ACENET
accelerate discovery

Basic BP ANNs

with

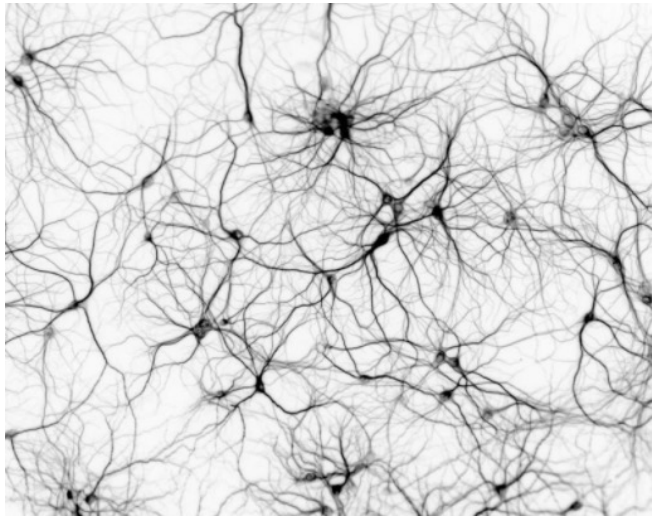
Daniel L. Silver, Ph.D.

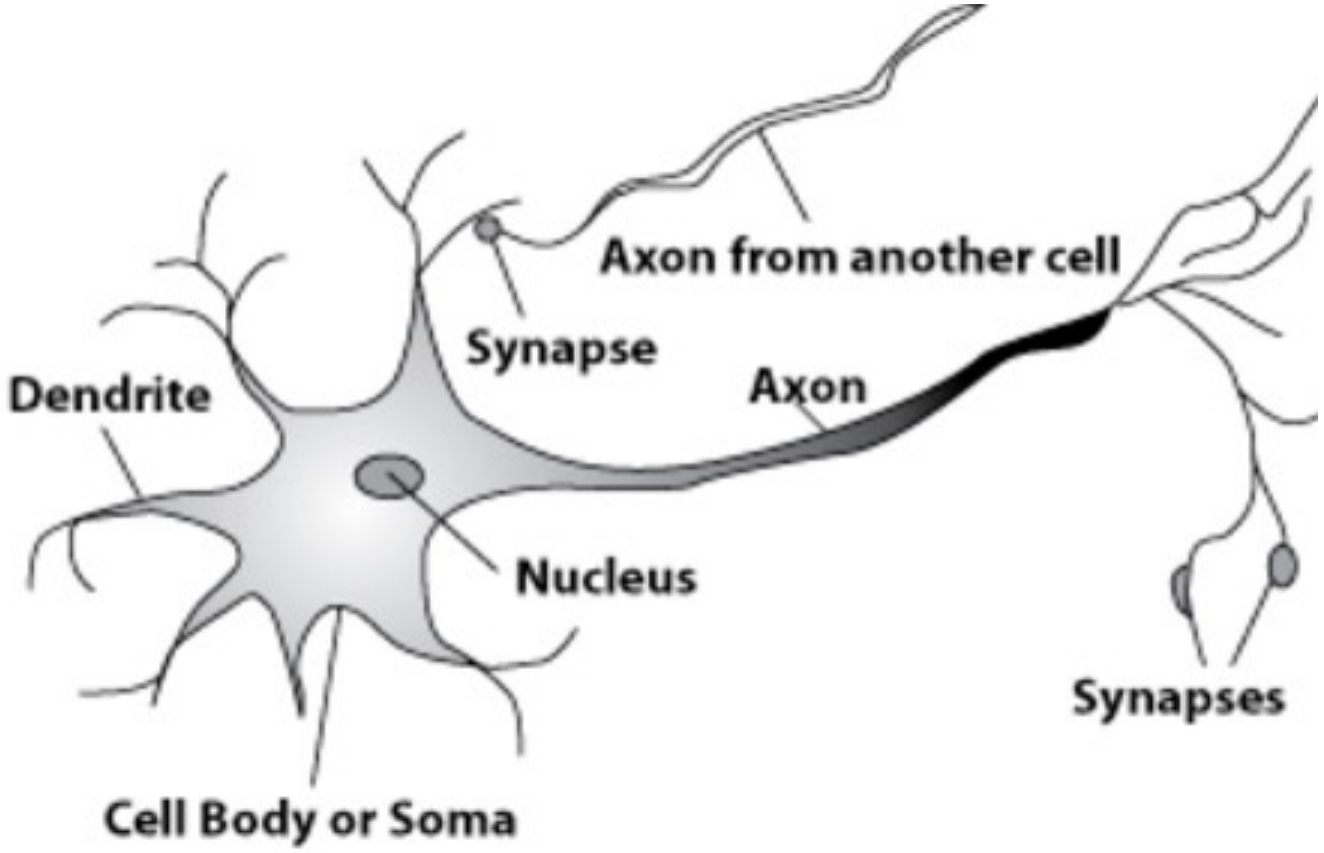
Andy McIntyre, Ph.D.

June 24, 2022

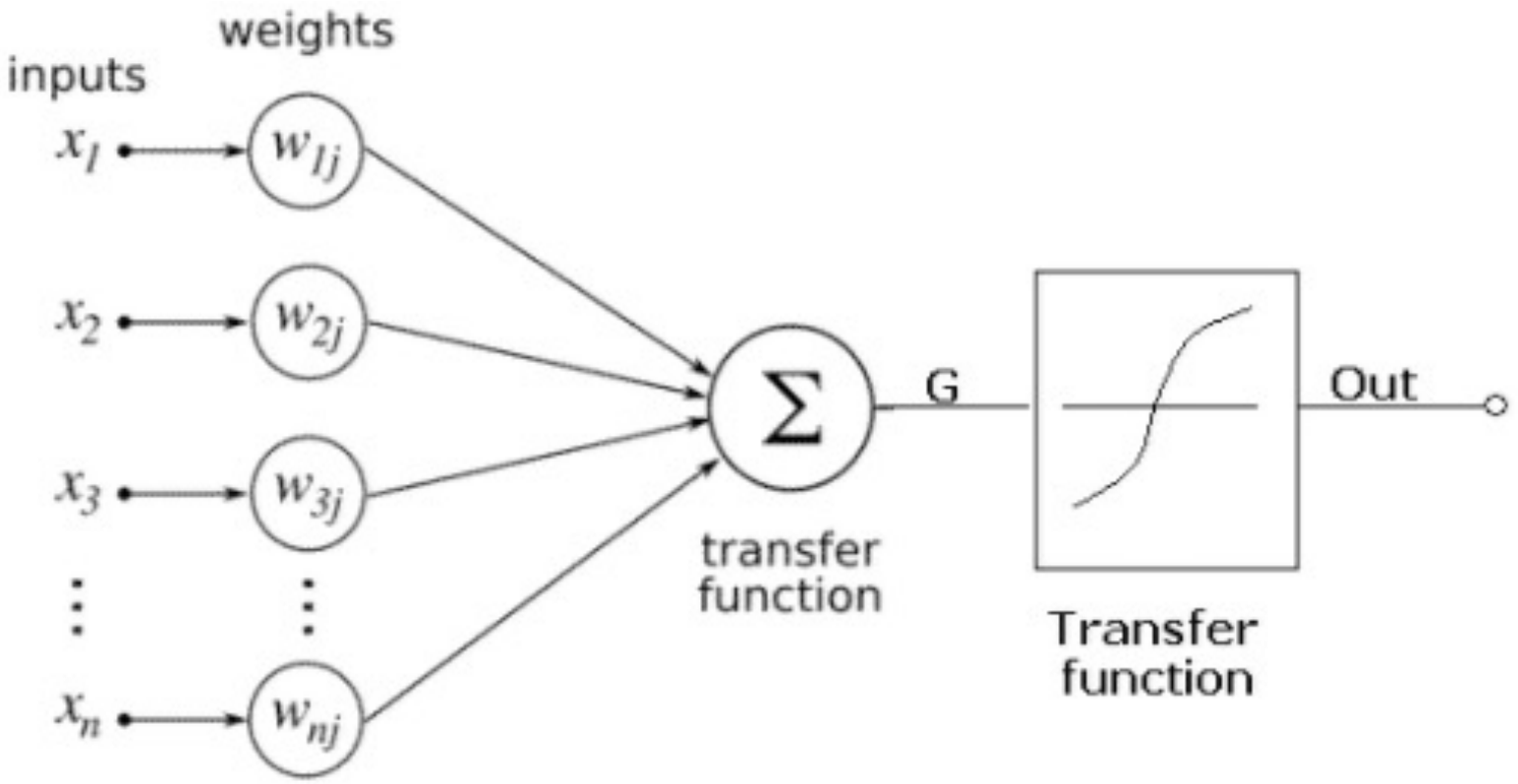
ACADIA INSTITUTE
FOR DATA ANALYTICS
digital harvest

From Biological to Artificial Neurons





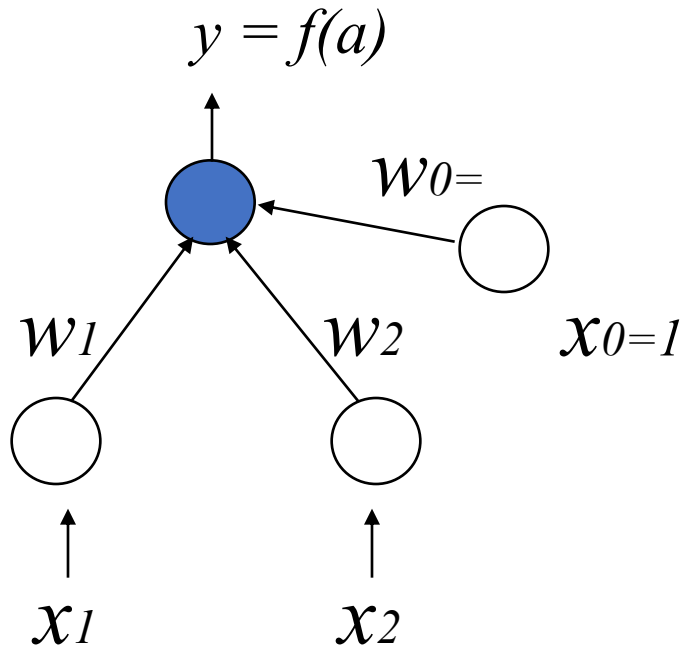
From Biological to Artificial Neurons



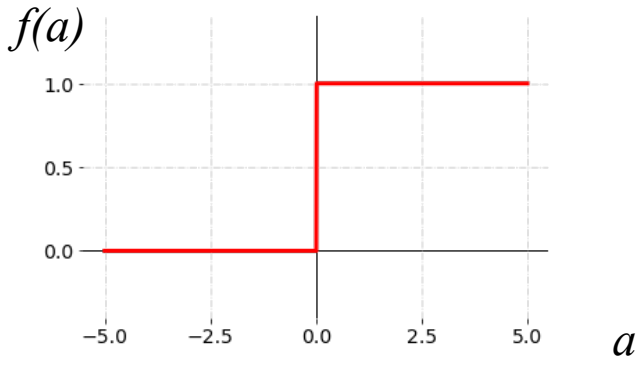
Learning in a Simple Neuron

Logical OR Function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



where f(a) is the step function, such that: f(a)=1, a > 0
f(a)=0, a <= 0



Learning in a Simple Neuron

Perceptron Learning Algorithm:

1. **Initialize weights**
2. **Present a pattern and target output**
3. **Compute output :**
4. **Update weights :**

$$y = f[\sum_{i=0}^{2} w_i x_i]$$

$$w_i(t+1) = w_i(t) + \Delta w_i$$

Repeat starting at 2 until acceptable level of error

This is an online method of computing a regression = fitting a function to a set of data points

Learning in a Simple Neuron

Widrow-Hoff or Delta Rule for Weight Modification

Where: $w_i(t+1) = w_i(t) + \Delta w_i$; $\Delta w_i = \eta d x_i(t)$

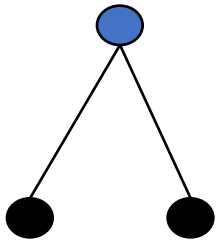
η = *learning rate* (*o* < η <= *1*), typically set = *0.1*

d = *error signal* = desired output - network output. = $t - y$

Fundamentally a gradient descent method of search

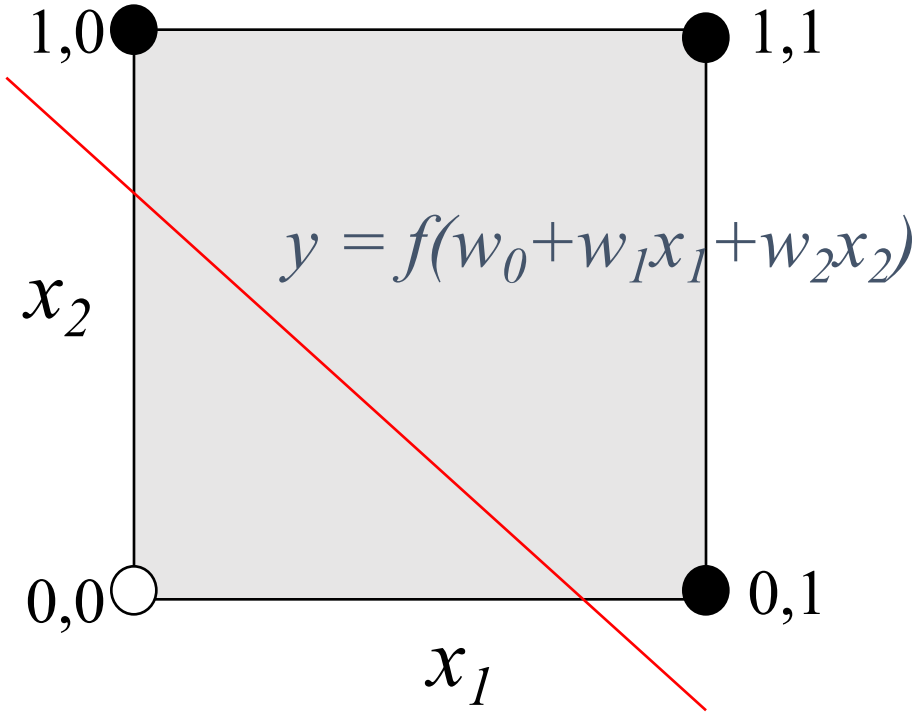
What is an artificial neuron doing when it learns?

Simple
Neural Network



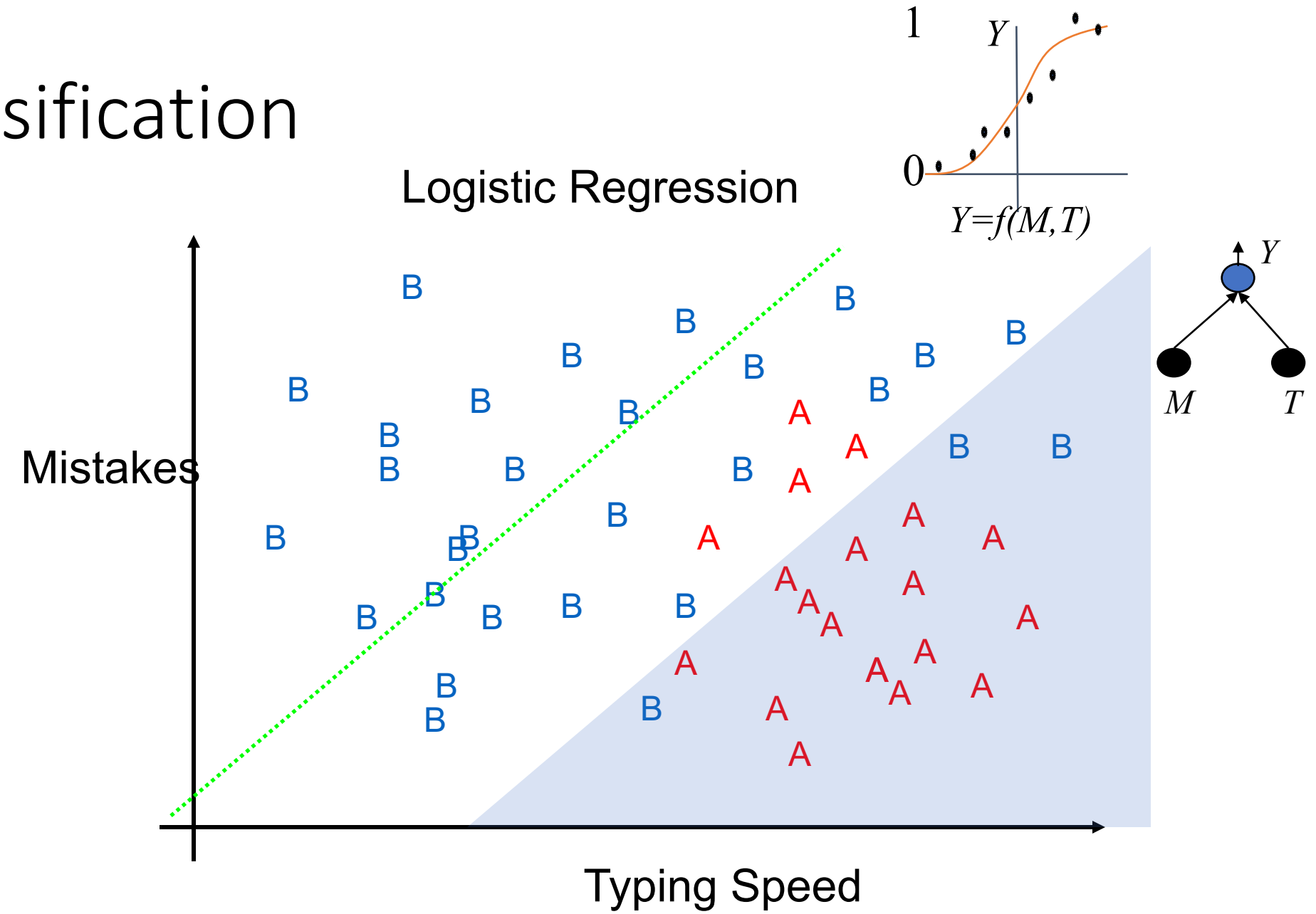
Logical OR
Function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



The discriminate function divides up the 0 and 1 class examples.

Classification



Limitations of Simple Neural Networks

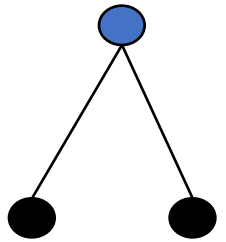
The Limitations of Perceptrons

(Minsky and Papert, 1969)

- Able to form only *linear discriminate functions*; *i.e.* classes which can be divided by a line or hyper-plane
- Most functions are more complex; *i.e.* they are *non-linear* or not *linearly separable*
- This crippled research in neural net theory for 15 years ....

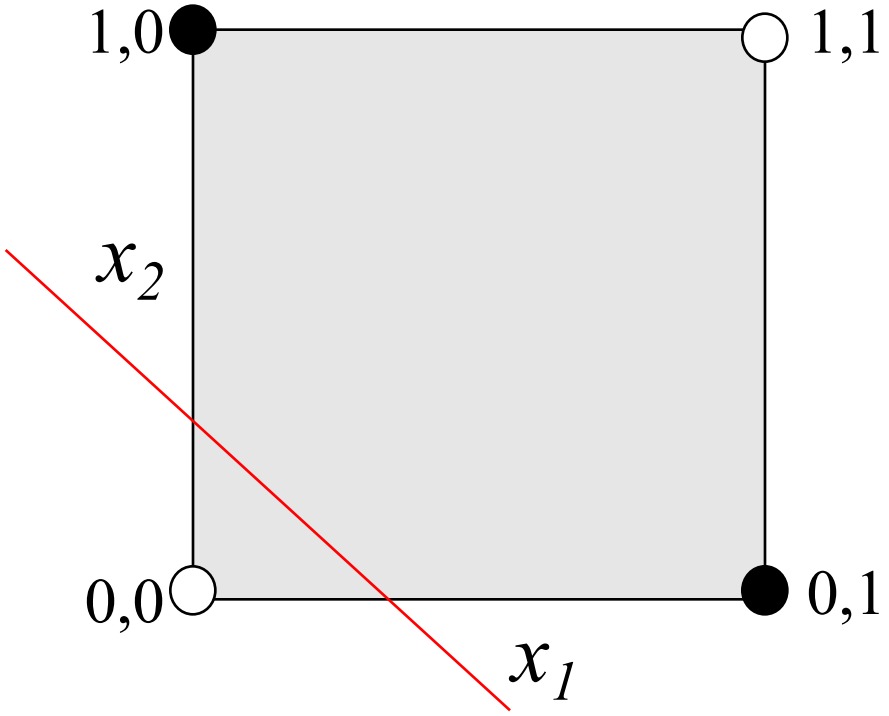
EXAMPLE

Simple
Neural Network



Logical XOR
Function

x1	*x2*	*y*
0	0	0
0	1	1
1	0	1
1	1	0

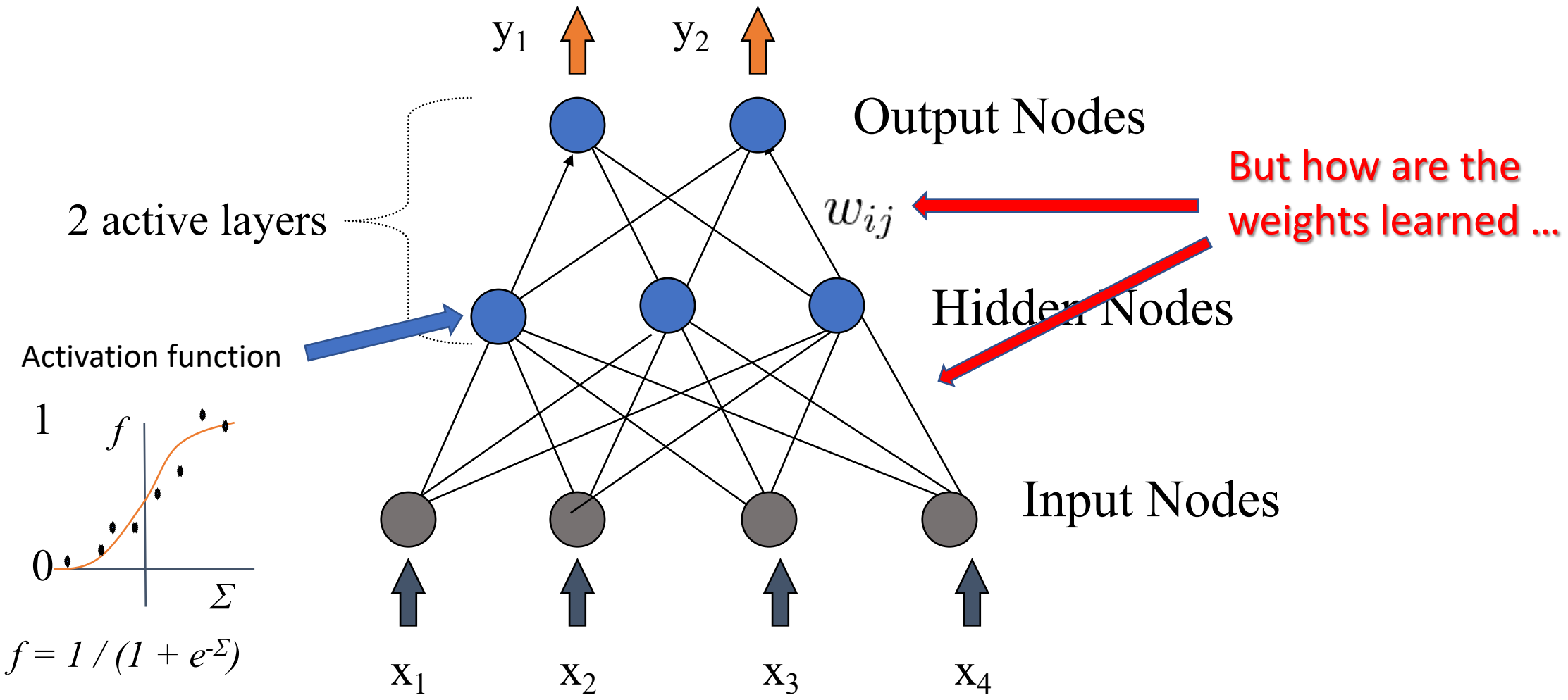


One linear discriminant function cannot divide up the 0 and 1 class examples.

TUTORIAL #1(a)

- Develop and train a simple neural network to learn the OR and XOR function (perceptron.ipynb)

Multi-layer Feed-forward ANNs

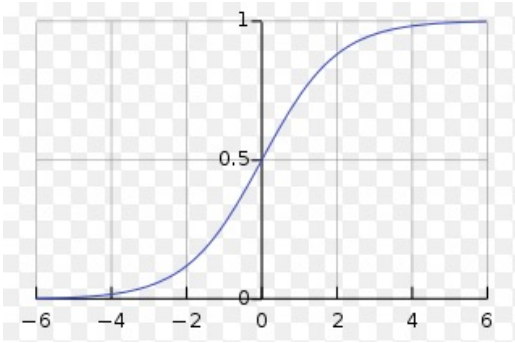


Multi-layer Feed-forward ANNs

Over the 15 years (1969-1984) research continued ...

- *hidden layer* of nodes allowed combinations of linear functions
- *non-linear activation functions* displayed properties closer to real neurons:
  - output varies continuously but not linearly
  - differentiable .... *Sigmoid* $f(a) = 1/(1 + e^{-a})$
- non-linear ANN classifier was possible
- But how are the weights learned …

$p = 1/(1+e^{-\Sigma})$



$\Sigma_{j=0..k} w_j x_j$

The Back-propagation Algorithm

- 1986: the solution to multi-layer ANN weight update rediscovered
- Conceptually simple - the global error is backward propagated to network nodes, weights are modified proportional to their contribution
- Most important ANN learning algorithm
- Become known as *back-propagation* because the error is sent back through the network to correct all weights

The Back-propagation Algorithm

- Like the Perceptron - calculation of error is based on difference between target and actual output:

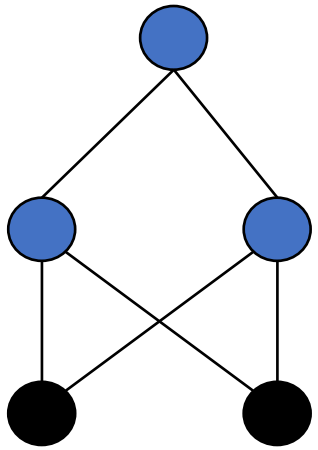
$$E = \frac{1}{2}\sum_j (t_j - o_j)^2$$

- However in BP it is the rate of change of the error with respect to each weight - the important feedback through the network
  generalized delta rule

$$\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}}$$

- Relies on a continuous non-linear activation function (sigmoid) for communication between layers

The Back-propagation Algorithm



Where: $$\Delta w_{ij} = -\eta * \frac{\delta E}{\delta w_{ij}} = \eta * \delta_j * o_i$$

For output nodes:

$$\delta_j = o_j(1 - o_j)(t_k - o_j)$$

For hidden nodes:

$$\delta_i = o_i(1 - o_i) \sum_j w_{ij}\delta_j$$

The Back-propagation Algorithm

On-Line algorithm:

1. Initialize weights
2. Present a pattern and target output $o_j = f[\sum_{i=0}^{n} w_{ij} o_i]$
3. Compute output : $w_i(t+1) = w_i(t) + \Delta w_i$
4. Update weights :

where $w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}}$

Repeat starting at 2 until acceptable level of error

The Back-propagation Algorithm

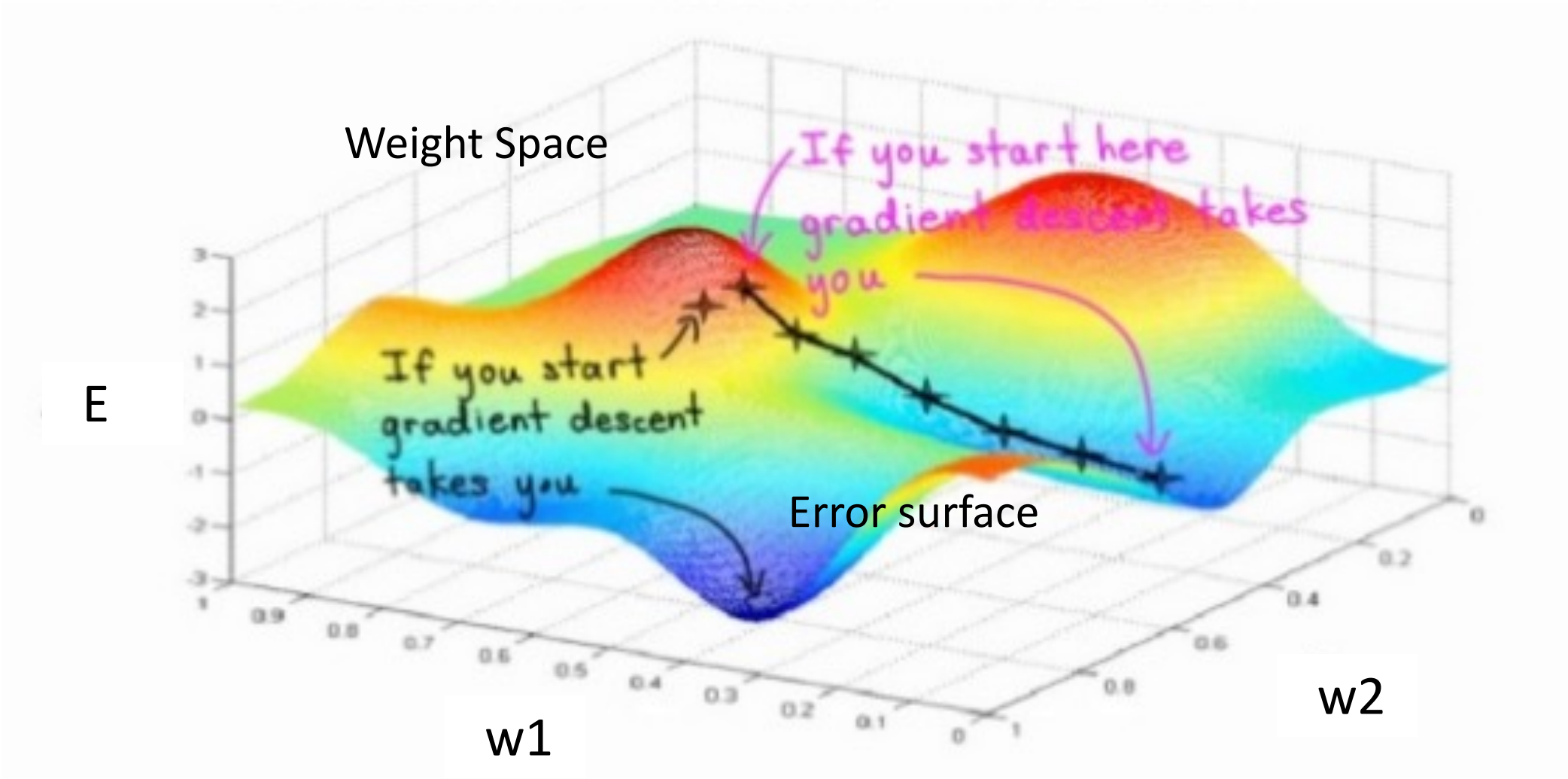
Visualizing the bp learning process:

The bp algorithm performs a *gradient descent* in weights space toward a minimum level of error using a fixed *step size* or learning rate η

The gradient is given by : $\frac{\delta E}{\delta w_{ij}}$

= rate at which error changes as weights change

Gradient Descent - Intuition



The Back-propagation Algorithm

Momentum Descent:

- Minimization can be speed-up if an additional term is added to the update equation:

$$\alpha[w_{ij}(t) - w_{ij}(t-1)]$$

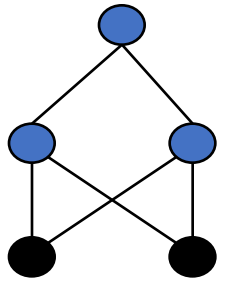
where: $0 < \alpha < 1$

$$\Delta w_{ij}(t) = \eta d_j o_i + \alpha \Delta w_{ij}(t-1)$$

- Thus:
  - Augments the effective learning rate η to vary the amount a weight is updated
  - Analogous to momentum of a ball - maintains direction
  - Rolls through small local minima
  - Increases weight update when on stable gradient

EXAMPLE

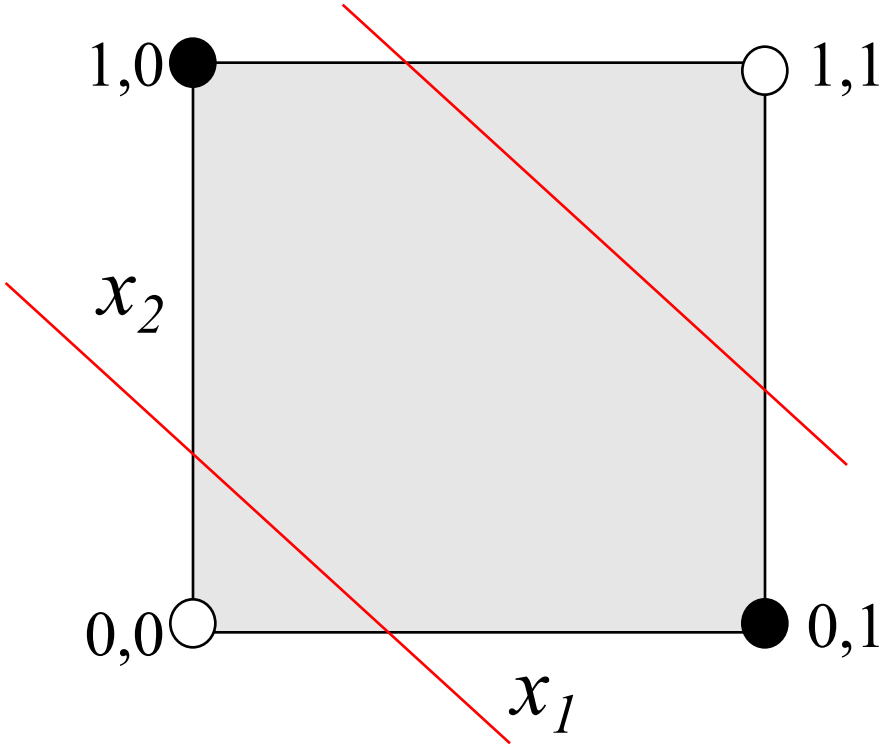
Multi-layer
Neural Network



Hidden layer
of neurons

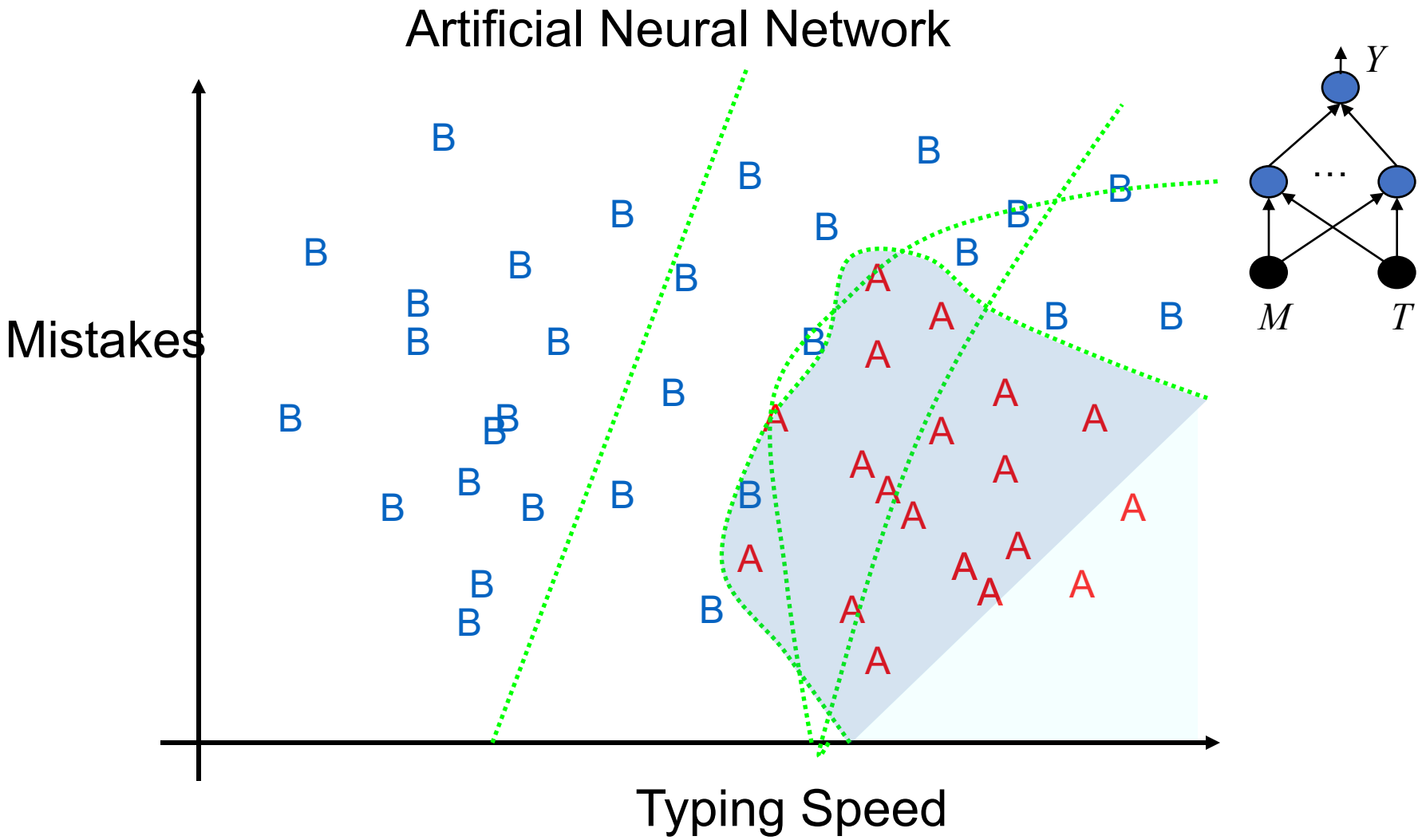
Logical XOR
Function

x1	*x2*	*y*
0	0	0
0	1	1
1	0	1
1	1	0



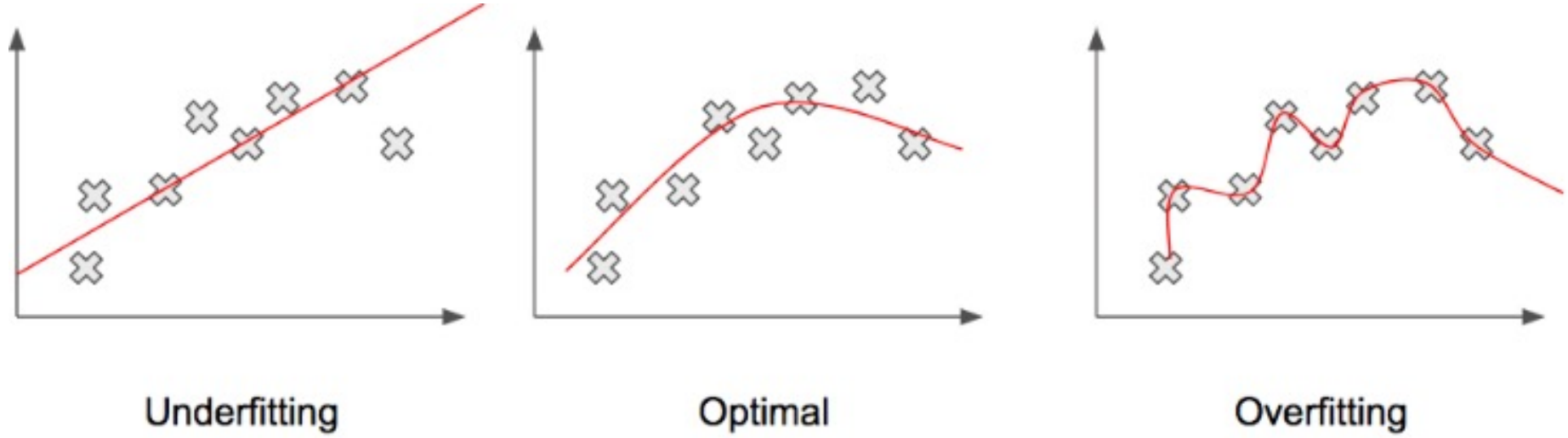
Two neurons are needed! Their combined results can produce good classification.

Classification



Generalization

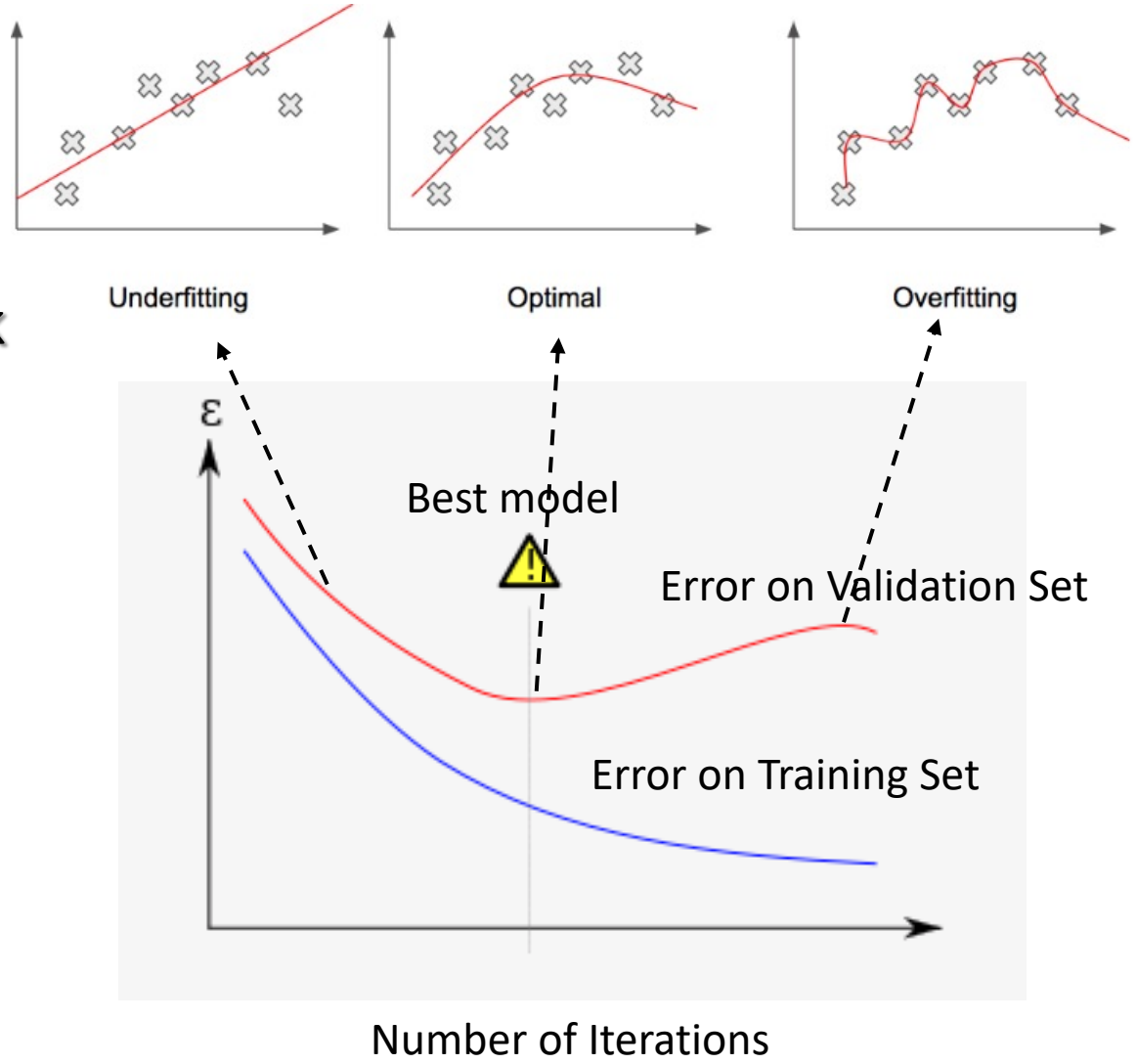
- The objective of learning is to achieve good *generalization* to new cases, otherwise just use a look-up table.
- Generalization can be defined as a mathematical *interpolation* or *regression* over a set of training points



Generalization

Preventing Over-training:

- Use a separate *validation* or *tuning set* of examples
- Monitor error on the val. set as network trains
- Stop training just prior to over-fit error occurring = *early stopping* or *tuning*
- Number of effective weights is reduced
- Most new systems have automated early stopping methods



TUTORIAL #1(b)

- Develop and train a backpropagation network to learn XOR function (mlp_bp_ann.ipynb)