

CS6510  
Applied Machine Learning

# Classifiers

9 Sep 2017

Vineeth N Balasubramanian



# Classification Methods

- k-Nearest Neighbors
- Decision Trees
- Naïve Bayes
- Support Vector Machines
- Logistic Regression
- Neural Networks (Deep Learning)
- Ensemble Methods (Boosting, Random Forests)

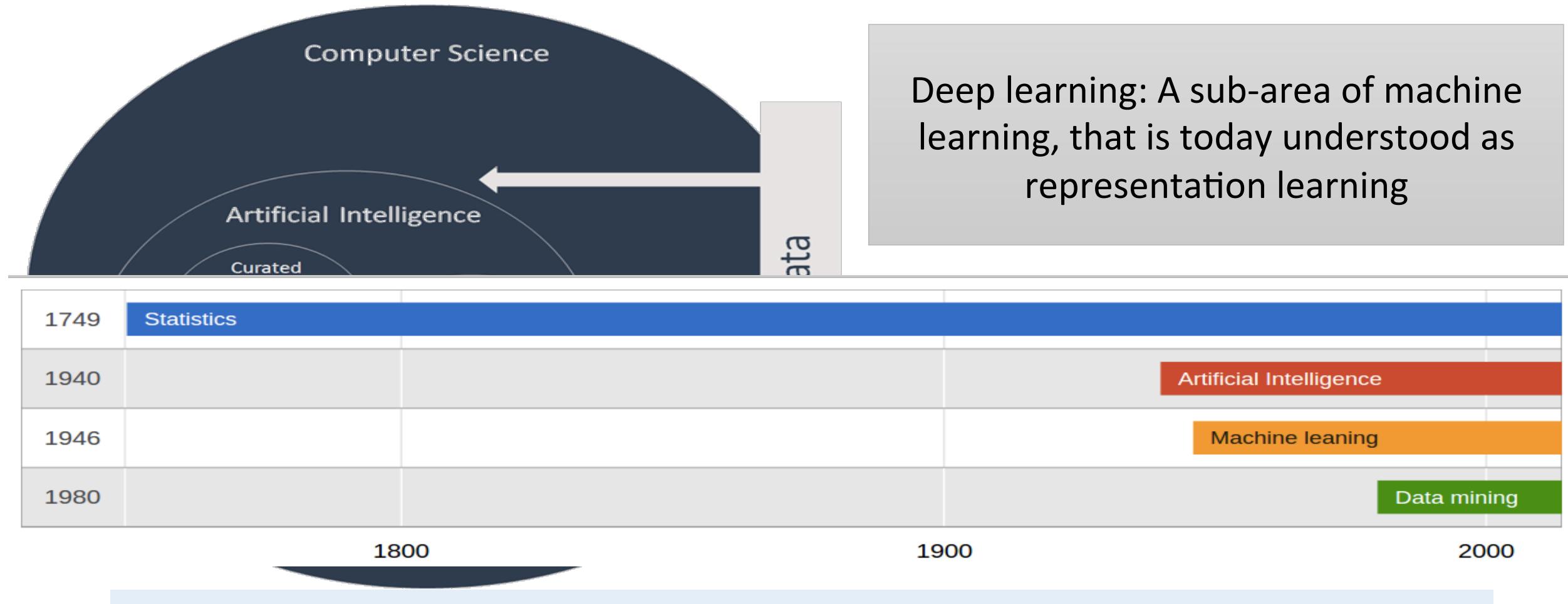
# Relevance

## AlphaGo: A Frontier



# Machine Learning (vs) Artificial Intelligence

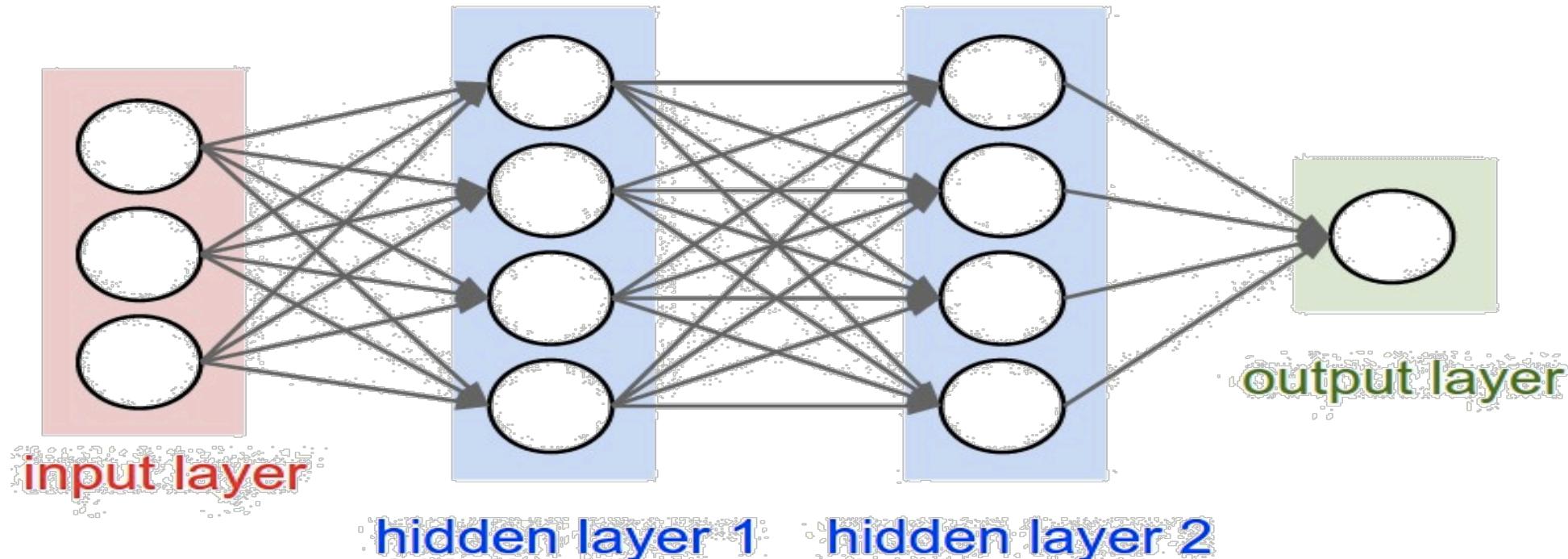
## Introduction



# Deep Learning

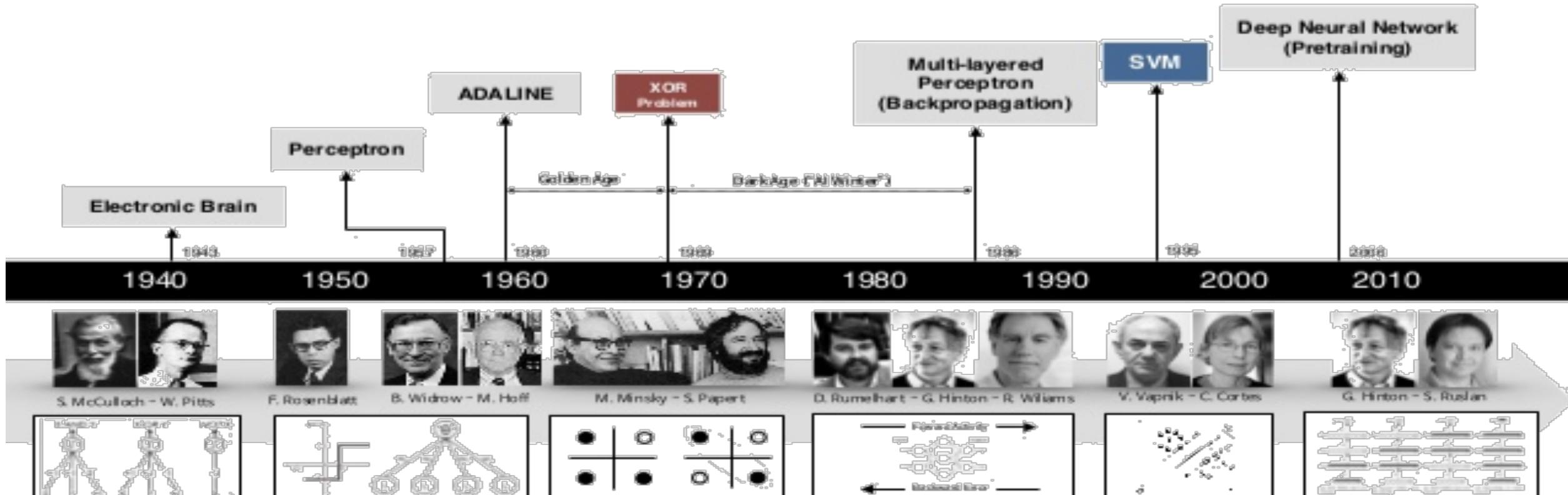
## Introduction

- Rebirth of neural networks
- Inspired by the human brain (networks of neurons)



# Deep Learning

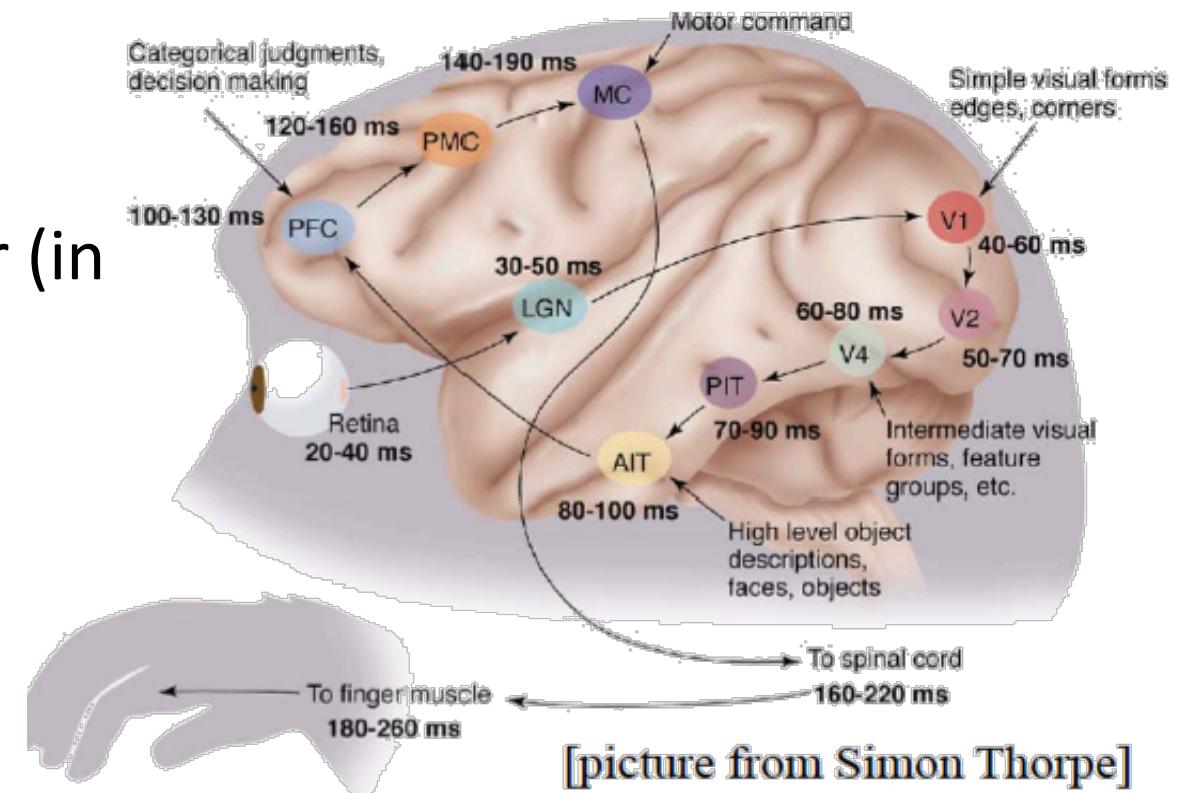
## History



# Deep Learning

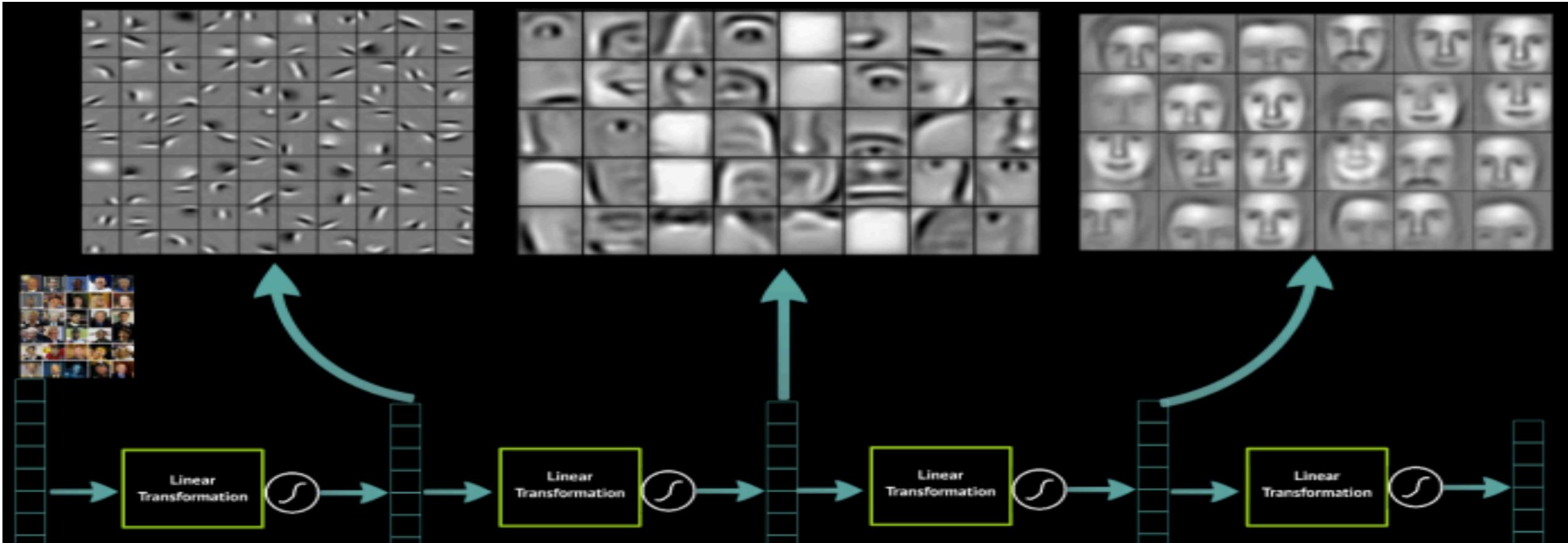
## Why is it successful?

- Learns representations of data that are useful (Other ML algorithms are “shallow”)
- Similar to the human brain
- Then, why was it not successful earlier (in the 90s)?
  - Computational power
  - Data power



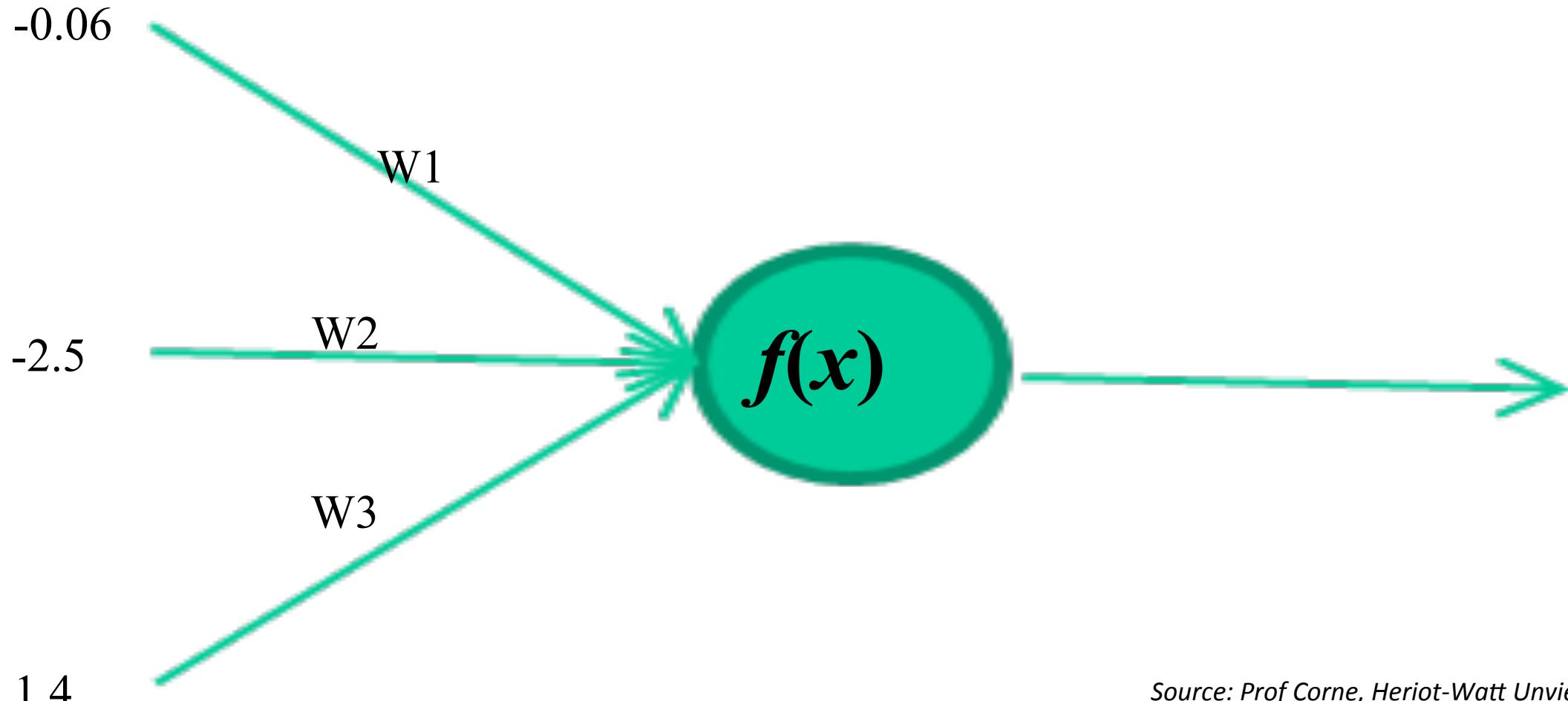
# Deep Learning

Why is it successful?



# Deep Learning

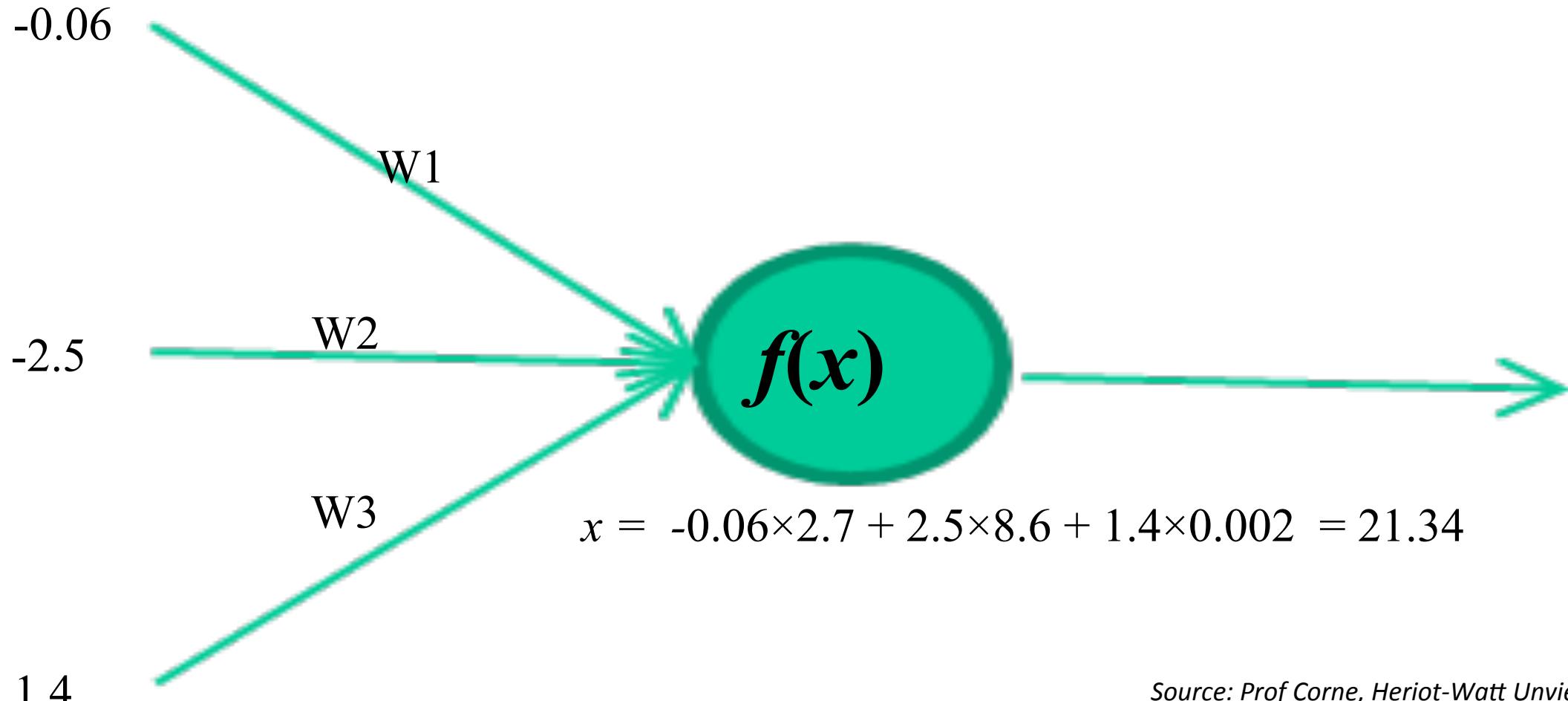
How do they learn?



Source: Prof Corne, Heriot-Watt University, UK

# Deep Learning

How do they learn?



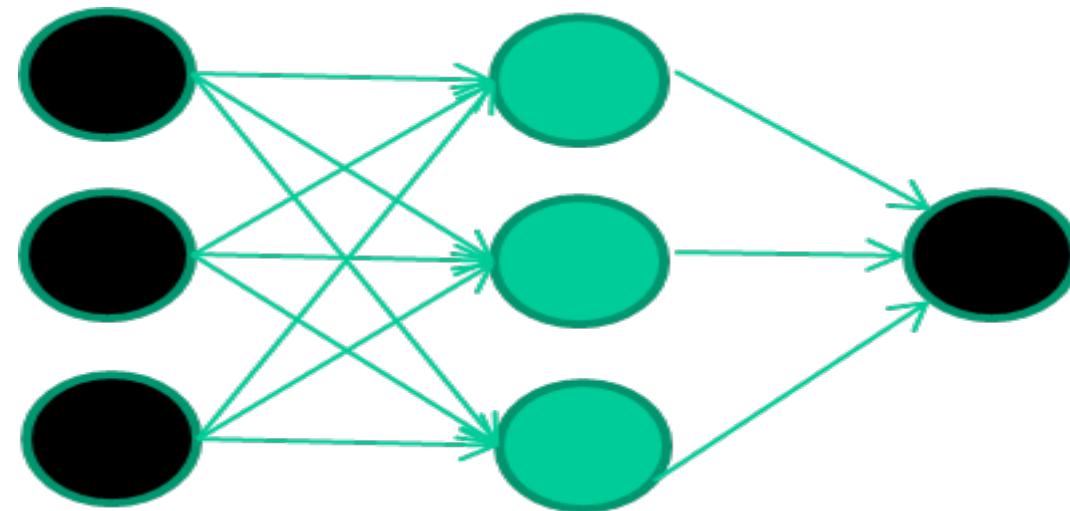
Source: Prof Corne, Heriot-Watt University, UK

# Deep Learning

How do they learn?

*A dataset*

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	



Source: Prof Corne, Heriot-Watt University, UK

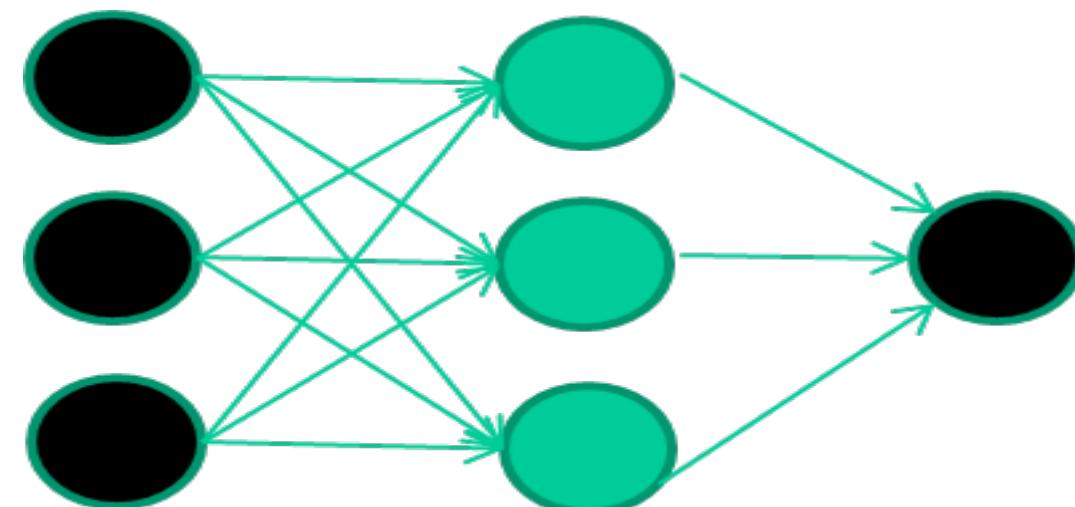
# Deep Learning

How do they learn?

*Training data*

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Initialise with random weights



Source: Prof Corne, Heriot-Watt University, UK

# Deep Learning

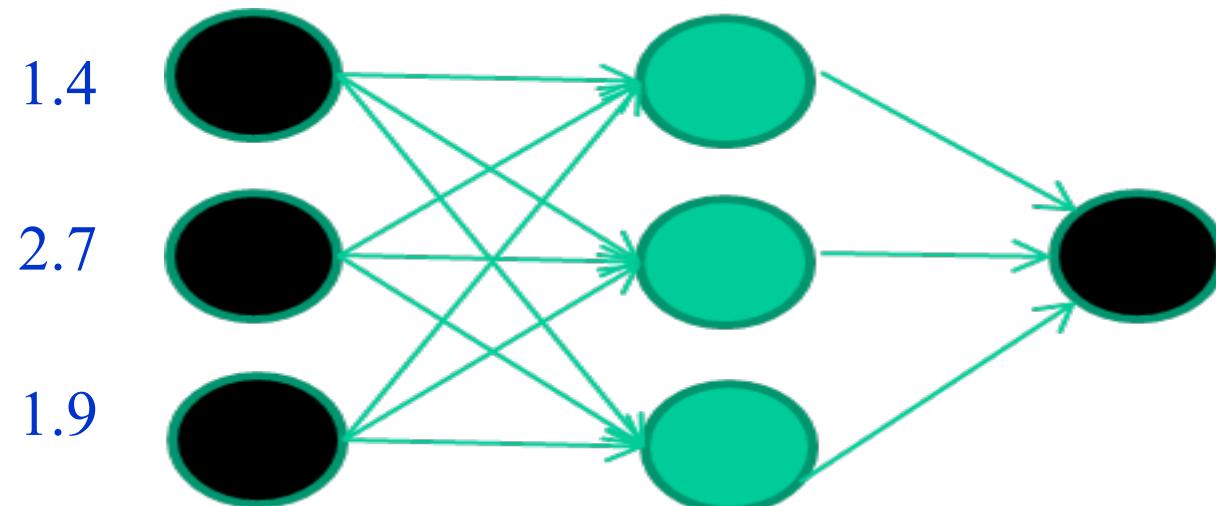
How do they learn?

*Training data*

*Fields*                    *class*

Fields	class
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	

Present a training pattern



Source: Prof Corne, Heriot-Watt University, UK

# Deep Learning

How do they learn?

*Training data*

*Fields*

*class*

1.4	2.7	1.9	0
-----	-----	-----	---

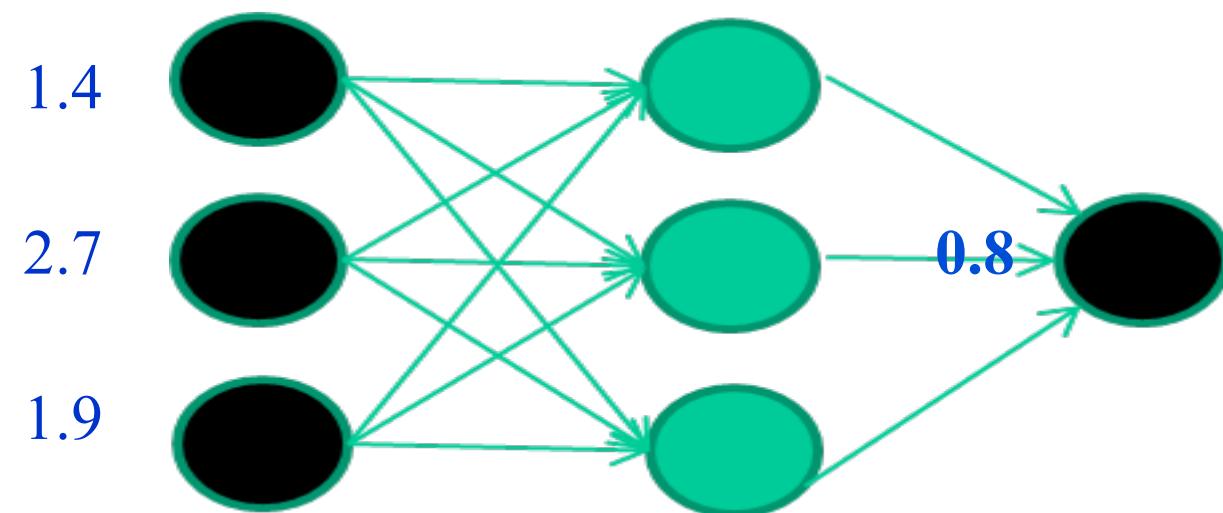
3.8	3.4	3.2	0
-----	-----	-----	---

6.4	2.8	1.7	1
-----	-----	-----	---

4.1	0.1	0.2	0
-----	-----	-----	---

etc ...

Feed it through to get output



Source: Prof Corne, Heriot-Watt University, UK

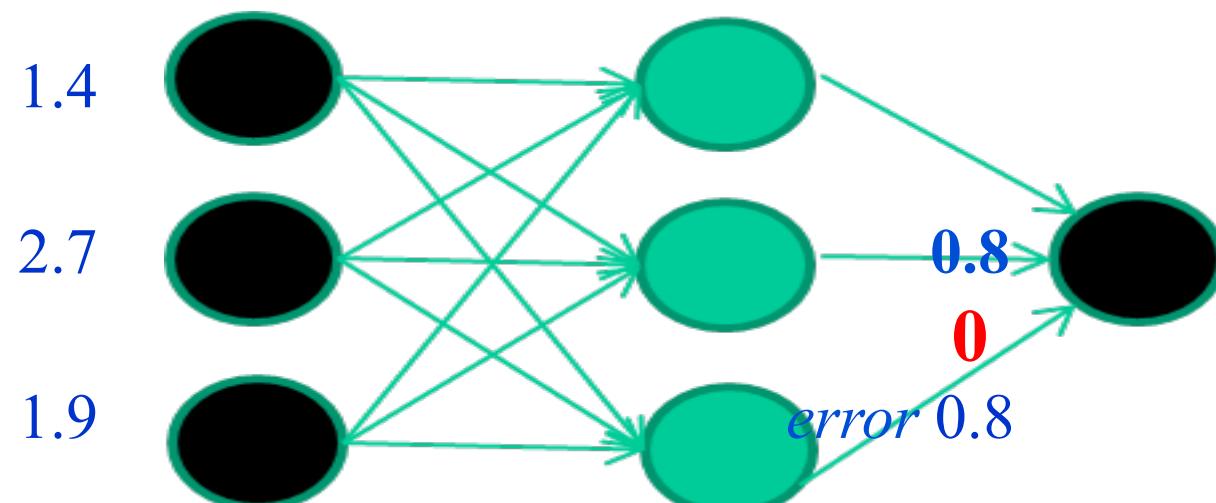
# Deep Learning

How do they learn?

*Training data*

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	

Compare with target output



Source: Prof Corne, Heriot-Watt University, UK

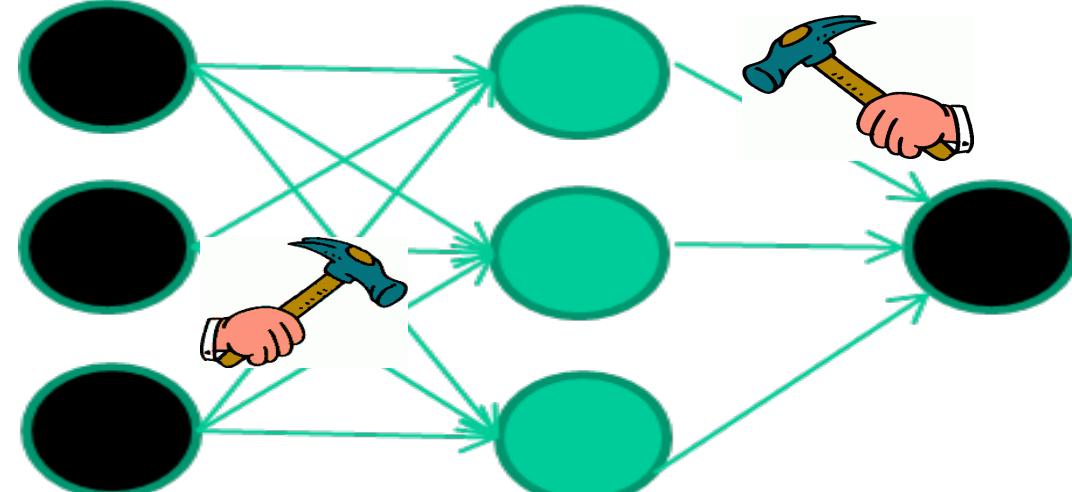
# Deep Learning

How do they learn?

*Training data*

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	

Adjust weights based on error



Source: Prof Corne, Heriot-Watt University, UK

# Deep Learning

How do they learn?

*Training data*

*Fields*              *class*

1.4  2.7  1.9        0

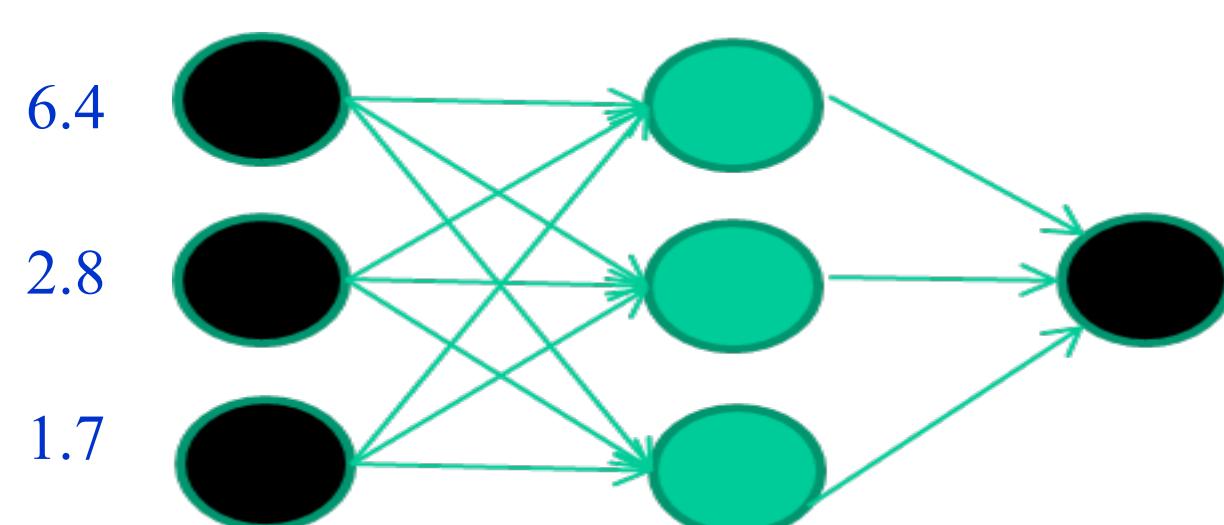
3.8  3.4  3.2        0

6.4  2.8  1.7        1

4.1  0.1  0.2        0

etc ...

Present a training pattern



Source: Prof Corne, Heriot-Watt University, UK

# Deep Learning

How do they learn?

*Training data*

*Fields*              *class*

1.4  2.7  1.9        0

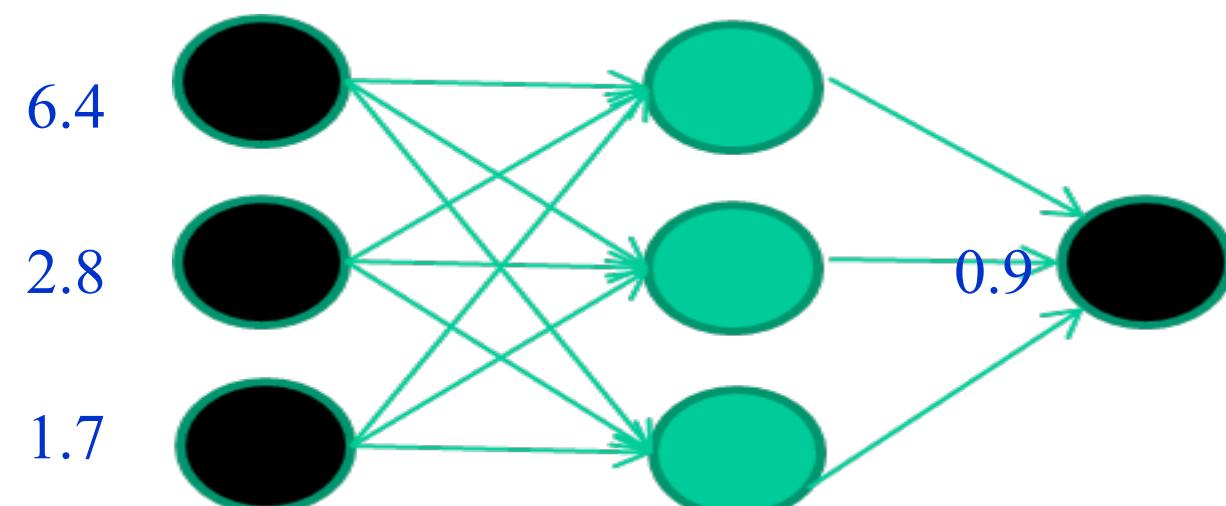
3.8  3.4  3.2        0

6.4  2.8  1.7        1

4.1  0.1  0.2        0

etc ...

Feed it through to get output



Source: Prof Corne, Heriot-Watt University, UK

# Deep Learning

How do they learn?

*Training data*

*Fields*              *class*

1.4 2.7 1.9 0

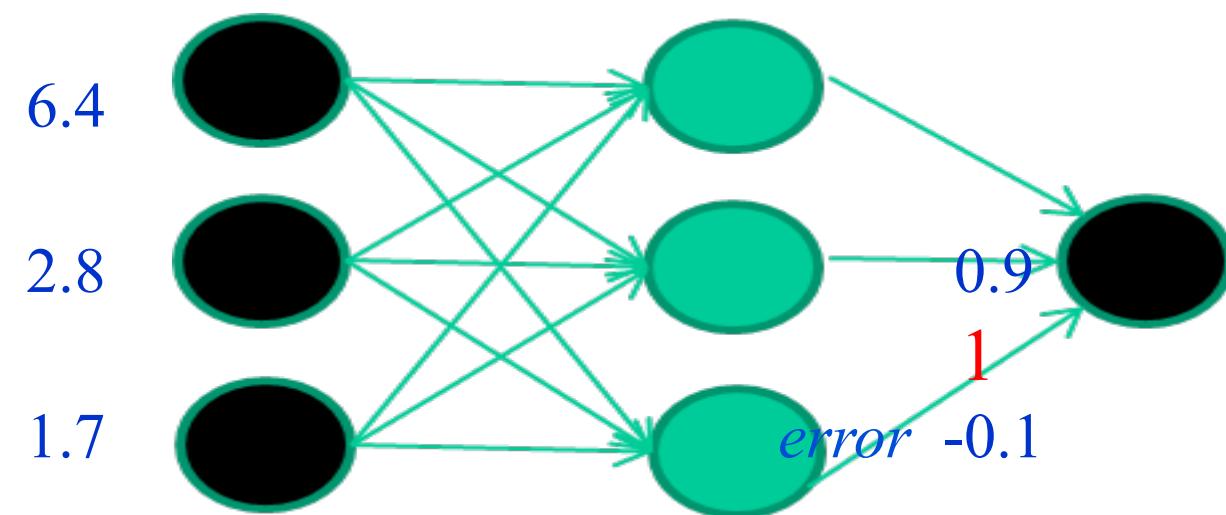
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Compare with target output



Source: Prof Corne, Heriot-Watt University, UK

# Deep Learning

How do they learn?

*Training data*

*Fields*            *class*

1.4 2.7 1.9 0

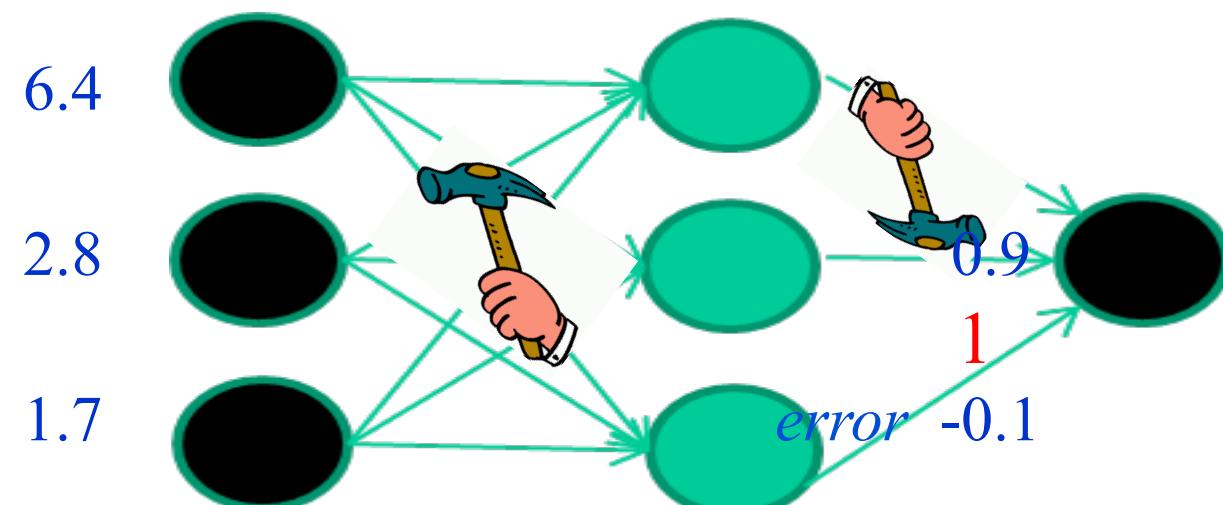
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Adjust weights based on error



Source: Prof Corne, Heriot-Watt University, UK

# Deep Learning

How do they learn?

*Training data*

**Fields**                    **class**

1.4 2.7 1.9        0

3.8 3.4 3.2        0

**6.4 2.8 1.7**        1

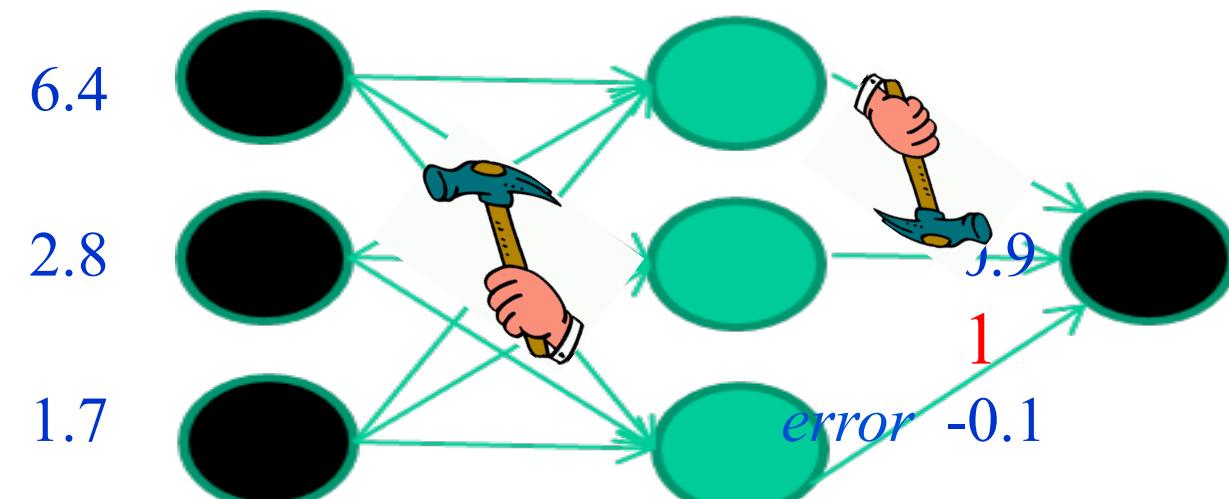
4.1 0.1 0.2        0

etc ...

Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments, reduce the error

Source: Prof Corne, Heriot-Watt University, UK

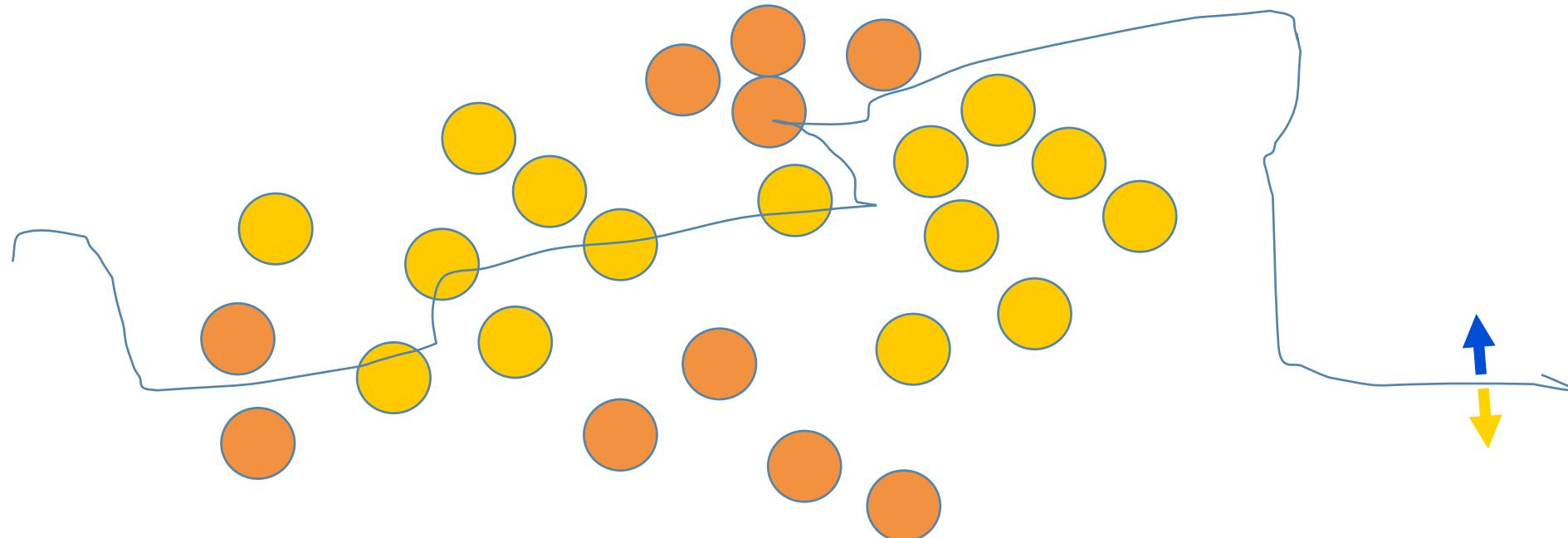
And so on ....



Called “Gradient Descent”

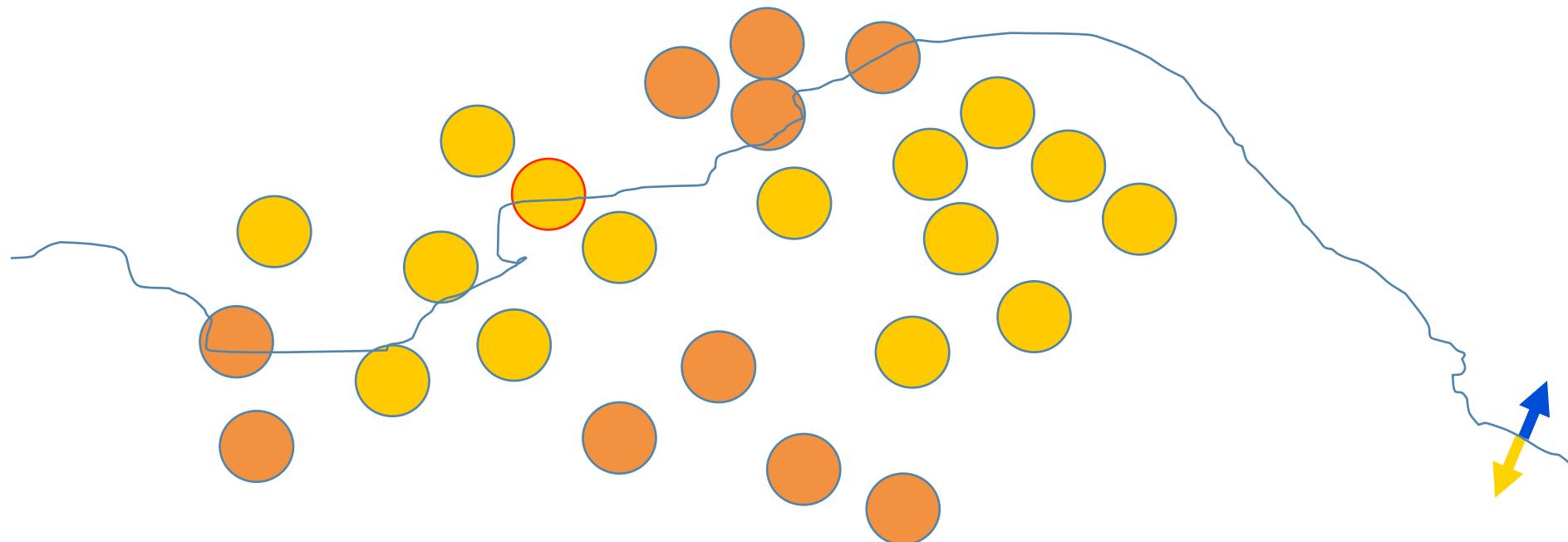
# The decision boundary perspective...

Initial random weights



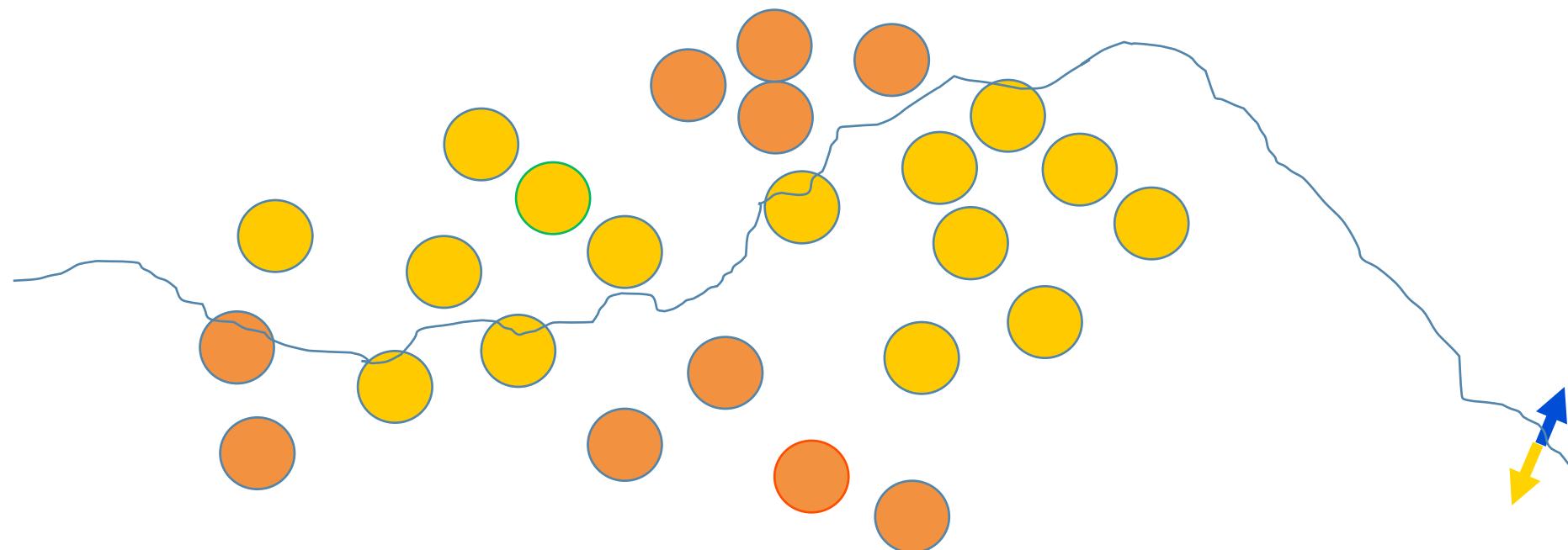
# The decision boundary perspective...

Present a training instance / adjust the weights



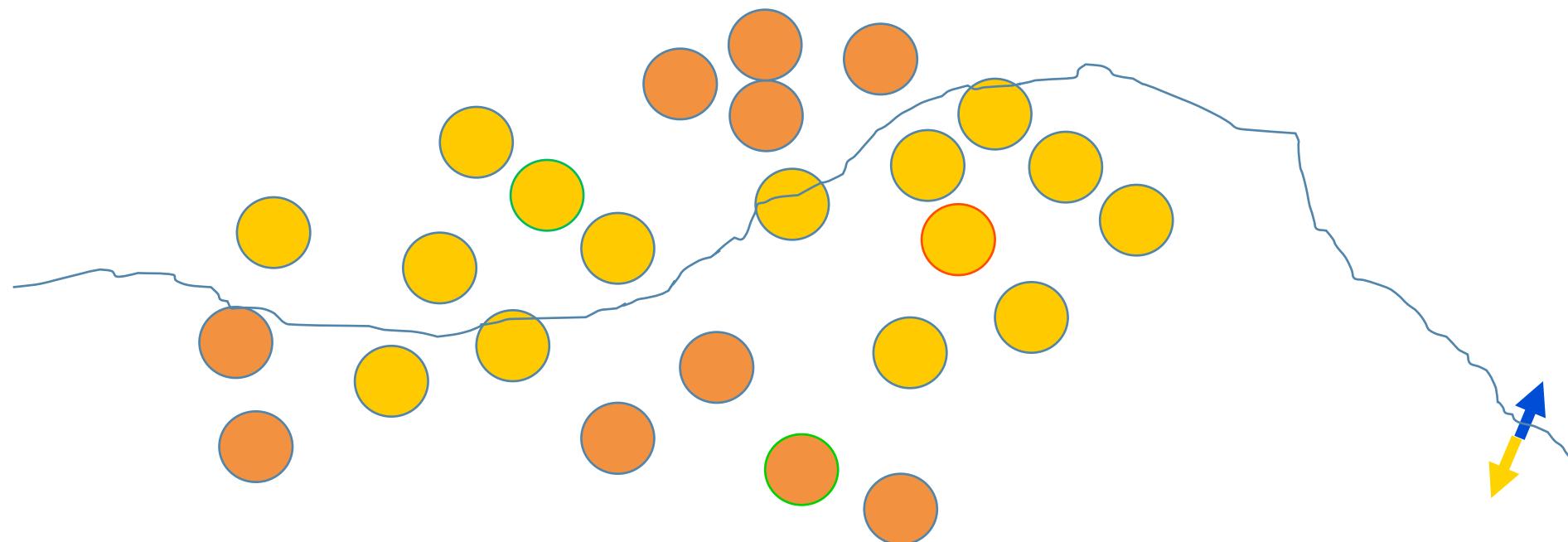
# The decision boundary perspective...

Present a training instance / adjust the weights



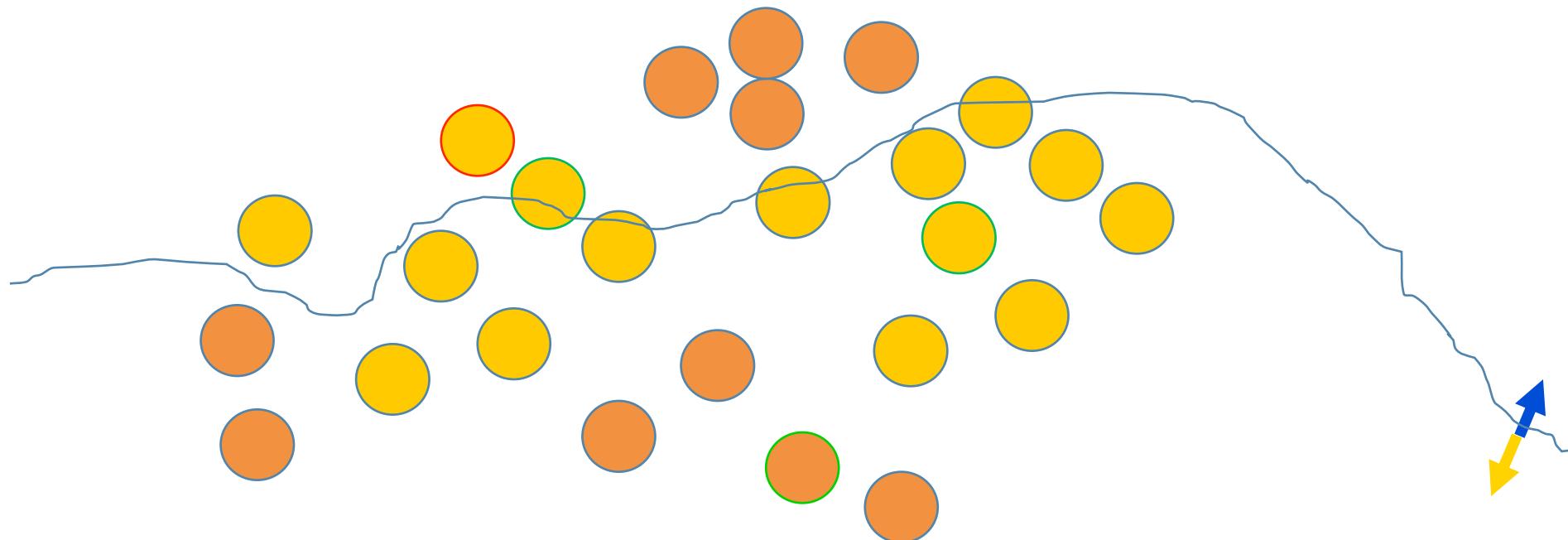
# The decision boundary perspective...

Present a training instance / adjust the weights



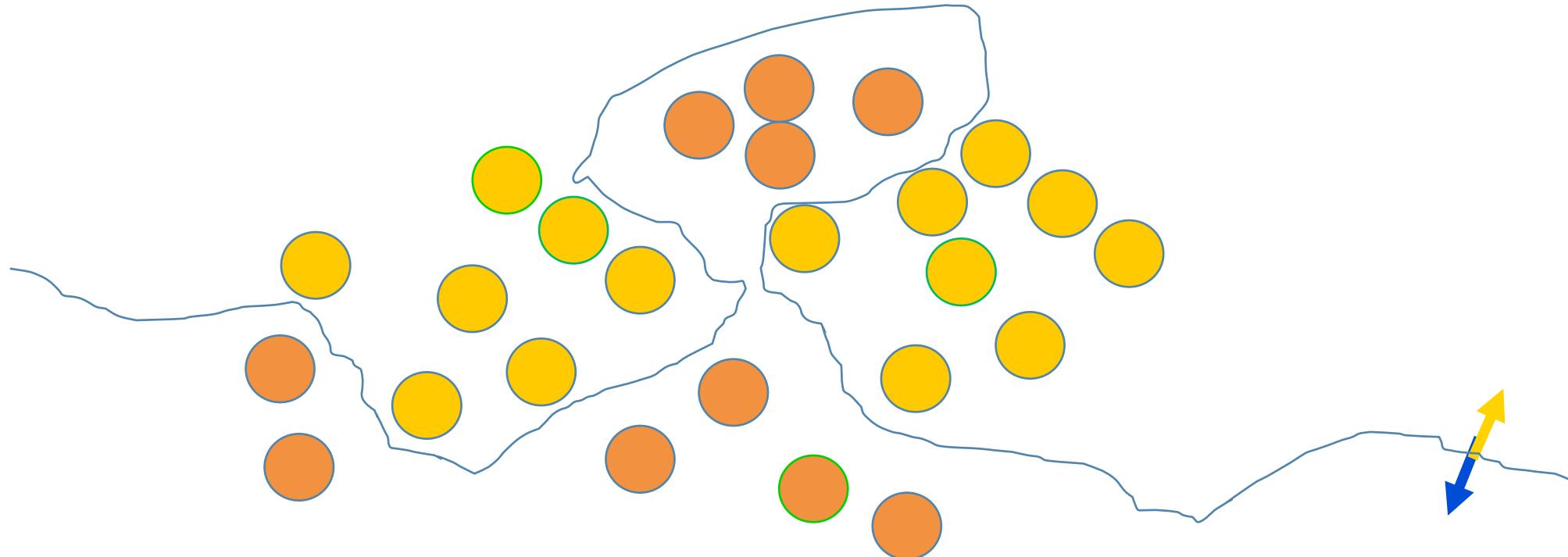
# The decision boundary perspective...

Present a training instance / adjust the weights



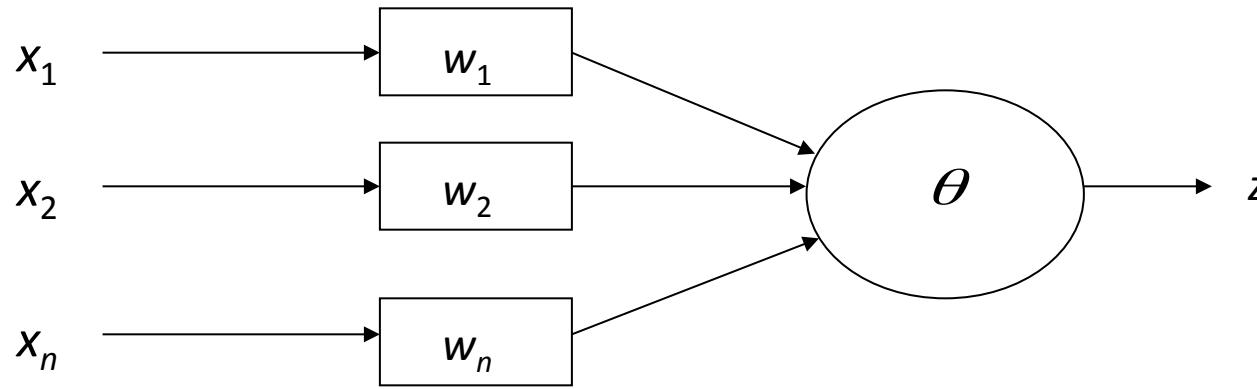
# The decision boundary perspective...

Eventually ....



# Neural Networks Training: Backpropagation

## Perceptrons



$$1 \quad \text{if} \quad \sum_{i=1}^n x_i w_i \geq \theta$$

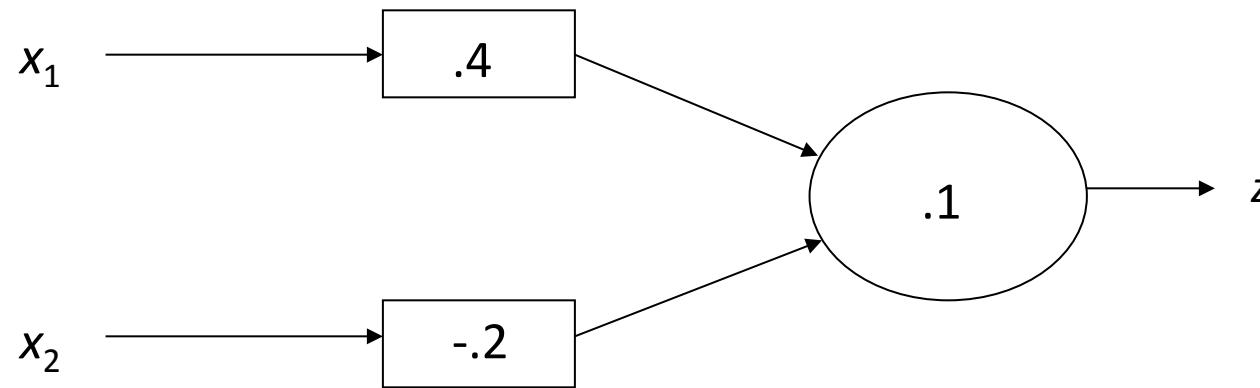
$z =$

$$0 \quad \text{if} \quad \sum_{i=1}^n x_i w_i < \theta$$

- Learn weights such that an objective function is maximized.
- What objective function should we use?
- What learning algorithm should we use?

# Neural Networks Training: Backpropagation

## Perceptrons

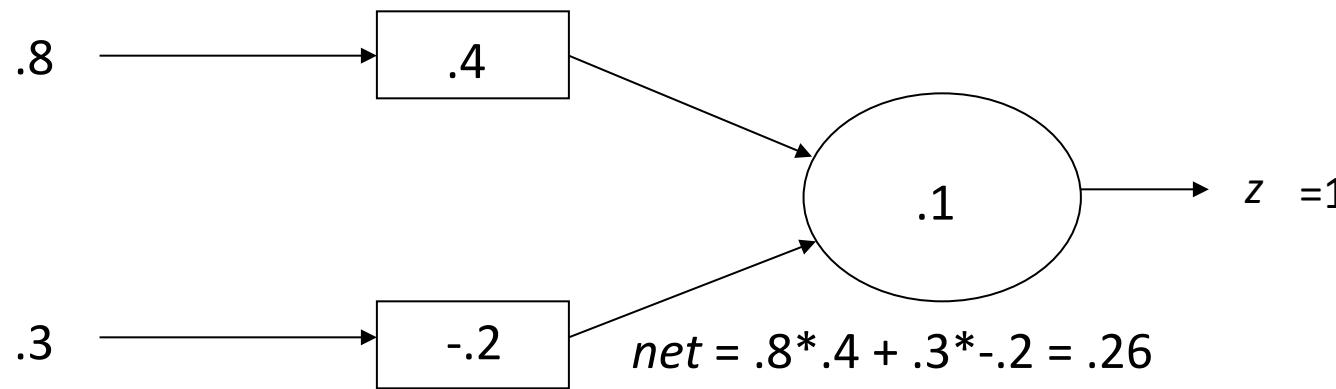


$x_1$	$x_2$	$t$
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^n x_i w_i < \theta \end{cases}$$

# Neural Networks Training: Backpropagation

## First Training Instance

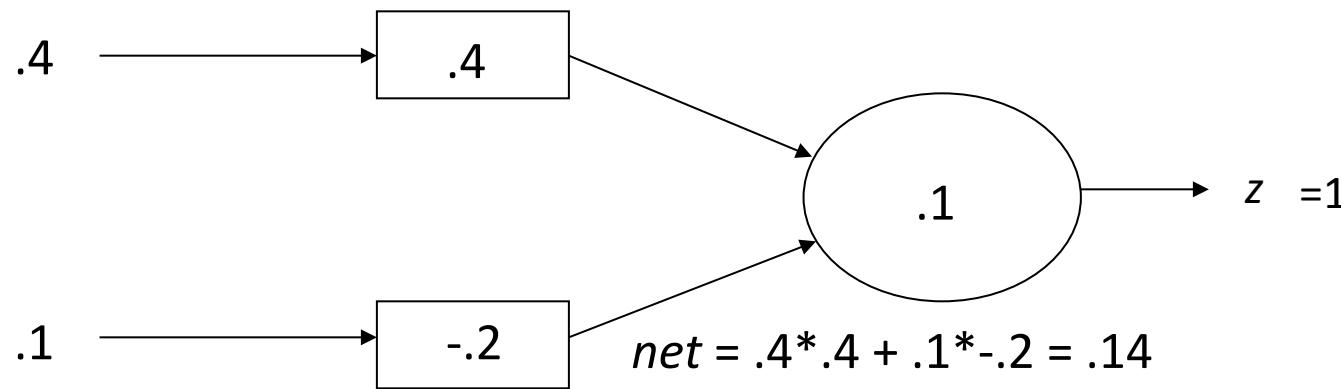


$x_1$	$x_2$	$t$
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^n x_i w_i < \theta \end{cases}$$

# Neural Networks Training: Backpropagation

## Second Training Instance



$x_1$	$x_2$	$t$
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^n x_i w_i < \theta \end{cases}$$
$$\Delta w_i = (t - z) * c * x_i$$

# Neural Networks Training: Backpropagation

## Perceptron Rule Learning

$$\Delta w_i = c(t - z) x_i$$

- Where  $w_i$  is the weight from input  $i$  to perceptron node,  $c$  is the learning rate,  $t_j$  is the target for the current instance,  $z$  is the current output, and  $x_i$  is  $i^{\text{th}}$  input
- Least perturbation principle
  - Only change weights if there is an error
  - small  $c$  rather than changing weights sufficient to make current pattern correct
  - Scale by  $x_i$
- Create a perceptron node with  $n$  inputs
- Iteratively apply a pattern from the training set and apply the perceptron rule
- Each iteration through the training set is an *epoch*
- Continue training until total training set error ceases to improve
- Perceptron Convergence Theorem: Guaranteed to find a solution in finite time if a solution exists

# Neural Networks Training: Backpropagation

## Multi-Layer Perceptrons

- Extension of perceptrons to multiple layers
- 1. **Initialize** network with **random** weights
- 2. **For all** training cases (**called examples**):
  - **a.** Present training inputs to network and calculate output
  - **b. For all layers** (starting with output layer, back to input layer):
    - i. Compare **network output** with **correct output** (error function)
    - ii. **Adapt weights** in current layer

# Neural Networks Training: Backpropagation

## Multi-Layer Perceptrons

- Method for **learning weights** in feed-forward (FF) nets
- Can't use Perceptron Learning Rule
  - no **teacher values** are possible for **hidden units**
- Use **gradient descent** to minimize the error
  - propagate deltas to adjust for errors  
backward from outputs  
to hidden layers  
to inputs

# Neural Networks Training: Backpropagation

## Multi-Layer Perceptrons

- The idea of the algorithm can be summarized as follows :

1. Computes the **error term for the output units** using the observed error.

- 2. From output layer, **repeat**
  - propagating the error term back to the previous layer and updating the weights between the two layers until the earliest hidden layer is reached.

# Neural Networks Training: Backpropagation

## Multi-Layer Perceptrons

- Initialize weights (typically random!)
- Keep doing epochs
  - For each example  $e$  in training set do
    - forward pass to compute
      - $O = \text{neural-net-output}(\text{network}, e)$
      - miss =  $(T - O)$  at each output unit
    - backward pass to calculate deltas to weights
    - update all weights
  - end
- until tuning set error stops improving

Forward pass

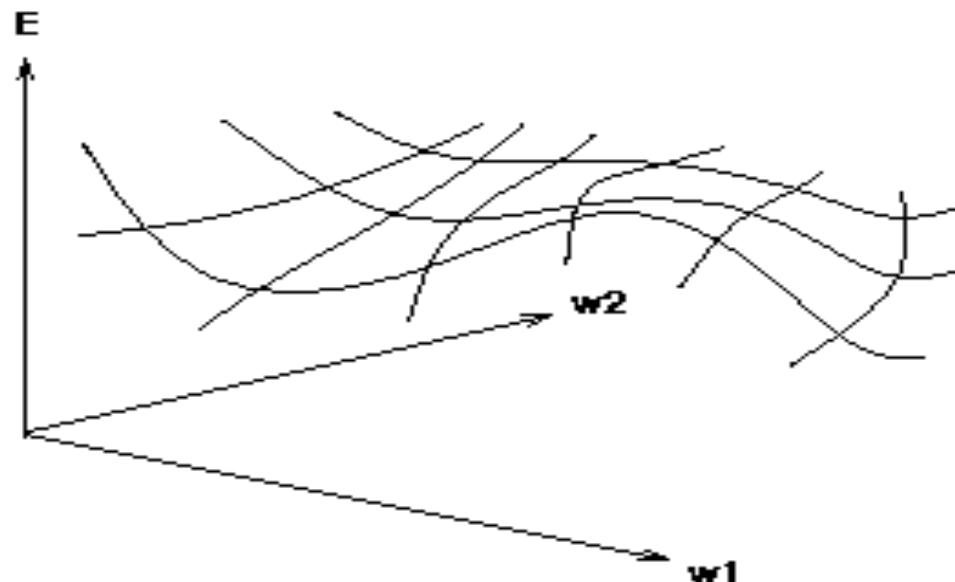
CS6510 - Applied M

Backward pass explained in next slide

# Neural Networks Training: Backpropagation

## Gradient Descent

- Think of the N weights **as a point** in an N-dimensional space
- Add a **dimension** for the observed error
- Try to **minimize your position** on the “error surface”

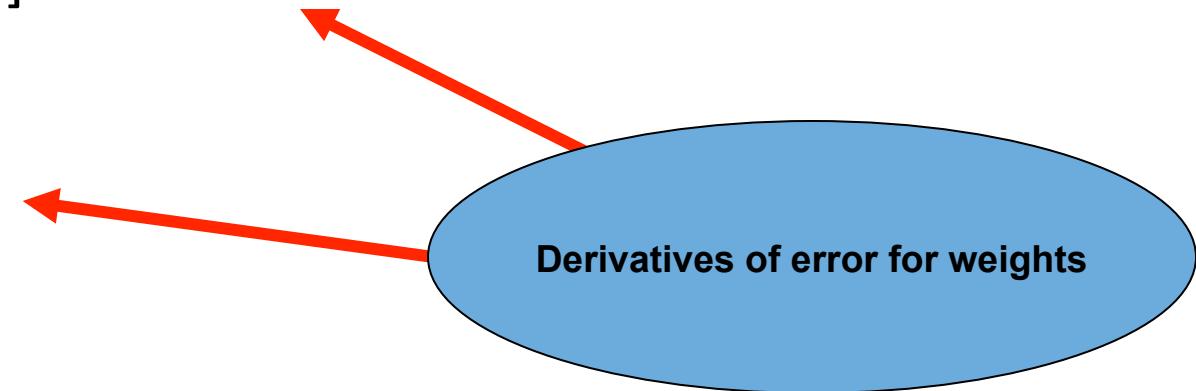


# Neural Networks Training: Backpropagation

Compute  
deltas

Gradient Descent

- Trying to make error decrease the fastest
- Compute:
  - $\text{Grad}_E = [dE/dw_1, dE/dw_2, \dots, dE/dw_n]$
- Change i-th weight by
  - $\text{delta}_{wi} = -\alpha * dE/dw_i$
- We need a derivative!
- Activation function must be continuous, differentiable, non-decreasing, and easy to compute



# Neural Networks Training: Backpropagation

## Updating Hidden-to-Output

- We have **teacher supplied** desired values

$$\begin{aligned}\delta_{wji} &= \alpha * (t_i - y_i) * g'(z_i) * a_j \\ &= \alpha * (t_i - y_i) * y_i * (1 - y_i) * a_j\end{aligned}$$

for sigmoid the derivative is,  $g'(x) = g(x) * (1 - g(x))$

Learning rate

miss

Here we have general formula with derivative, next we use for sigmoid

Derivative of activation function

# Neural Networks Training: Backpropagation

Updating interior weights

- Layer k units provide values to all layer k+1 units
  - “miss” is ***sum of misses*** from all units on k+1
  - $\text{miss}_j = \sum [ a_i(1-a_i) (t_i - a_i) w_{ji} ]$
  - weights coming into this unit are **adjusted based on their contribution**

$$\delta_{kj} = \alpha * l_k * a_j * (1 - a_j) * \text{miss}_j$$

Compute deltas

For layer k+1

# Making Choices

## Backpropagation

- Number of hidden layers – *empirically determined*
- Number of neurons in each hidden layer – *empirically determined*
- Activation functions
- Error/loss functions
- Gradient descent methods

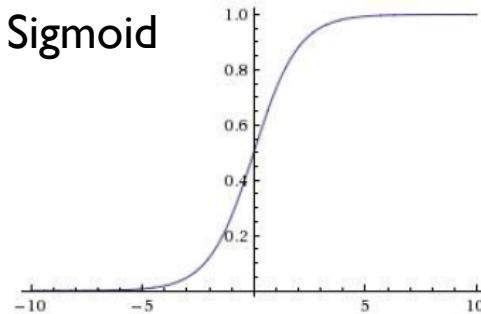
# Making Choices

## Backpropagation

- How do we pick  $\alpha$ ?
  - Small = slow, conservative learning; Large = can overshoot good solutions
  - Cross validation or adaptive methods (we'll see soon)
- How many hidden layers?
  - Too few ==> can't learn
  - Too many ==> poor generalization
- How big a training set?
  - Determine your target error rate,  $e$ ; Success rate is  $1 - e$
  - Typical training set approx.  $n/e$ , where  $n$  is the number of weights in the net
  - Example:
    - $e = 0.1$ ,  $n = 80$  weights; Training set size 800
  - Works for small datasets/networks

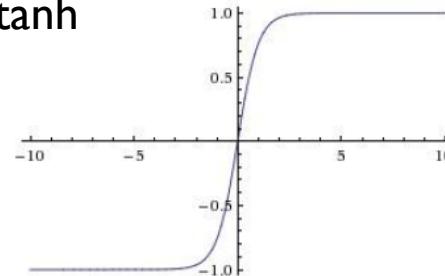
# Activation Functions

Sigmoid

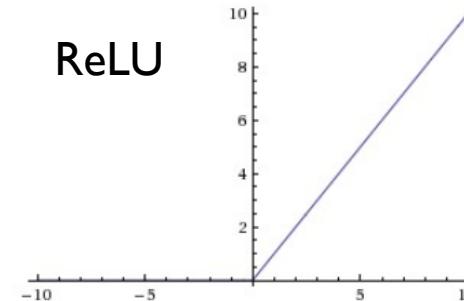


$$y = \frac{1}{1 + e^{-x}}$$

tanh

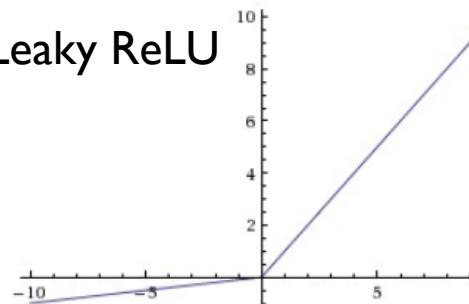


ReLU



$$y = \max(0, x)$$

Leaky ReLU



$$y = \begin{cases} x & \text{if } x < 0 \\ 0.01x & \text{if otherwise} \end{cases}$$

maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

# Loss Functions

- Euclidean loss / Squared loss  $L = \frac{1}{2} \|x_i - y_i\|_2^2$ 
  - Derivative w.r.t.  $x_i$   $\frac{\partial L}{\partial x_i} = x_i - y_i$
- Soft-max loss/multinomial logistic regression loss
  - Derivative w.r.t.  $x_i$   $\frac{\partial L}{\partial x_i} = p_i - y_i$
  - Also called: Cross-entropy loss

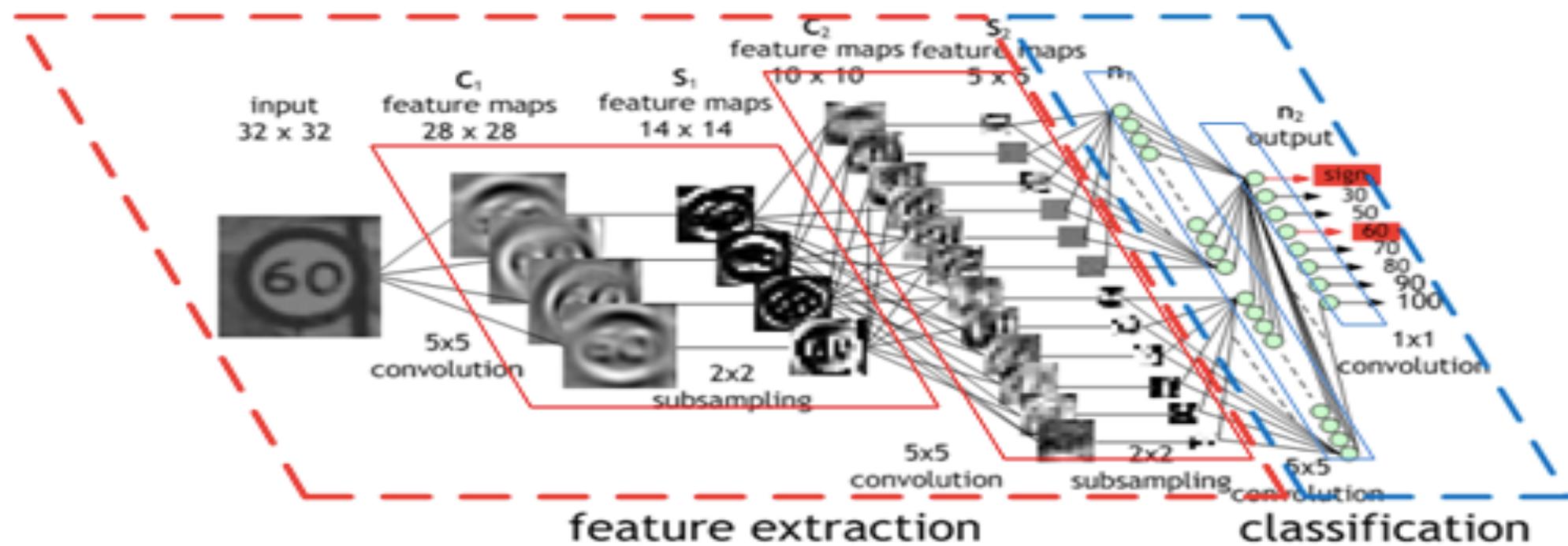
# Gradient Descent Methods

- Batch gradient descent (vs) Stochastic gradient descent (vs) Mini-batch stochastic gradient descent
  - Mini-batch SGD the most popularly used
- Using momentum
- Setting learning rate
  - Fixed learning rate
  - Using learning rate schedules
  - Adaptive learning rate methods: Adam, Adadelta, Adagrad, RMSProp

# Deep Learning Architectures

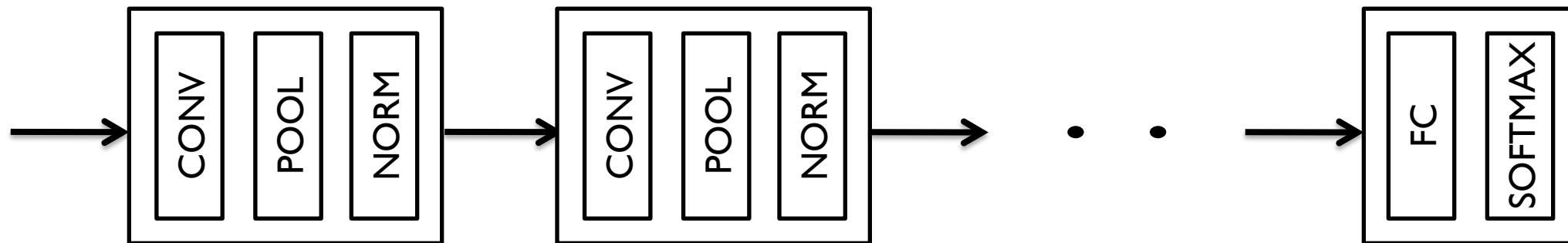
## Variants

### Convolutional Neural Networks for Image and Video Understanding



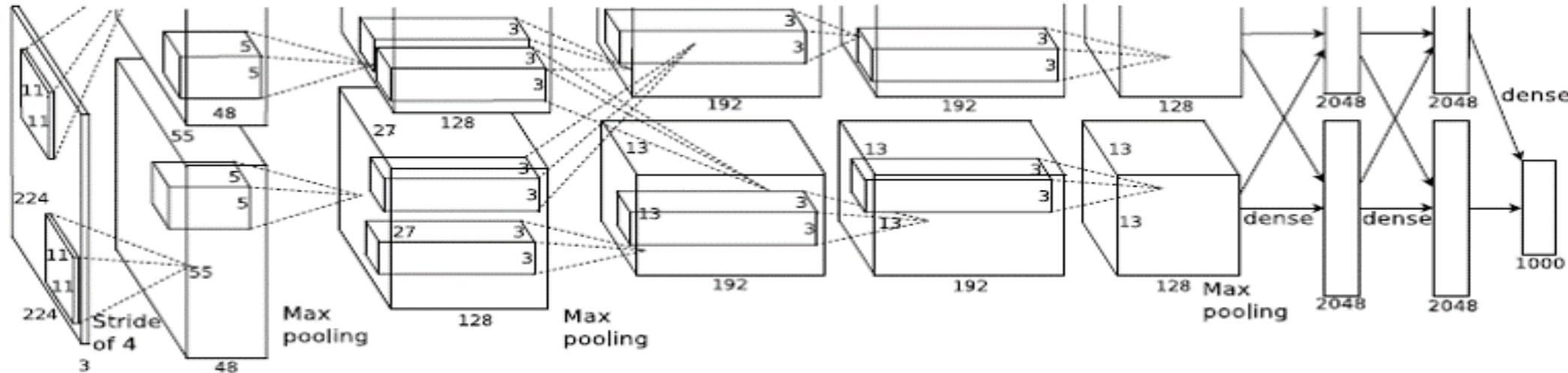
# CNNs: Typical Architecture

- A typical deep convolutional network



# Deep Learning in Computer Vision: The Turning Point

## AlexNet in the ImageNet Challenge



## ImageNet Classification with Deep Convolutional Neural Networks

**Alex Krizhevsky**  
University of Toronto  
kriz@cs.utoronto.ca

**Ilya Sutskever**  
University of Toronto  
ilya@cs.utoronto.ca

**Geoffrey E. Hinton**  
University of Toronto  
hinton@cs.utoronto.ca

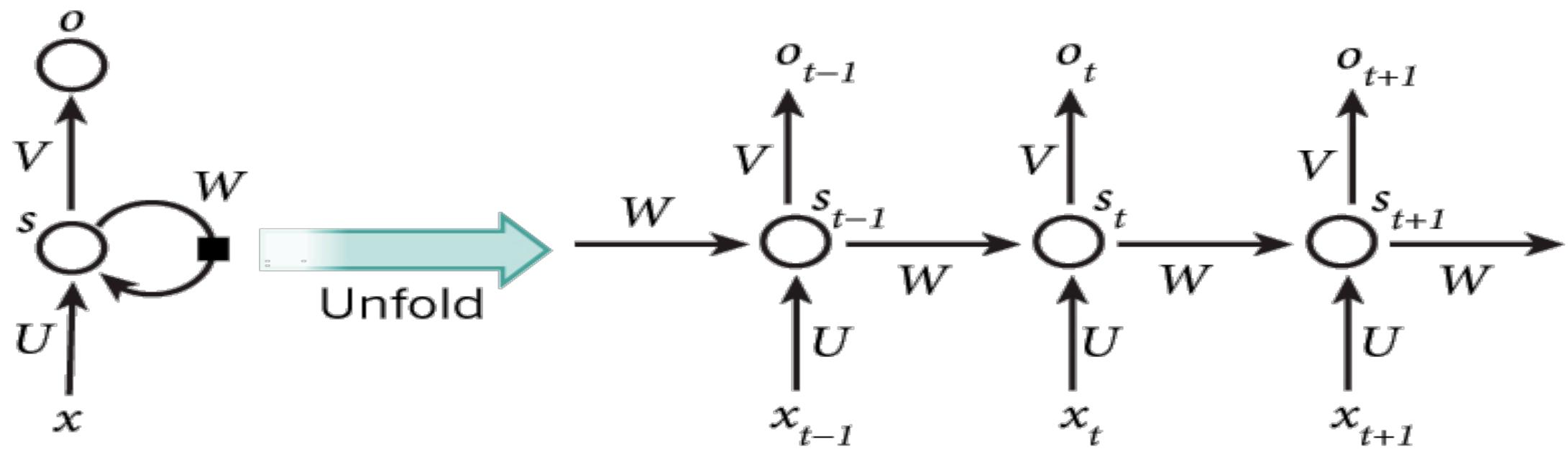
## ImageNet Classification Task:

*Previous Best: ~25% (CVPR-2011)*  
*AlexNet : ~15 % (NIPS-2012)*

# Deep Learning Architectures

## Variants

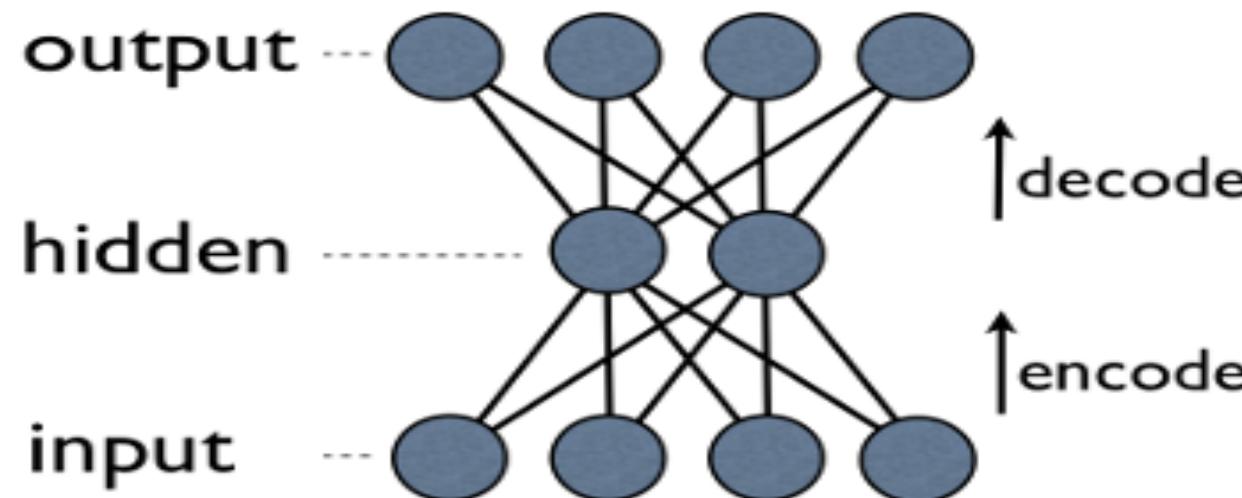
### Recurrent Neural Networks for Time Series and Sequence Data Understanding



# Deep Learning Architectures

## Variants

### Deep Autoencoders for Dimensionality Reduction



# Deep Learning

Why has use of multiple layers led to success?

- Captures compositionality: world is compositional!
  - **Image recognition:** Pixel → edge → texton → motif → part → object
  - **Text:** Character → word → word group → clause → sentence → story
  - **Speech:** Sample → spectral band → sound → ... → phone → phoneme → word
- Exploiting compositionality gives an exponential gain in representational power

# Deep Learning

## Applications and Successes

- AlexNet (Object Recognition): The network that catapulted the success of deep learning in 2012



# Deep Learning

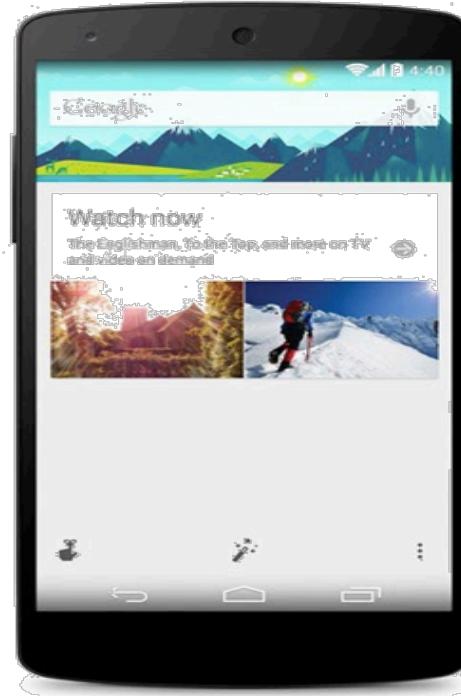
## Applications and Successes

- Speech understanding and natural language processing

Apple Siri



Google Now



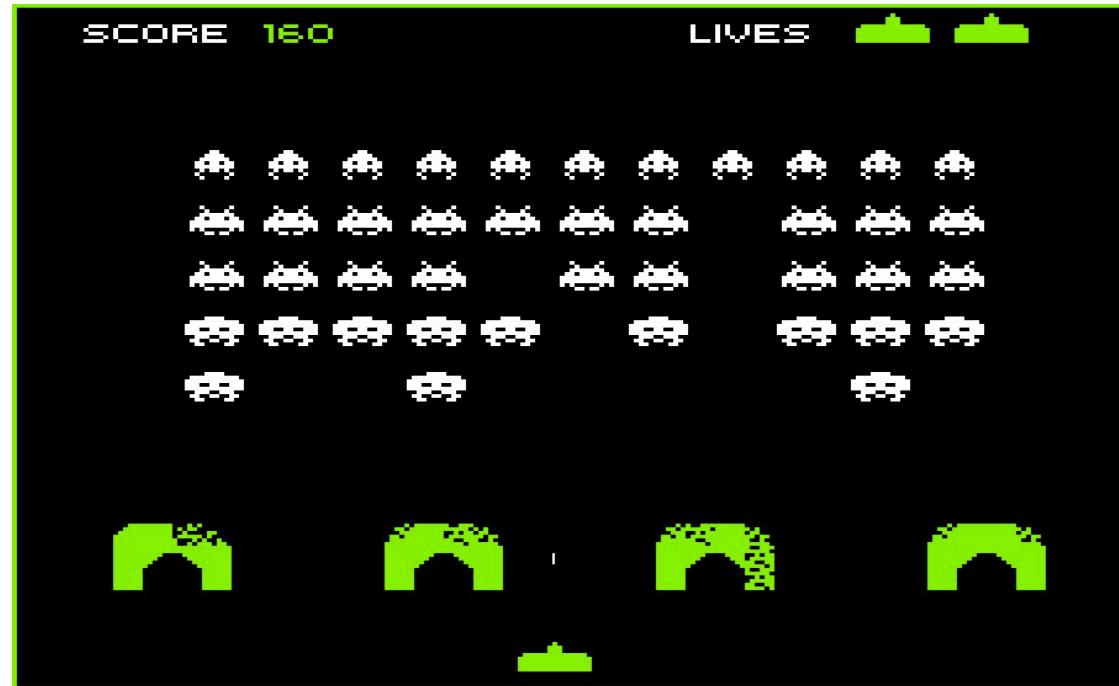
Windows Cortana



# Deep Learning

## Applications and Successes

- Game playing: <https://www.youtube.com/watch?v=V1eYniJ0Rnk>



# Stepping to higher levels of intelligence

## From recognition to generation

- Handwriting generation - <http://www.cs.toronto.edu/~graves/handwriting.html>
- RNN-generated TED talk (visit TED<sup>RNN</sup>)  
<https://www.youtube.com/watch?v=-OodHtJ1saY>
- RNN-generated Obama speech:
  - *Good afternoon. God bless you. The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. The promise of the men and women who were still going to take out the fact that the American people have fought to make sure that they have to be able to protect our part. It was a chance to stand together to completely look for the commitment to borrow from the American people. And the fact is the men and women in uniform and the millions of our country with the law system that we should be a strong stretches of the forces that we can afford to increase our spirit of the American people and the leadership of our country who are on the Internet of American lives. Thank you very much. God bless you, and God bless the United States of America.*

# Stepping to higher levels of intelligence

## From recognition to generation

- RNN trained on Shakespeare:

- <http://cs.stanford.edu/people/karpathy/char-rnn/shakespear.txt>
- <https://www.youtube.com/watch?v=Jkkjy7dVdaY>

VIOLA:

Why, Salisbury must find his flesh and thought  
That which I am not aps, not a man and in fire,  
To show the reining of the raven and the wars  
To grace my hand reproach within, and not a fair are hand,  
That Caesar and my goodly father's world;  
When I was heaven of presence and our fleets,  
We spare with hours, but cut thy council I am great,  
Murdered and by thy master's ready there  
My power to give thee but so much as hell:  
Some service in the noble bondman here,  
Would show him to her wine.

KING LEAR:

O, if you were a feeble sight, the courtesy of your law,  
Your sight and several breath, will wear the gods  
With his heads, and my hands are wonder'd at the deeds,  
So drop upon your lordship's head, and your opinion  
Shall be against your honour.

# Stepping to even higher levels of intelligence

From generation to imagination

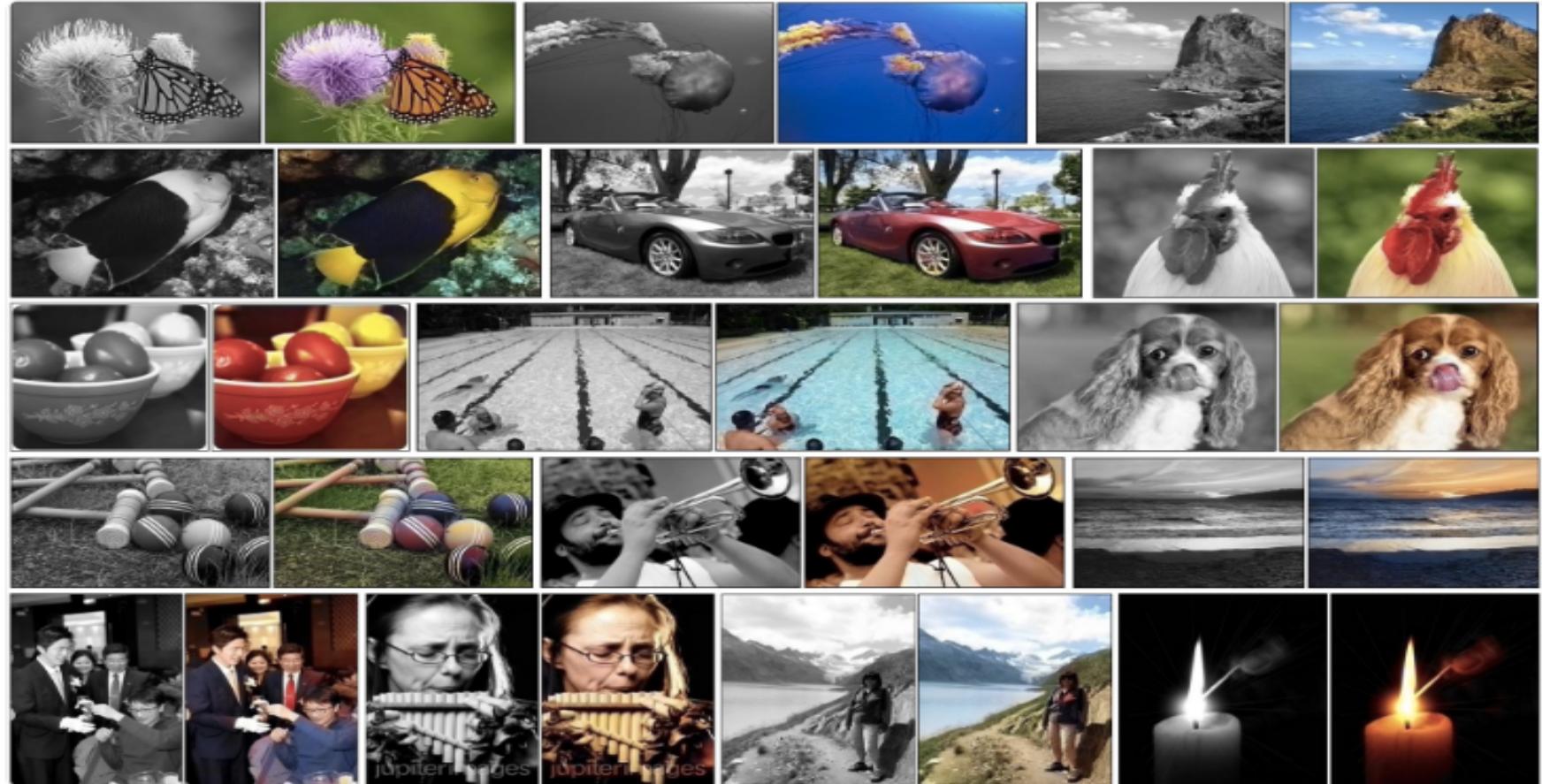
- Deep dreaming/hallucination
  - <https://www.youtube.com/watch?v=oyxSerkkP4o>



# Stepping to even higher levels of intelligence

From generation to imagination

- Deep colorization



# More?

## Where to look?

- One-stop shop
  - <https://github.com/ChristosChristofidis/awesome-deep-learning>
- Check this out for hours of fun and amazement
  - <http://fastml.com/deep-nets-generating-stuff/>
- Books (on Deep Learning)
  - <http://www.deeplearningbook.org>
  - <http://neuralnetworksanddeeplearning.com/>
- Programming
  - Tensorflow, PyTorch, Theano/Pylearn2, Caffe, Torch, Keras

# Readings

- “Introduction to Machine Learning” by Ethem Alpaydin, Chapters III.I-III.II