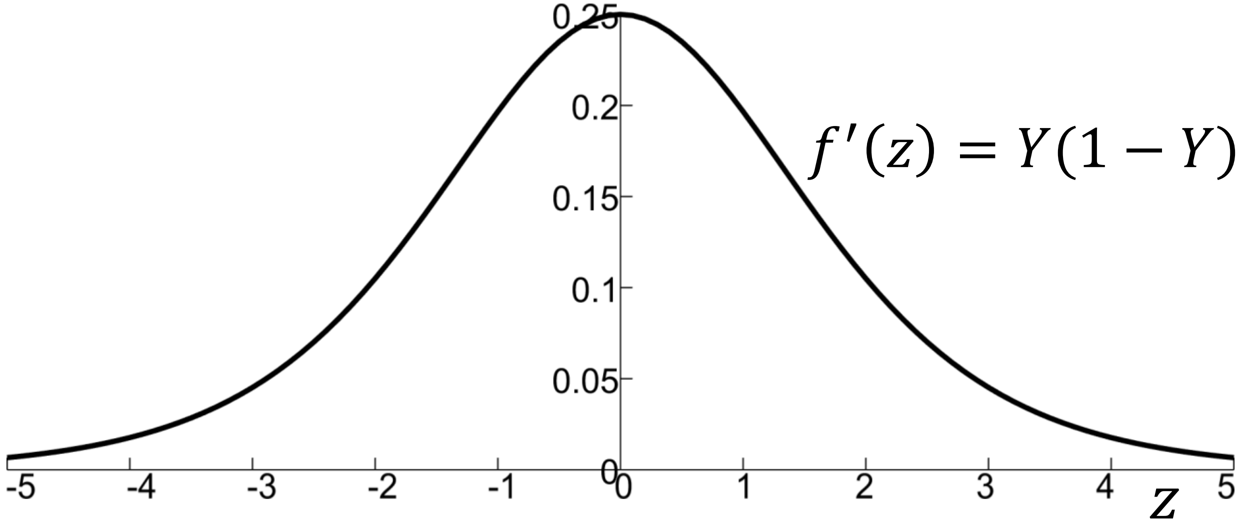
w_0	w_1
5.00000011663	7.00000026826

Y	T	E
0.999999999313902	2.7	1.445000001166367

Sigmoid Function

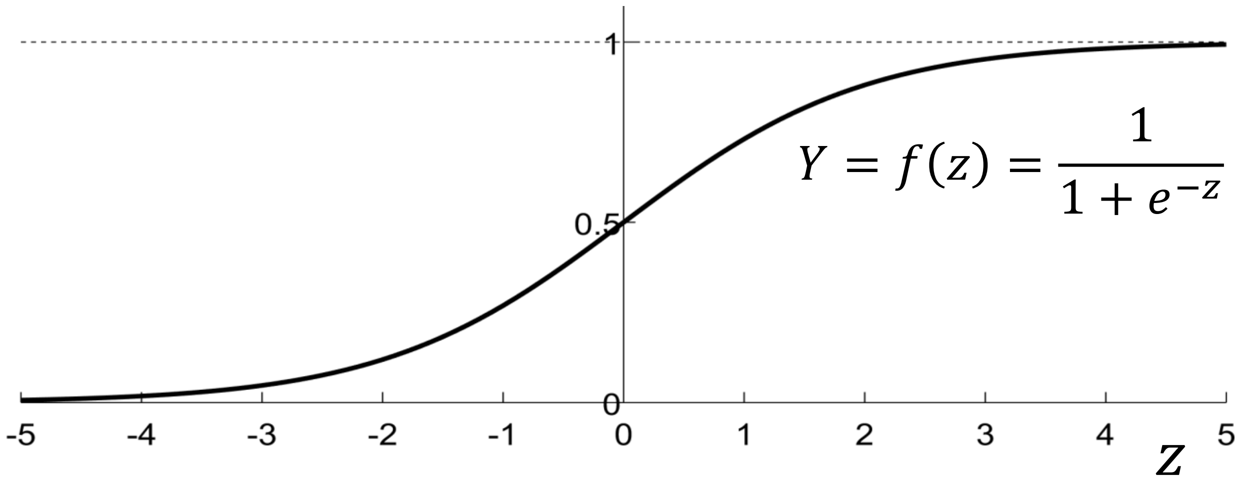
- $z = \sum_{i=0}^{D} w_i x_i$
- $Y = f(z) = \frac{1}{1+e^{-z}}$
- $f'(z) = Y(1 - Y)$
- "Squashing" function
- Nonlinear
- Saturating



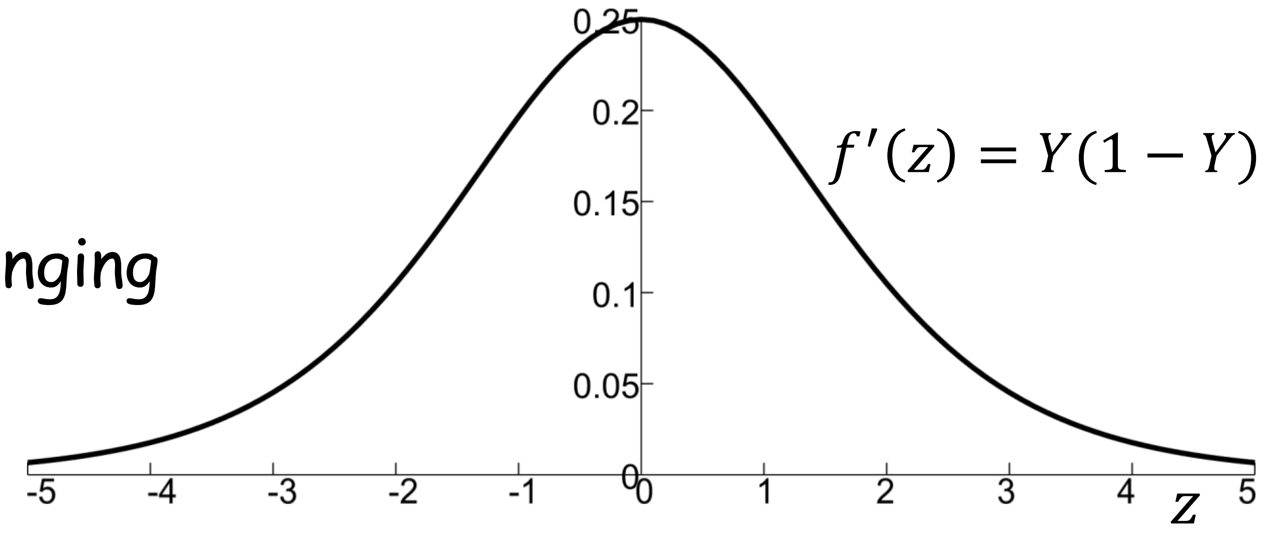


Sigmoid Function

- $z = \sum_{i=0}^{D} w_i x_i$
- $Y = f(z) = \frac{1}{1+e^{-z}}$
- $f'(z) = Y(1 - Y)$
- "Squashing" function
- Nonlinear
- Saturating



- Nearly-linear around 0
- "Vanishing Gradient": Slow-changing when far away from 0
- $\frac{\partial E}{\partial w_i} = (Y - T) f'(z) x_i$

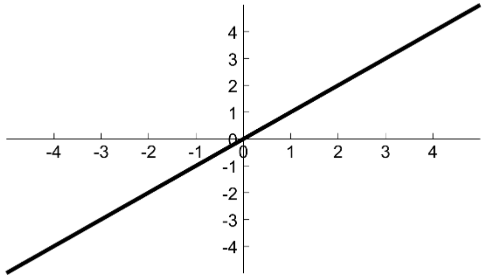


Activation Functions

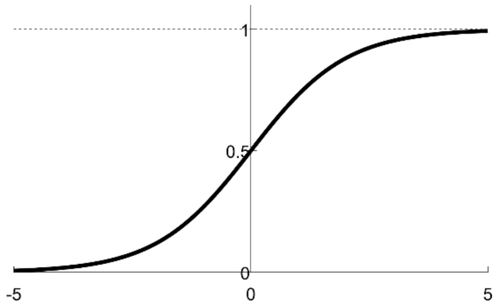
- Continuous
- Differentiable
- Monotonic

Activation Functions

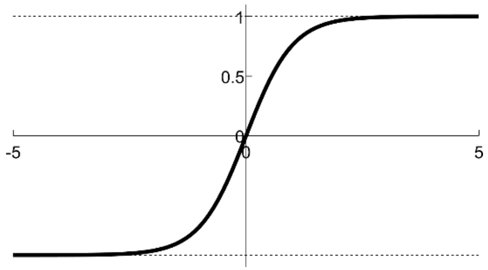
Linear:



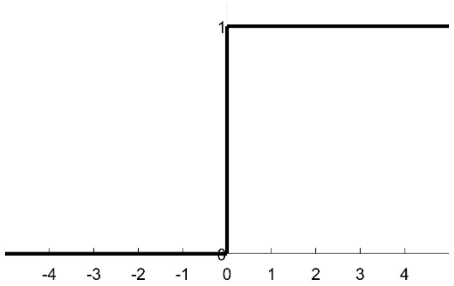
Sigmoid aka Logistic:



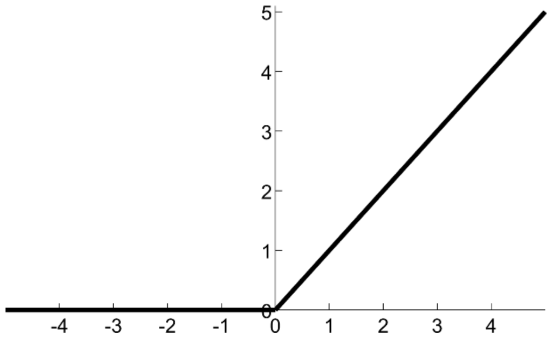
tanh:



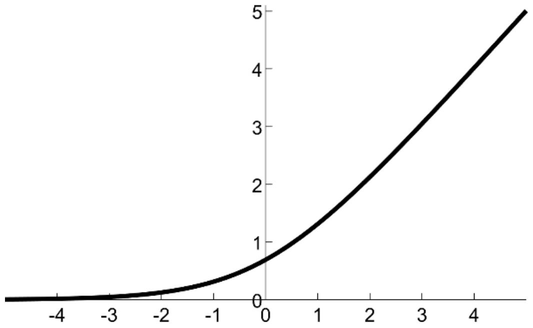
Step:



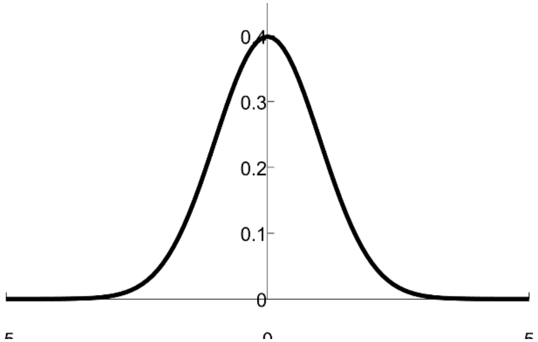
ReLu
(Rectified Linear Activation Unit):



Smooth ReLu

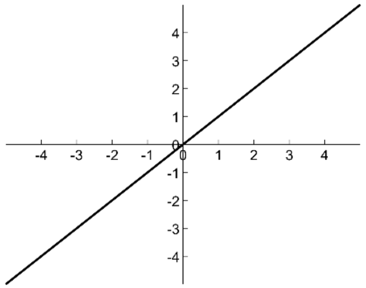


Gaussian:



Activation Function

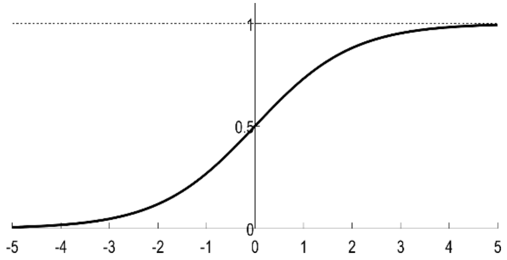
Linear



- Regression Problems
  - Continuous Target values

	x_0	x_1	x_2	. . .	x_D		T
X^1	1	2.3	5.6	. . .	7.9		**2.7**
X^2	1	6.6	0.4	. . .	4.3		**3.2**
	1			. . .			**. . .**
	1			. . .			
	1			. . .			
	1			. . .			
X^N	1			. . .			T^N

Sigmoid

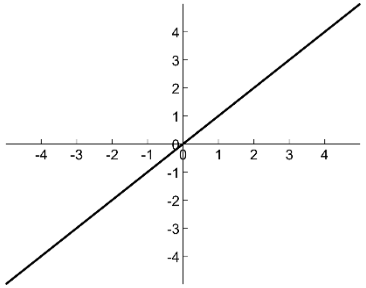


- Classification Problems
  - Binary Target values

	x_0	x_1	x_2	. . .	x_D		T
X^1	1	2.3	5.6	. . .	7.9		**0**
X^2	1	6.6	0.4	. . .	4.3		**1**
	1			. . .			**. . .**
	1			. . .			**1**
	1			. . .			**1**
	1			. . .			**0**
X^N	1			. . .			T^N

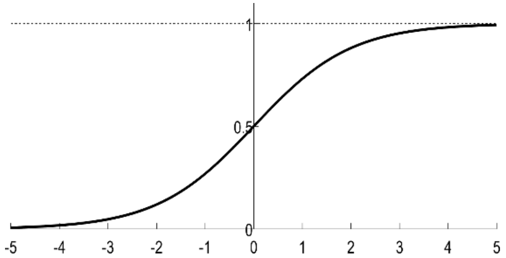
Activation Function

Linear



- Able to produce any numerical output
- Use for "regression" problems
  - Target Values: Continuous

Sigmoid



- Output: between 0 and 1.
- Good for "classification" problems
- Target values:
  - 0 representing one class
  - 1 representing the other
- Interpret neuron output Y
  - Y< 0.5 represents one class
  - Y>= 0.5 represents the other

What if we had more than one sample?

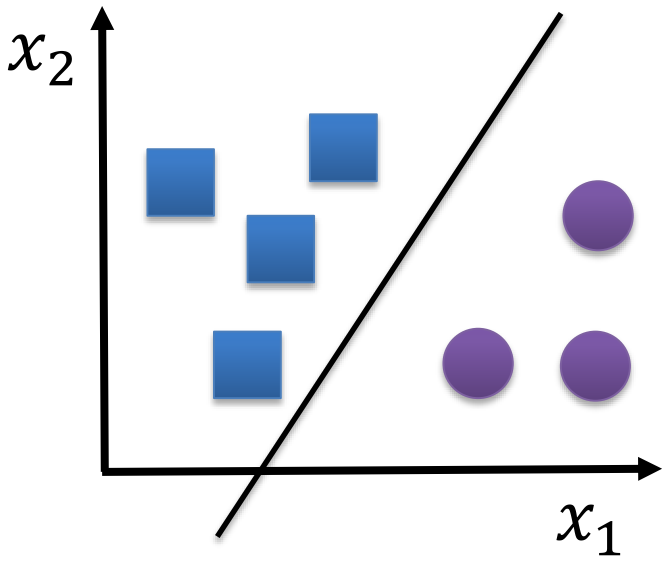
	x_0	x_1	x_2	...	x_D		T
X^1	1	2.3	5.6	...	7.9		2.7
X^2	1	6.6	0.4	...	4.3		3.2
	1			...			...
	1			...			
	1			...			
	1			...			
X^N	1			...			T^N

$$E = \frac{1}{N}\sum_{i=0}^{N}\left(Y^i - T^i\right)^2$$

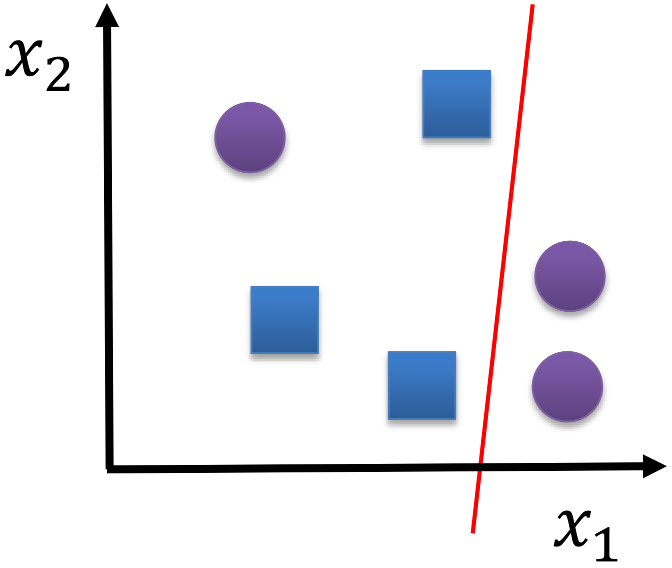
- Batch
  - Present all the patterns
  - "epoch": single presentation of all patterns.
- Online/Iterative
  - Sequentially, one pattern at a time
- Stochastic
  - Choose input pattern/sample/instance randomly

Single Neuron

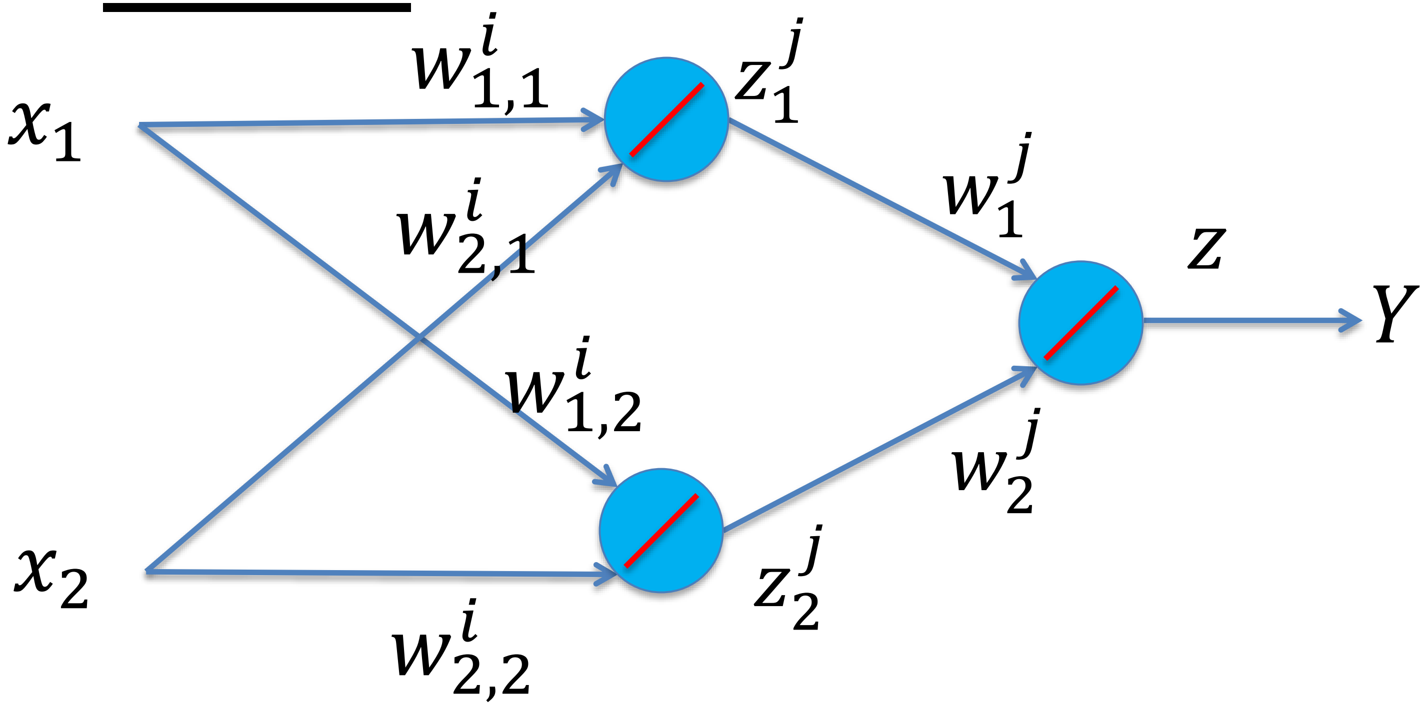
- A single neuron (linear or sigmoid) can only learn linearly-separable classification.



$$z = \sum_{i=0}^{D} w_i x_i$$



Network of Linear Neurons



Network of Linear Neurons

$$Y = f(z) = z$$

$$= w_1^j f(z_1^j) + w_2^j f(z_2^j)$$

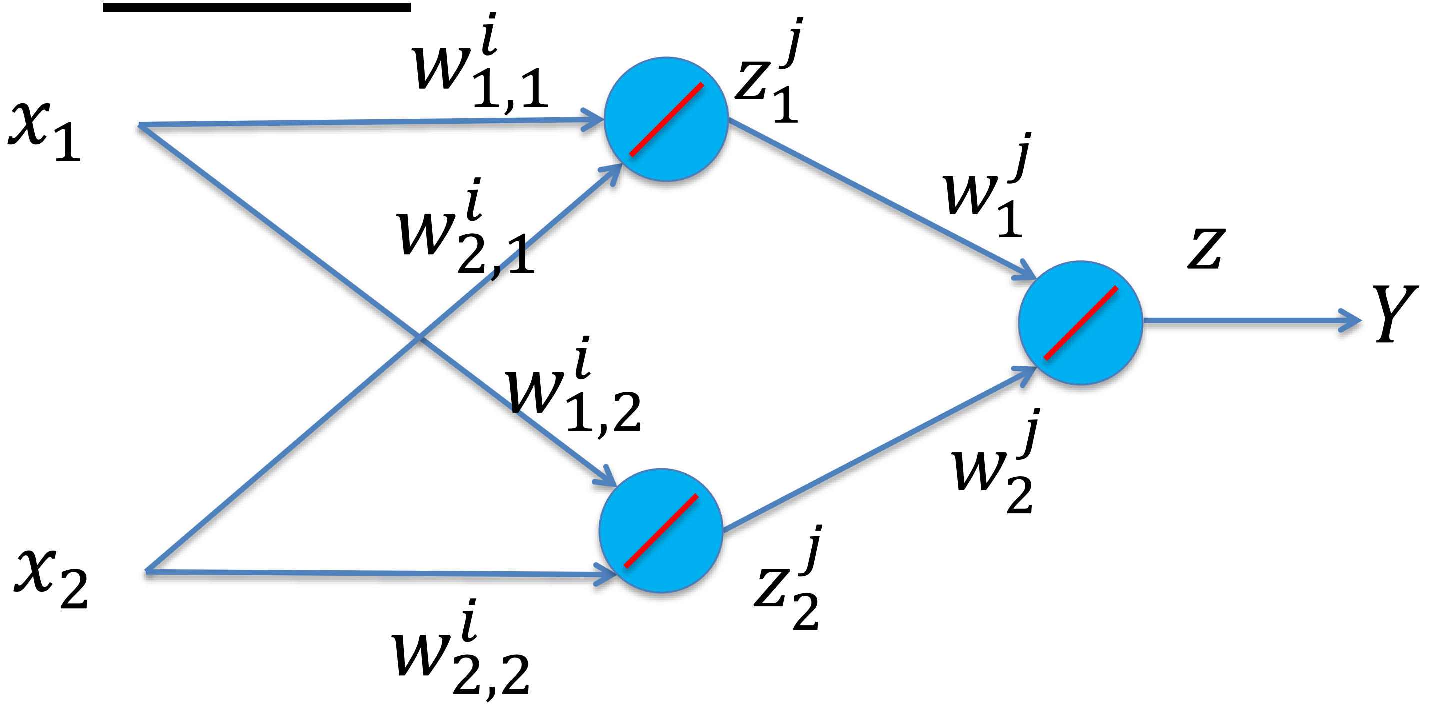
$$= w_1^j z_1^j + w_2^j z_2^j$$

$$= w_1^j (w_{1,1}^i x_1 + w_{2,1}^i x_2)$$

$$+ w_2^j (w_{1,2}^i x_1 + w_{2,2}^i x_2)$$

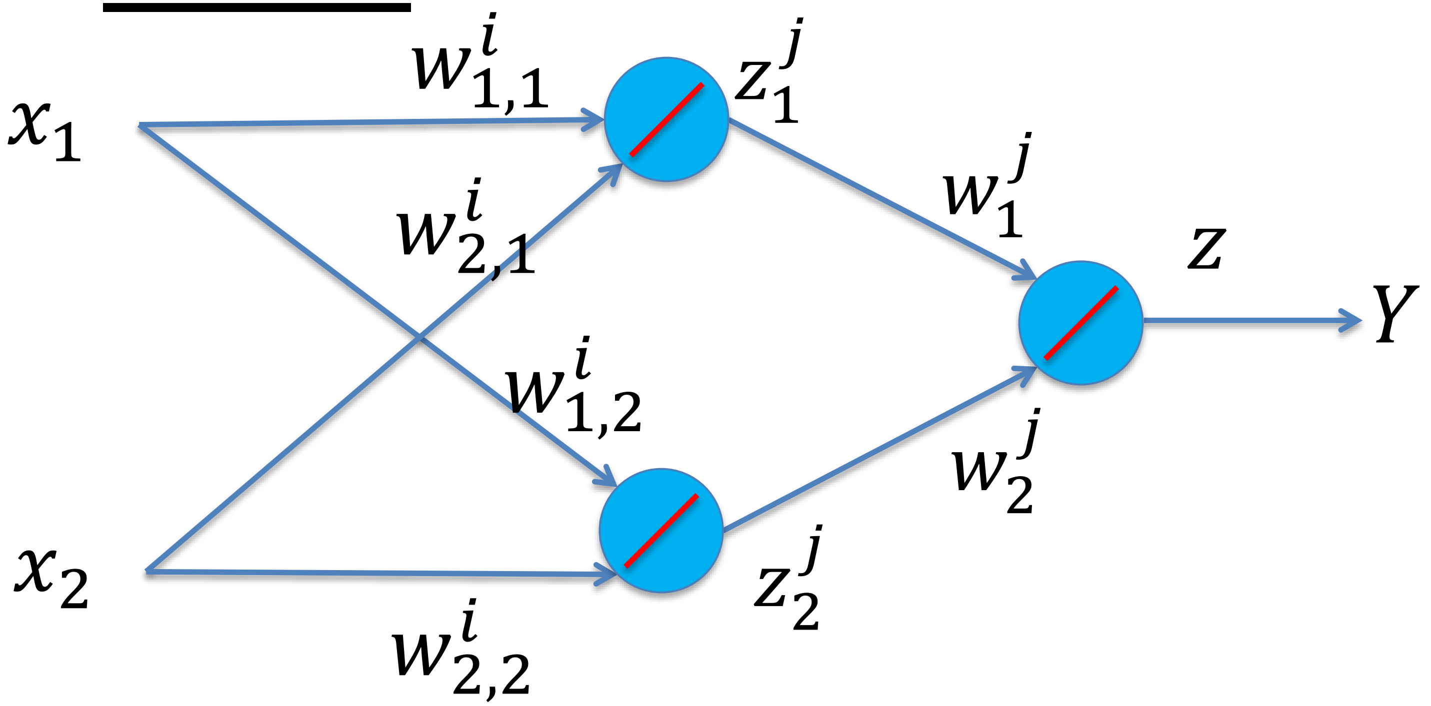
$$= (w_1^j w_{1,1}^i + w_2^j w_{1,2}^i) \mathrm{x}_1$$

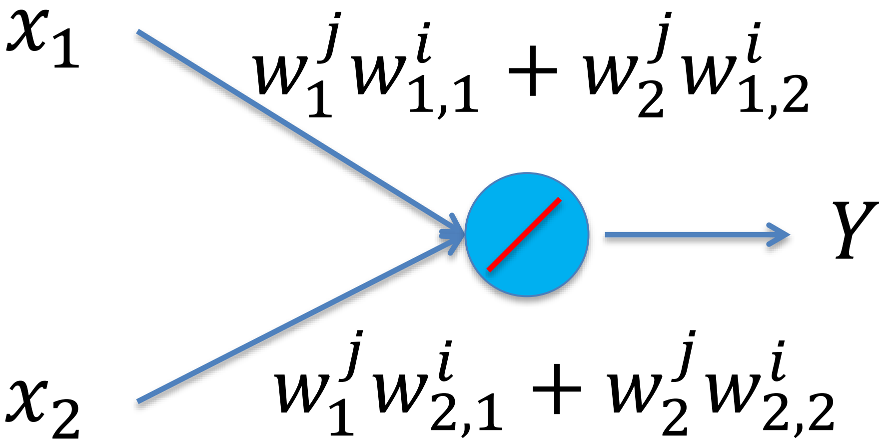
$$+ (w_1^j w_{2,1}^i + w_2^j w_{2,2}^i) \mathrm{x}_2$$



Network of <u>Linear</u> Neurons

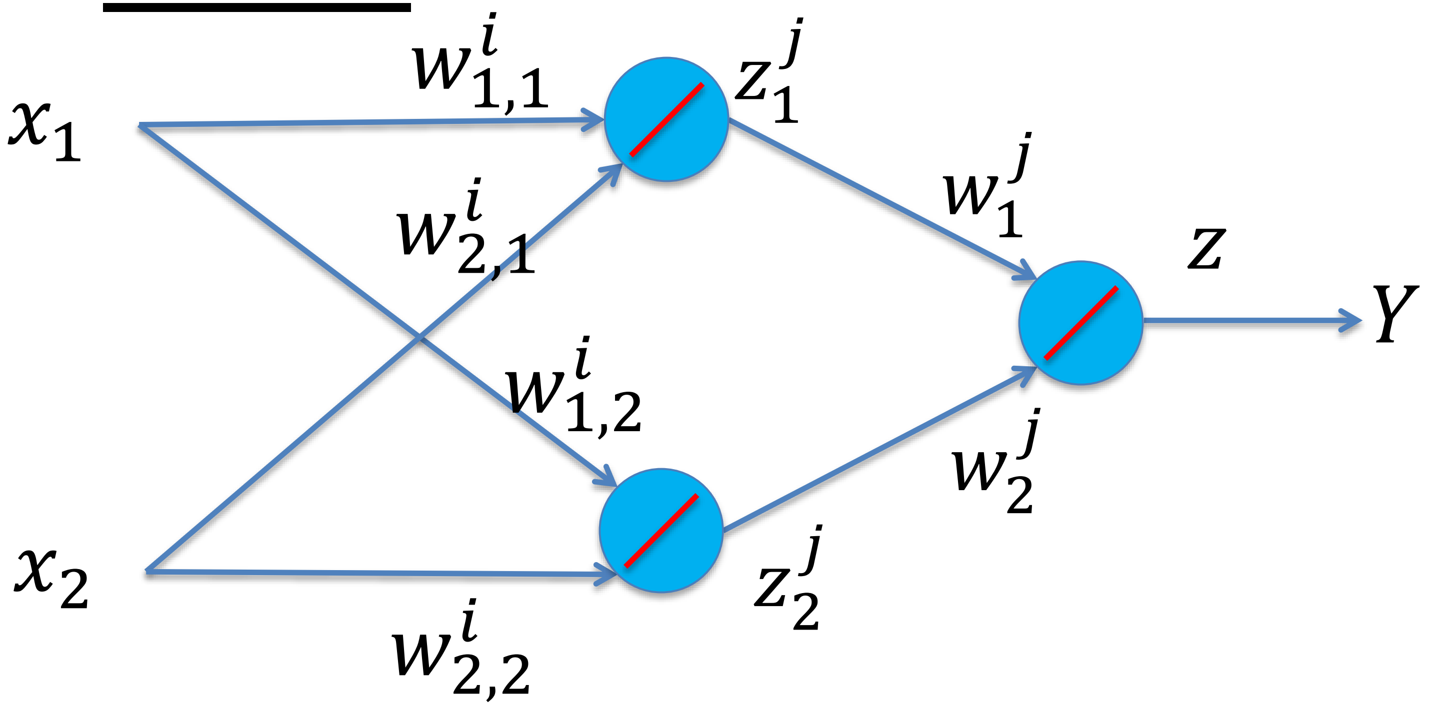
$$
\begin{aligned}
Y = f(z) &= z \\
&= w_1^j f(z_1^j) + w_2^j f(z_2^j) \\
&= w_1^j z_1^j + w_2^j z_2^j \\
&= w_1^j \left(w_{1,1}^i x_1 + w_{2,1}^i x_2 \right) \\
&\quad + w_2^j \left(w_{1,2}^i x_1 + w_{2,2}^i x_2 \right) \\
&= \left(w_1^j w_{1,1}^i + w_2^j w_{1,2}^i \right) \mathrm{x}_1 \\
&\quad + \left(w_1^j w_{2,1}^i + w_2^j w_{2,2}^i \right) \mathrm{x}_2
\end{aligned}
$$

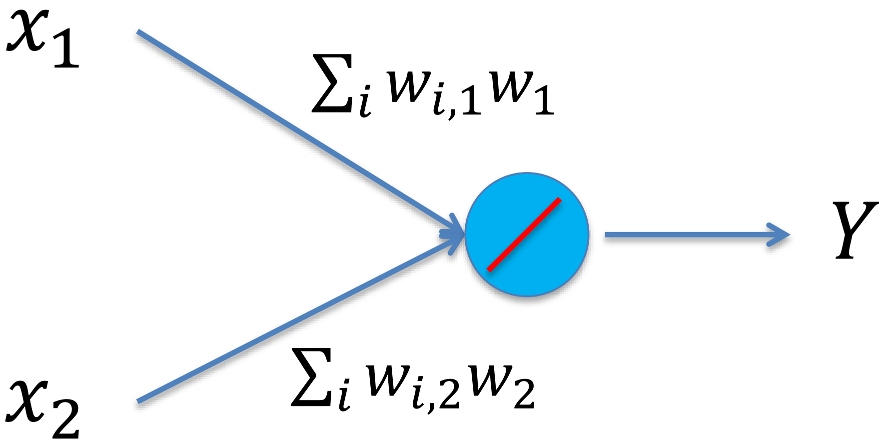




Network of Linear Neurons

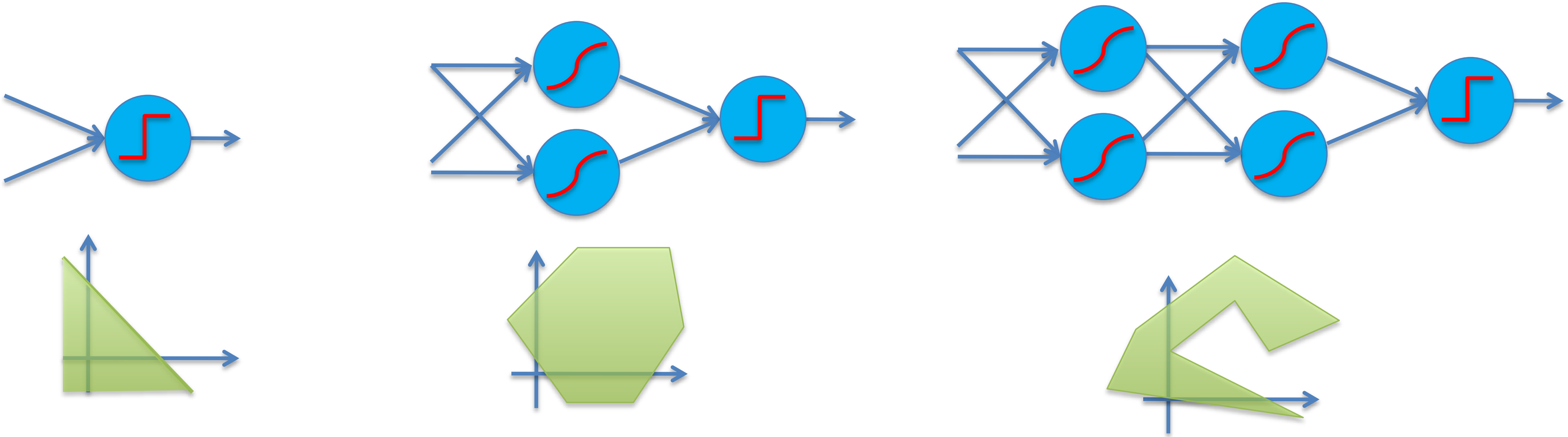
$$Y = f(z) = z$$
$$= \sum_j w_j z_j$$
$$= \sum_j w_j \left(\sum_i w_{i,j} x_i\right)$$
$$= \sum_i \left(\sum_i w_{i,j} w_j\right) x_i$$



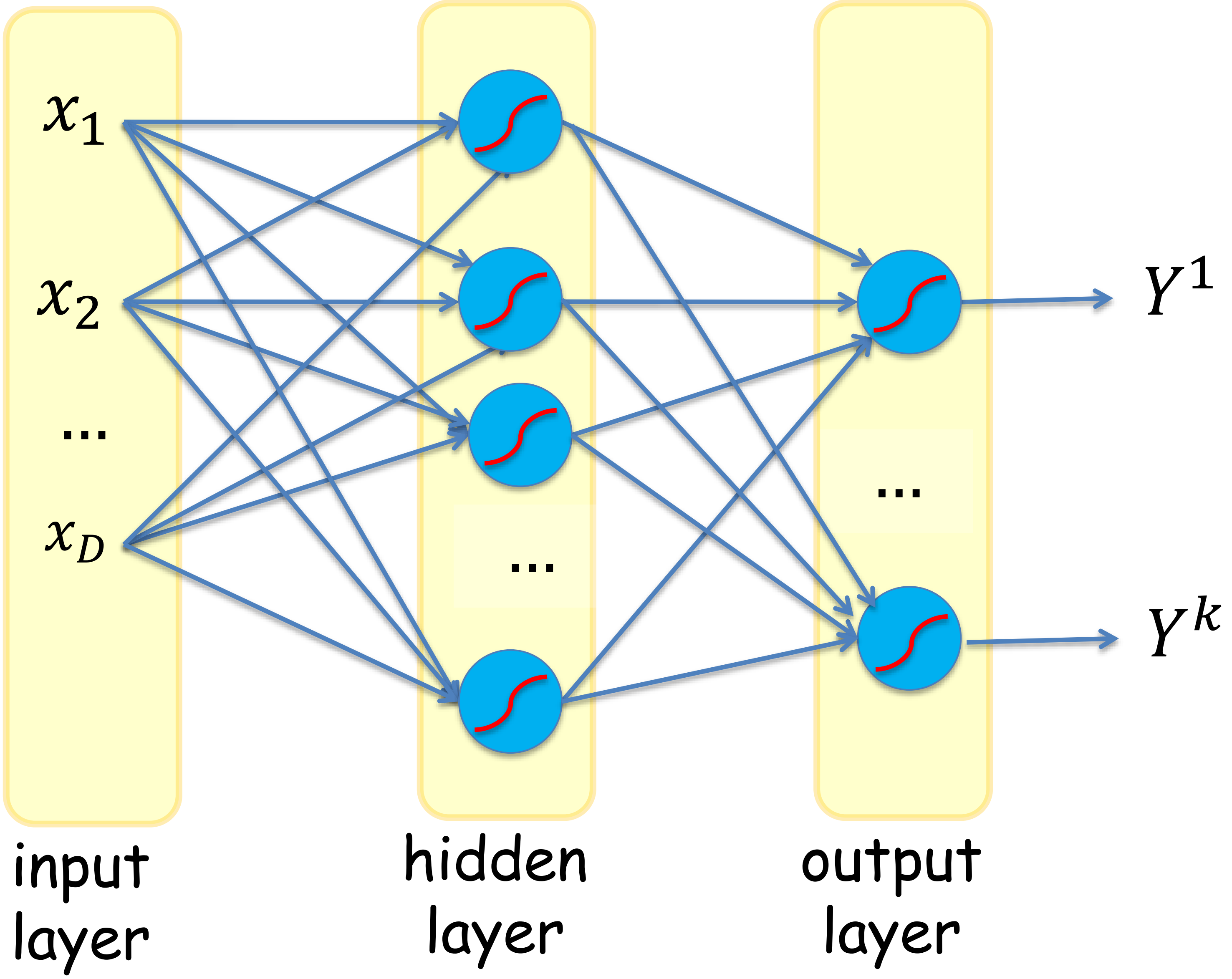


Network of Non-Linear (e.g. sigmoid) Neurons

- Universal approximation theorem
  - Given sufficient number of neurons, a feed-forward network of non-linear neurons can approximate any continuous function.



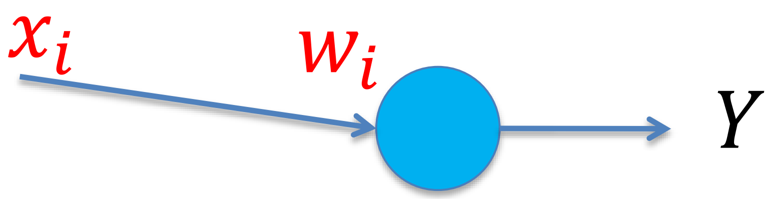
3-layer Feed-forward Neural Network

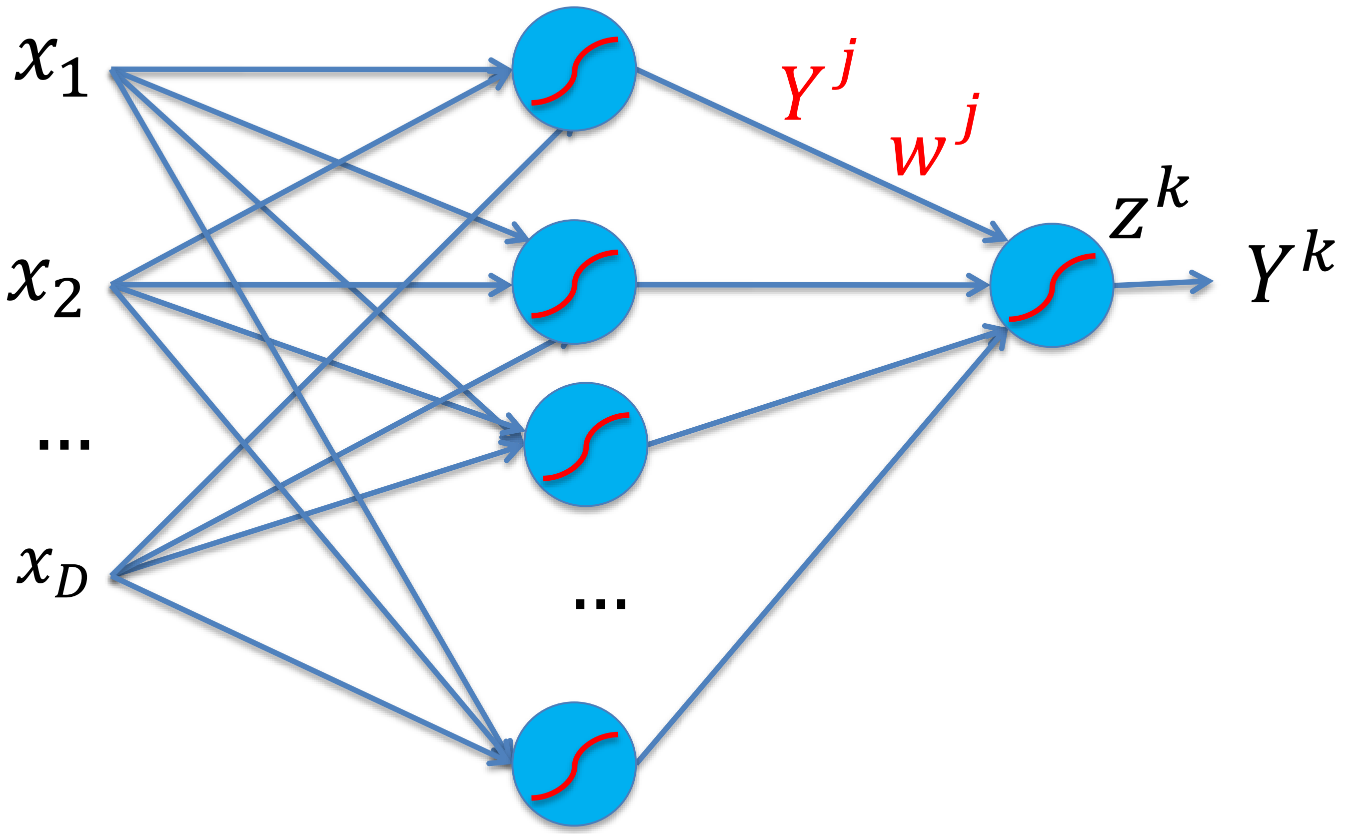


Training a Feed-Forward Network

Single Neuron:

$$\frac{\partial E}{\partial w_i} = (Y - T)f'(z)x_i$$





- Output Neuron:

$$\frac{\partial E}{\partial w^j} = \left(Y^k - T^k\right) f'\left(z^k\right)Y^j$$

Error Backpropagation

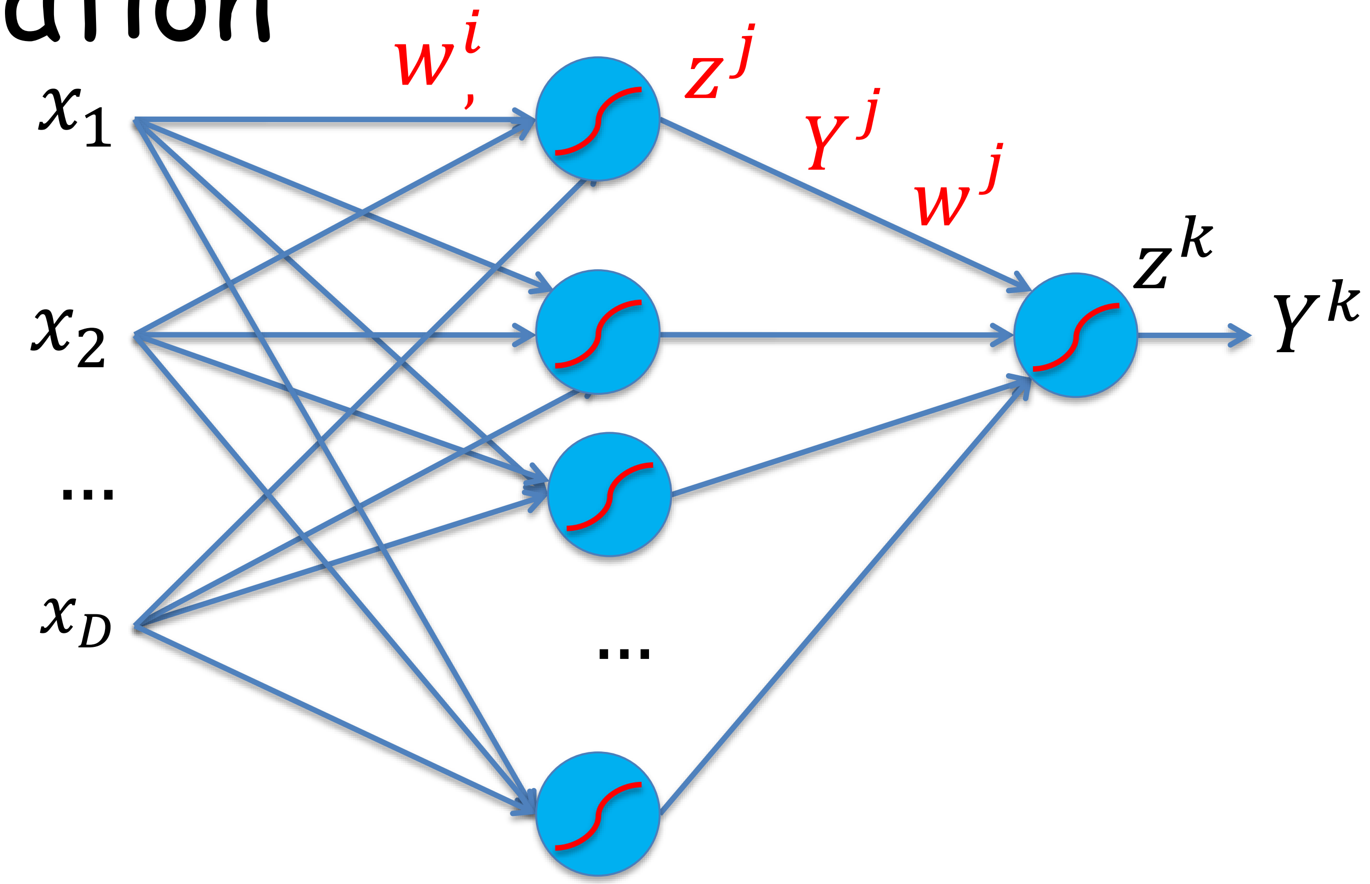
- Output Neuron:

$$\frac{\partial E}{\partial w^j} = \left(Y^k - T^k\right) f'\left(z^k\right) Y^j$$

- Hidden Neuron

- $$\frac{\partial E}{\partial w^i} = \left(Y^k - T^k\right) f'\left(z^k\right) \frac{\partial z^k}{\partial w^i}$$

- $$\frac{\partial z^k}{\partial w^i} = \frac{\partial z^k}{\partial Y^j} \frac{\partial Y^j}{\partial z^j} \frac{\partial z^j}{\partial w^i} = w^j f'\left(z^j\right) x_i$$



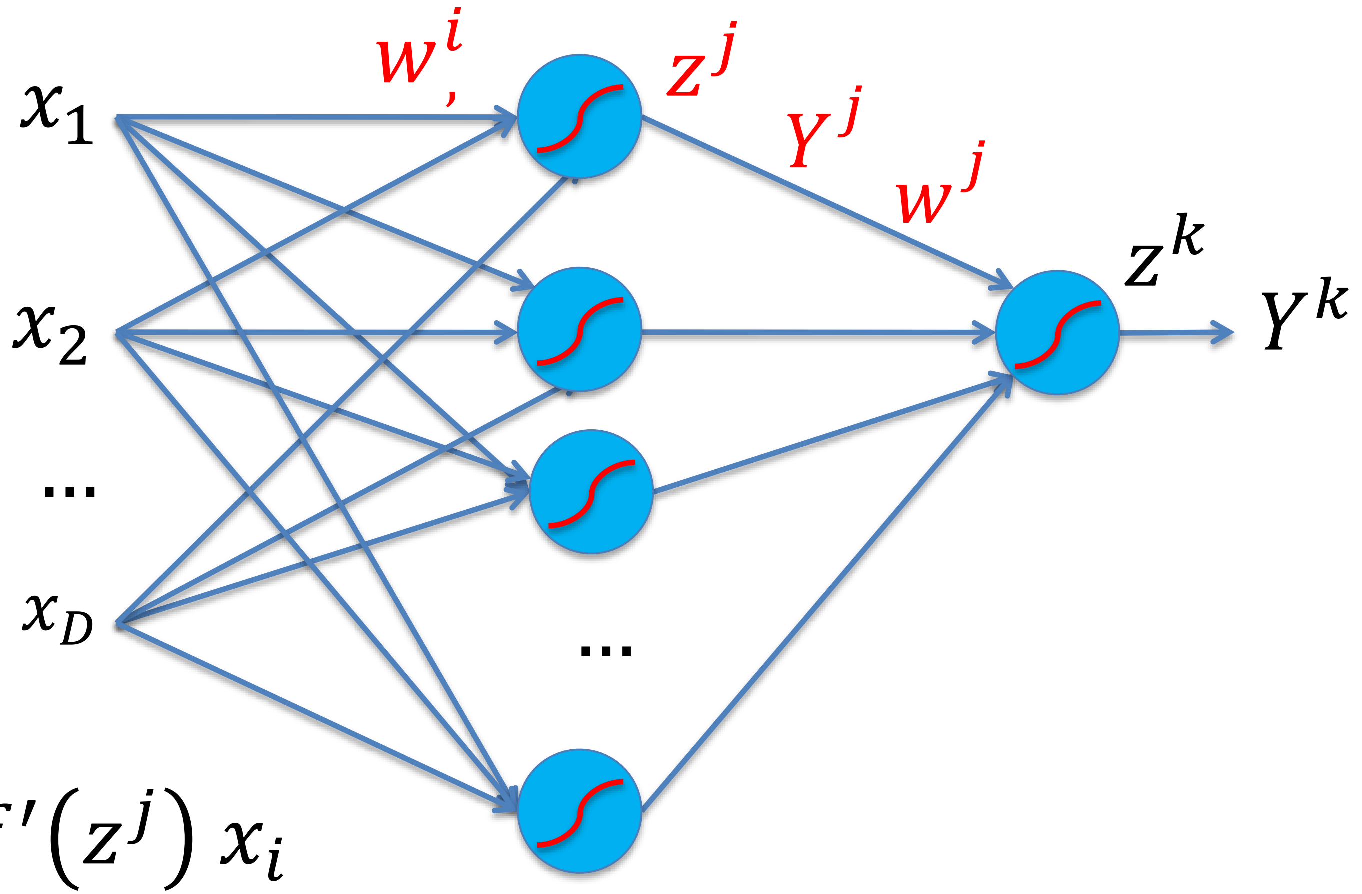
Training a Feed-Forward Network

- Output Neuron:

$$\frac{\partial E}{\partial w^j} = \left(Y^k - T^k\right) f'\left(z^k\right) Y^j$$
$$= \delta^k Y^j$$

- Hidden Neurons:

$$\frac{\partial E}{\partial w^i} = \left(Y^k - T^k\right) f'\left(z^k\right) w^j f'\left(z^j\right) x_i$$
$$= \delta^k w^j f'\left(z^j\right) x_i = \delta^j x_i$$



Training a Feed-Forward Network

- Output Neuron:

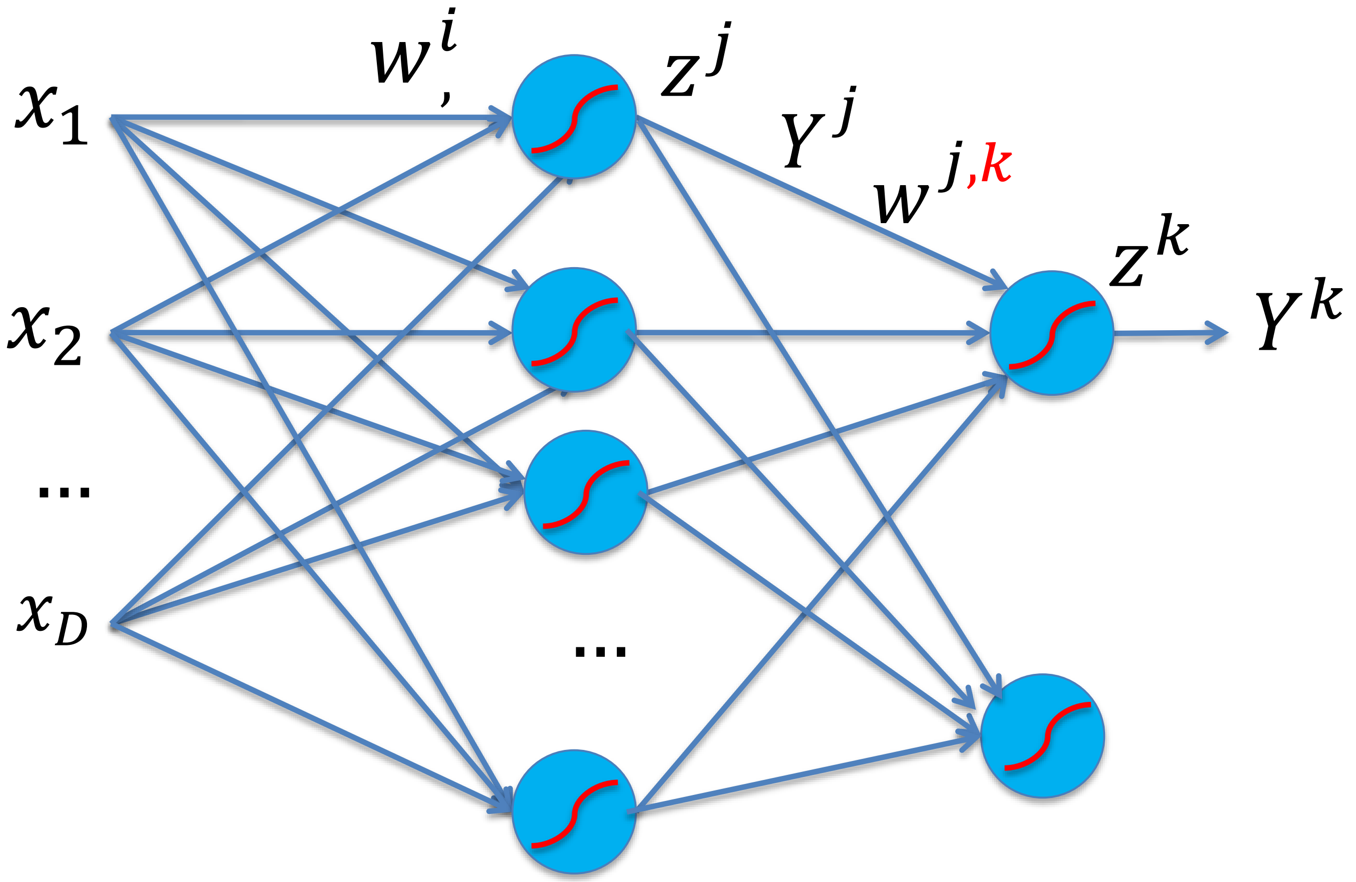
$$\delta^k = (Y^k - T^k) f'(z^k)$$

$$\frac{\partial E}{\partial w^j} = \delta^k Y^j$$

- Hidden Neurons:

$$\delta^j = \left(\sum_k \delta^k w^{j,k}\right) f'(z^j)$$

$$\frac{\partial E}{\partial w^i} = \delta^j x_i$$



Generalized Delta Rule

- Output Neuron:

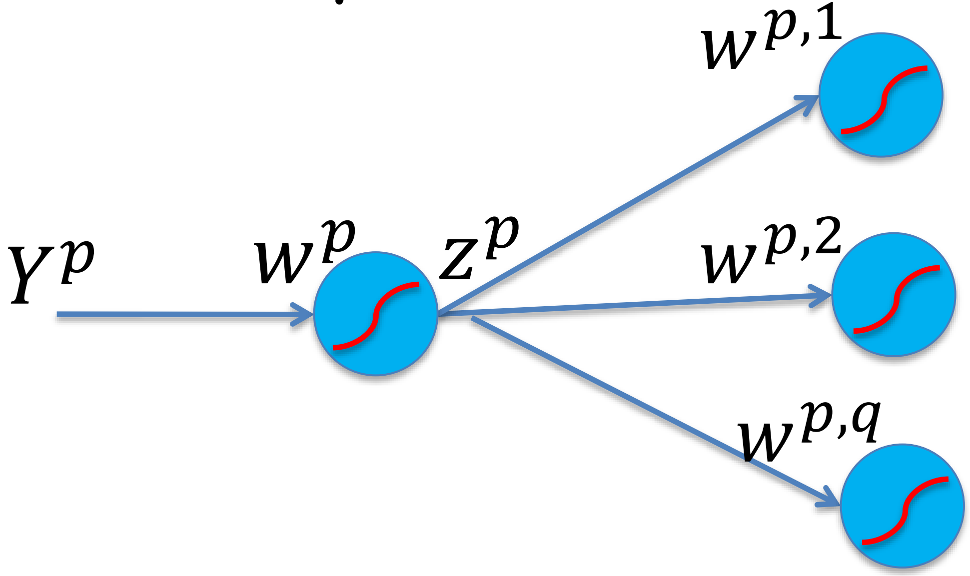
$$\delta^k = (Y^k - T^k) f'(z^k)$$

$$\frac{\partial E}{\partial w^j} = \delta^k Y^j$$

- Non-output Neurons at Layer p (Let next layer be q)

$$\delta^p = (\sum_q \delta^q w^{p,q}) f'(z^p)$$

$$\frac{\partial E}{\partial w^p} = \delta^p Y_p$$

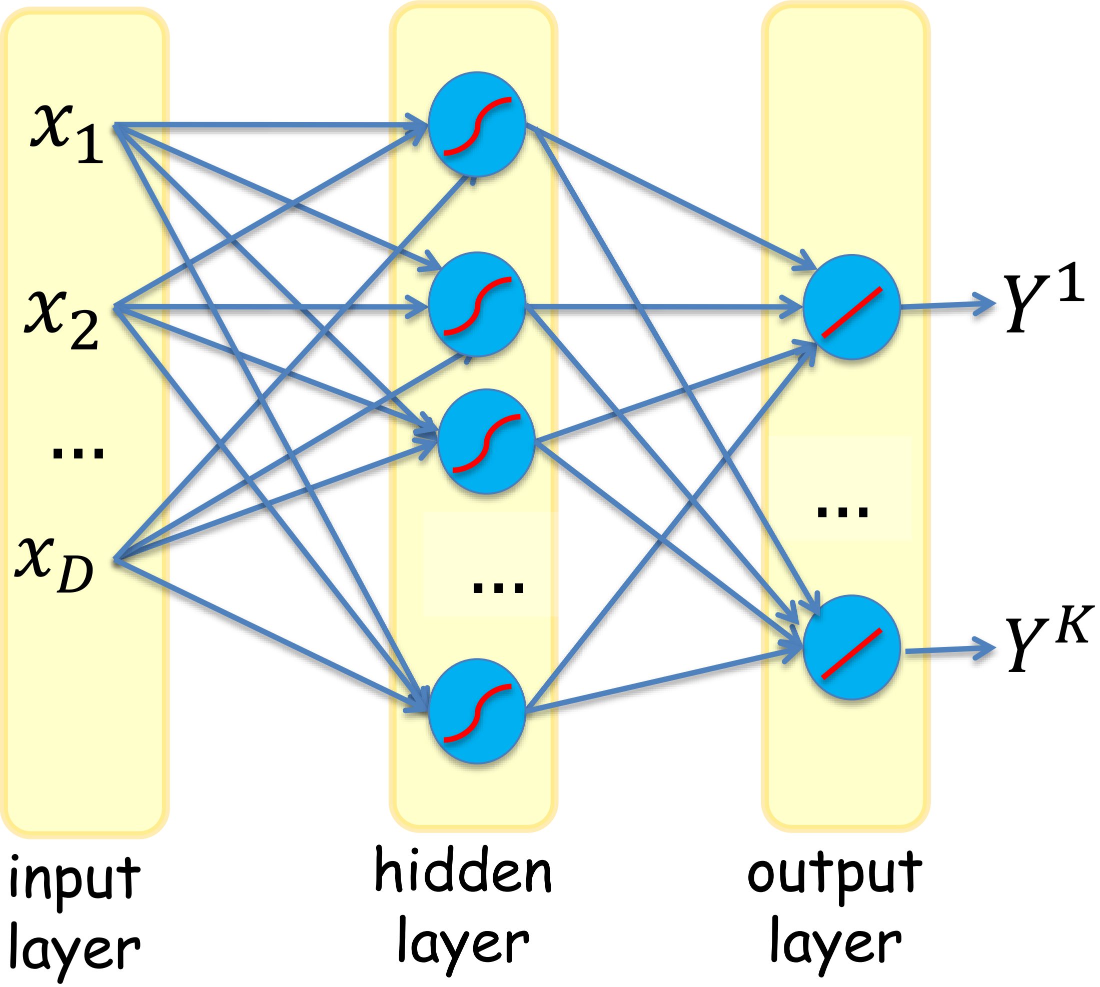


- For input layer: Y^p becomes x_i

Neural Networks in Practice

	x_0	x_1	x_2	...	x_D
X^1	1	2.3	5.6	...	7.9
X^2	1	6.6	0.4	...	4.3
	1			...	
	1			...	
	1			...	
	1			...	
X^N	1	x_1^N	x_2^N	...	x_D^N

T_1	...	T_K
2.7	...	6.7
3.2	...	5.9
...	...	...
	...	
	...	
	...	
T_1^N	...	T_K^N



Problems with Neural Networks

- Interpretation of Hidden Layers

- Slow Training, Local Minima
  - Adaptive learning rate, momentum, random restarts

- Overfitting

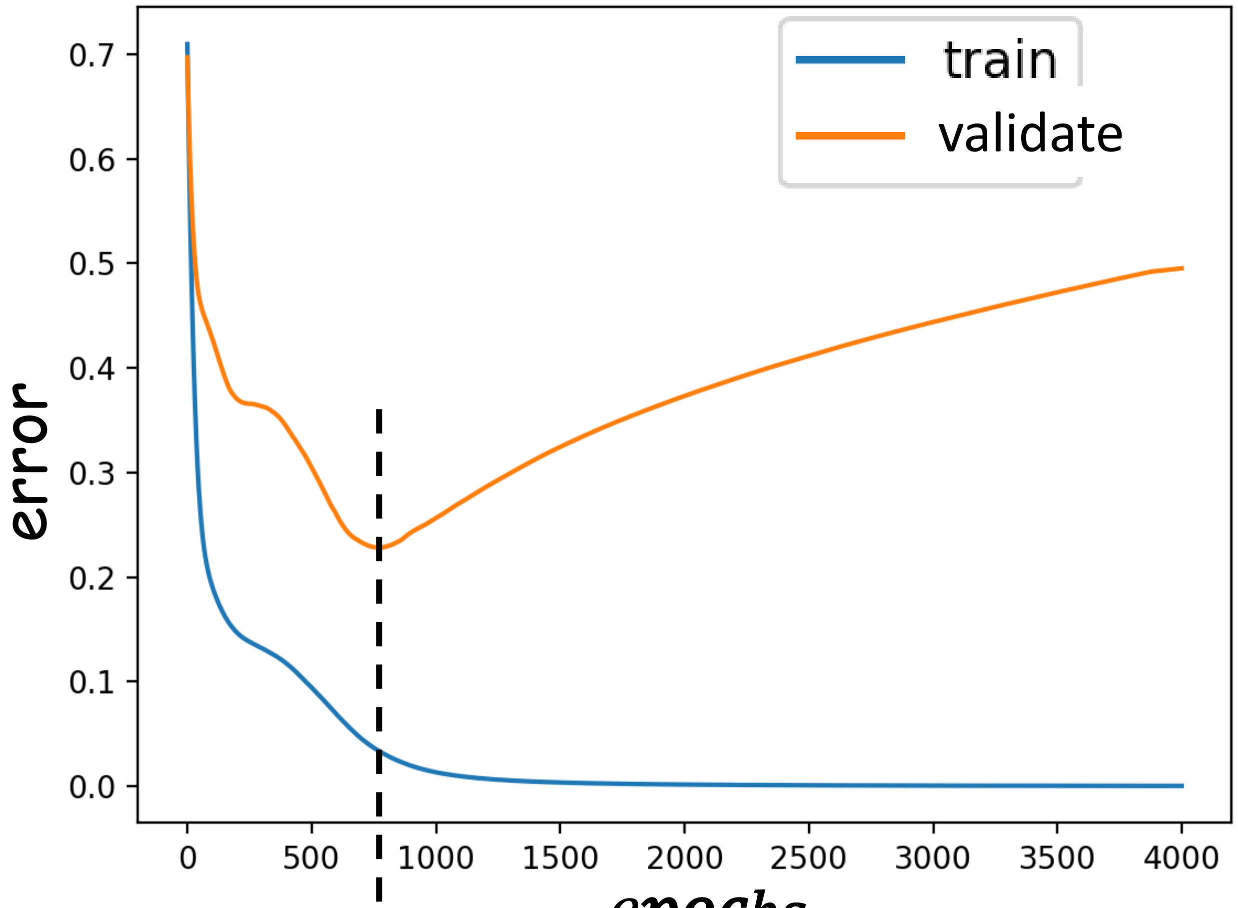
Overfitting

- Use a validation set to detect/prevent overfitting

training

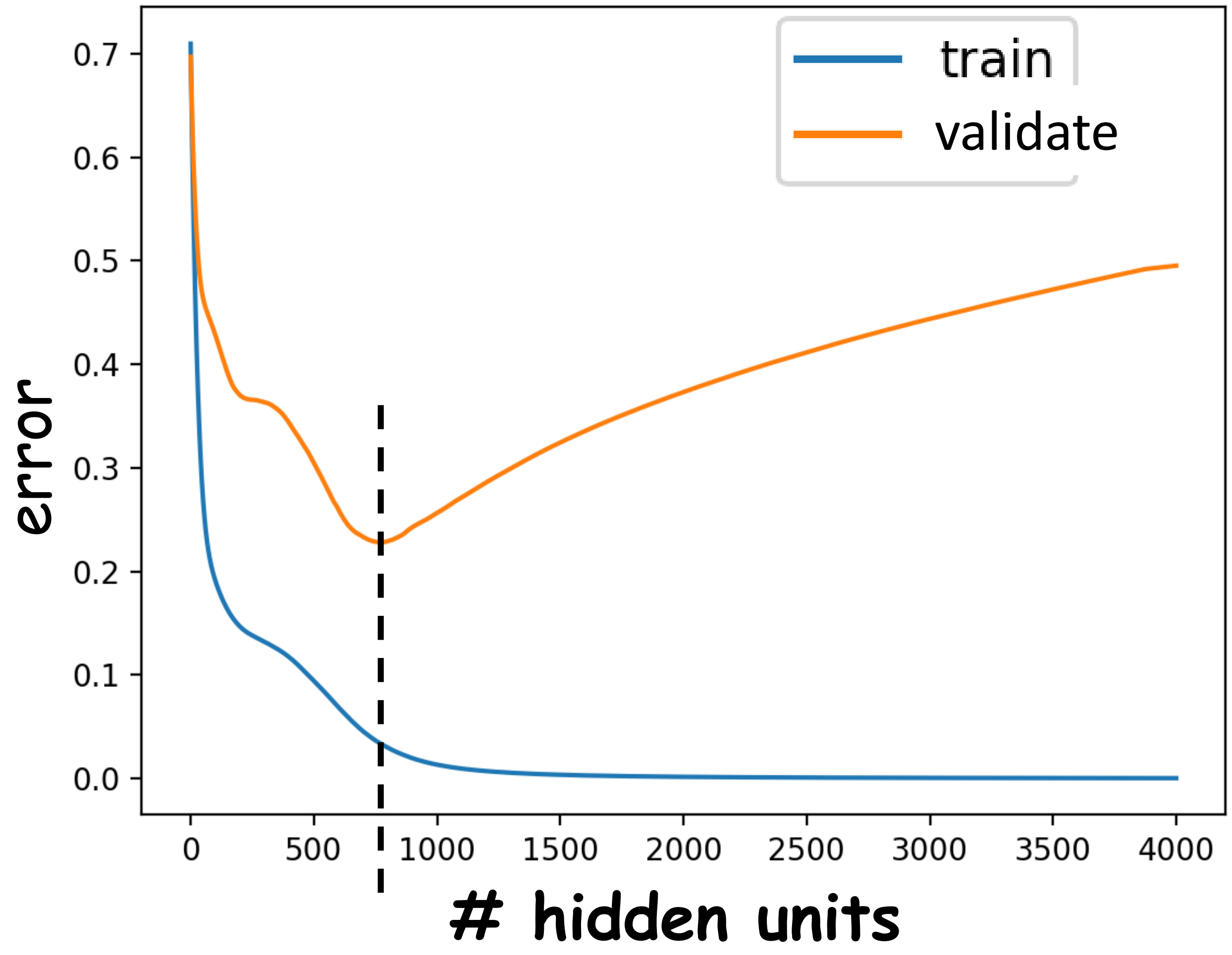
	x_0	x_1	x_2	...	x_D	T
X^1	1	2.3	5.6	...	7.9	2.7
X^2	1	6.6	0.4	...	4.3	3.2
	1			...		...
	1			...		
	1			...		
	1			...		
X^N	1			...		T^N

validation

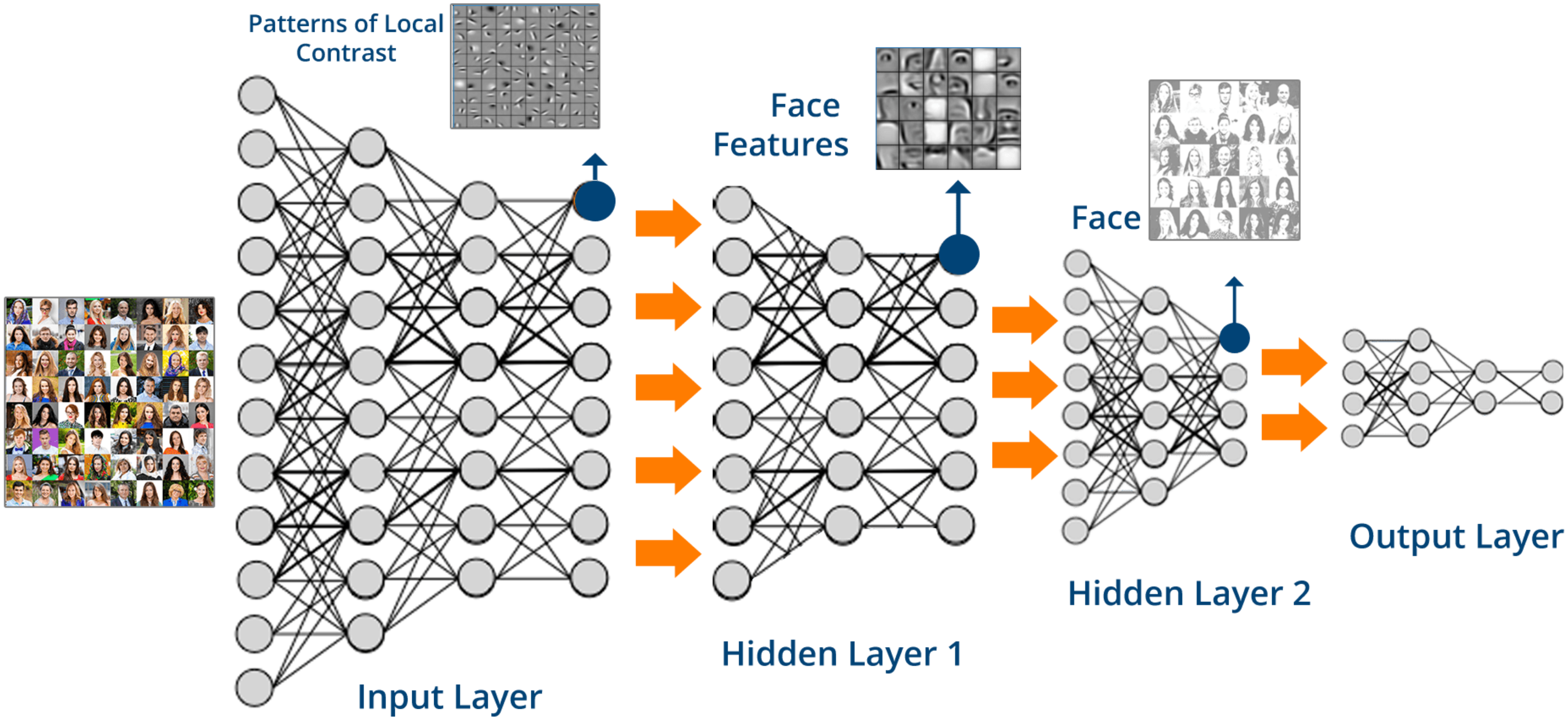


Overfitting

- Use a network only as powerful as needed, and not more



Deep Learning

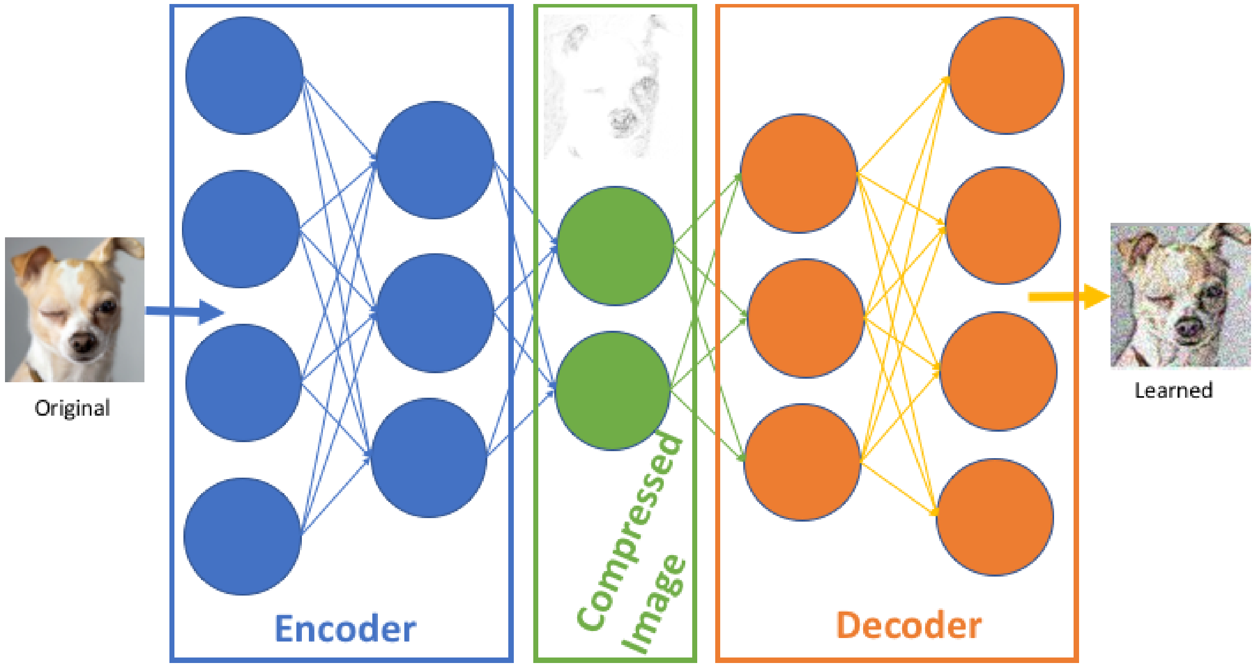


https://i1.wp.com/www.michaelchimenti.com/wp-content/uploads/2017/11/Deep-Neural-Network-What-is-Deep-Learning-Edureka.png

Auto-Encoder Network

	x_1	x_2	...	x_D
X^1	2.3	5.6	...	7.9
X^2	6.6	0.4	...	4.3
			...	
			...	
			...	
			...	
X^N			...	

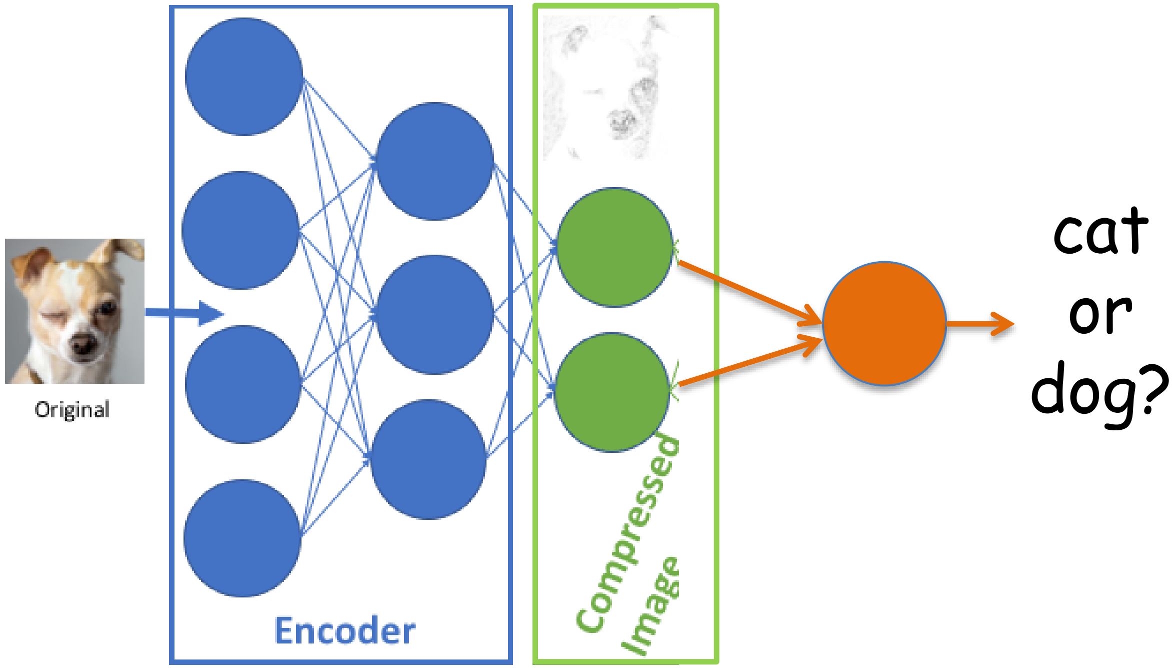
T_1	T_2	...	T_D
2.3	5.6	...	7.9
6.6	0.4	...	4.3
...			
...			
...			
...			
T^N			

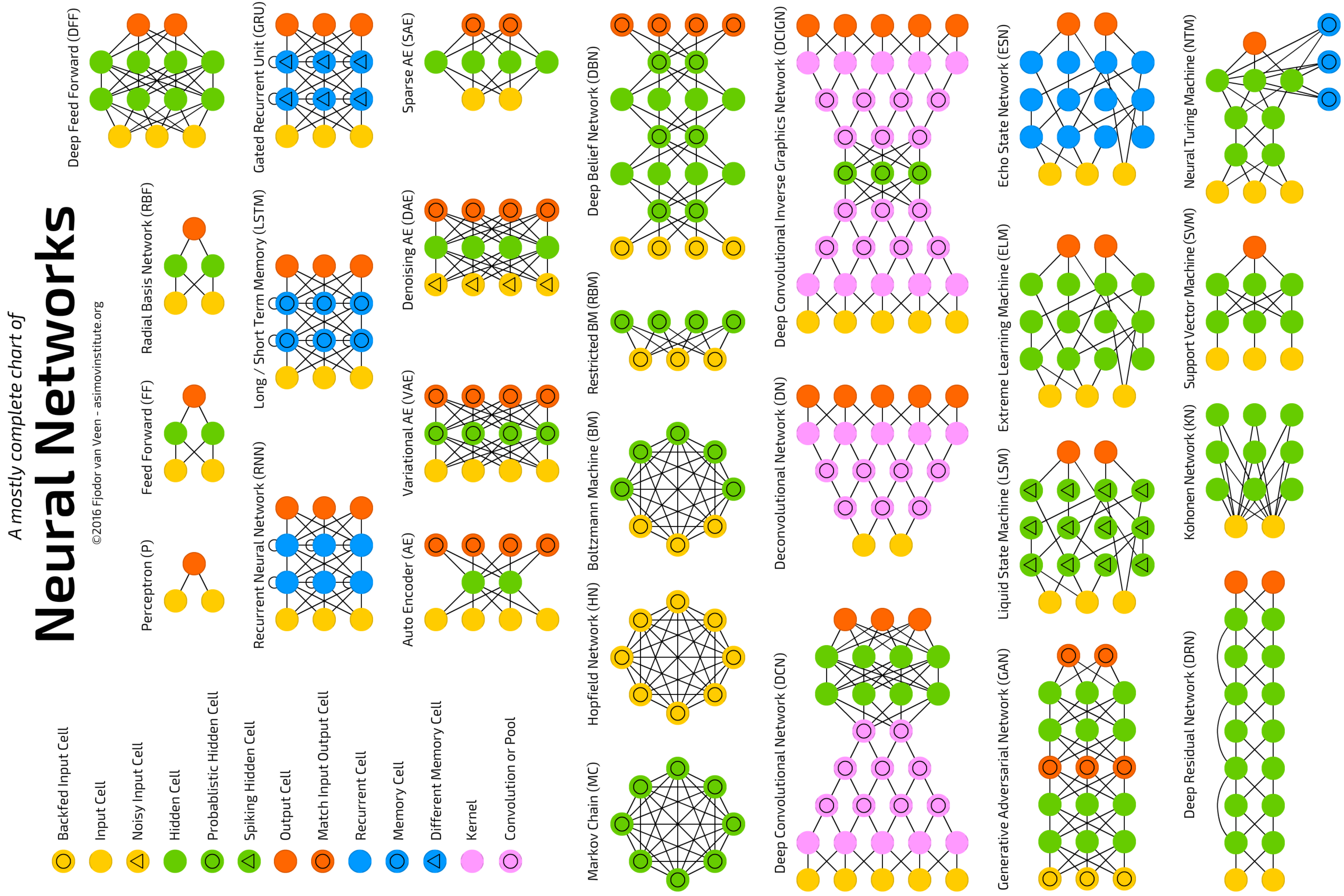


Auto-Encoder Network - Transfer Learning

	x_1	x_2	$\ldots$	x_D
X^1	2.3	5.6	$\ldots$	7.9
X^2	6.6	0.4	$\ldots$	4.3
			$\ldots$	
			$\ldots$	
			$\ldots$	
			$\ldots$	
X^N			$\ldots$	

T
dog
cat
dog
cat
T^N





https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464