Neural Networks

Slide credits : Vineeth N Balasubramanian

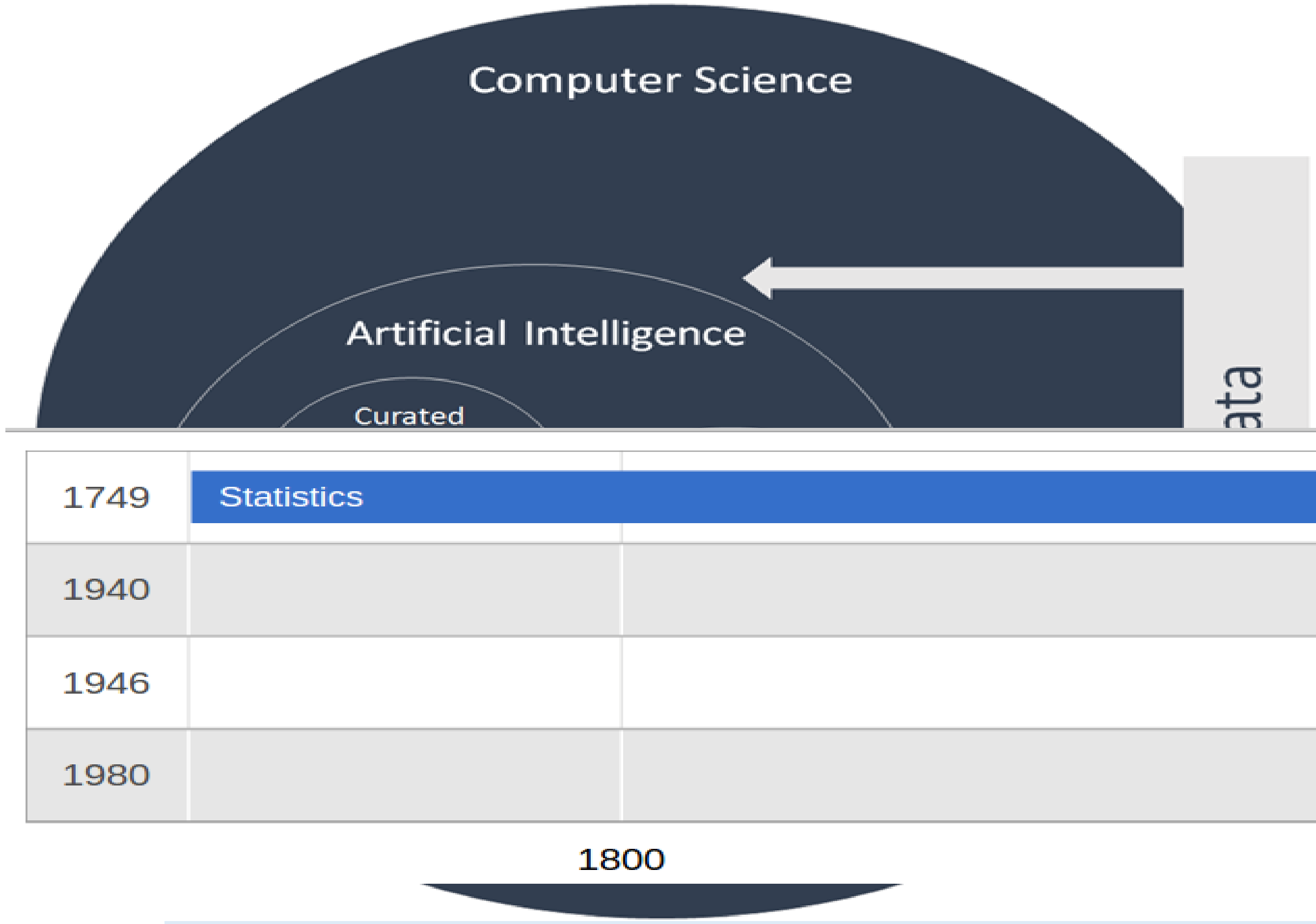
आई आई टी हैदराबाद
IIT Hyderabad

Classification Methods

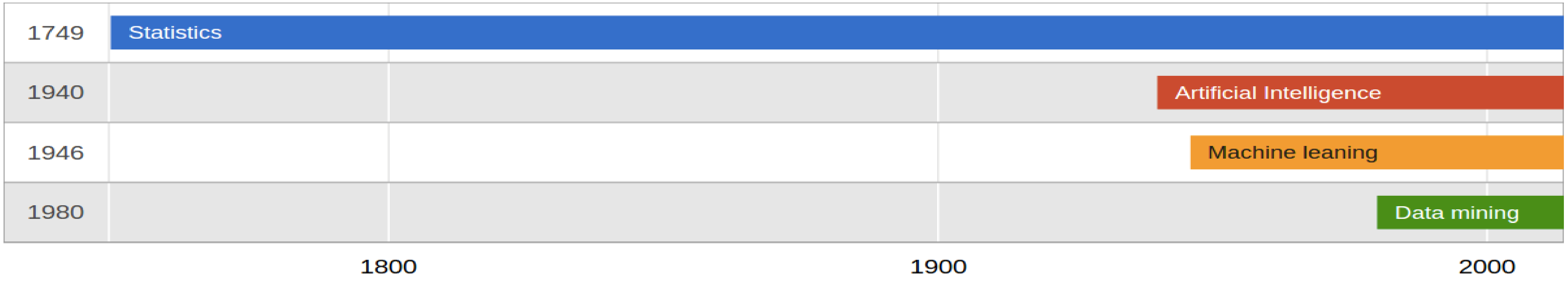
- k-Nearest Neighbors
- Decision Trees
- Naïve Bayes
- Support Vector Machines
- Logistic Regression
- Neural Networks (Deep Learning)
- Ensemble Methods (Boosting, Random Forests)

Neural Networks (aka) Deep Learning

Introduction



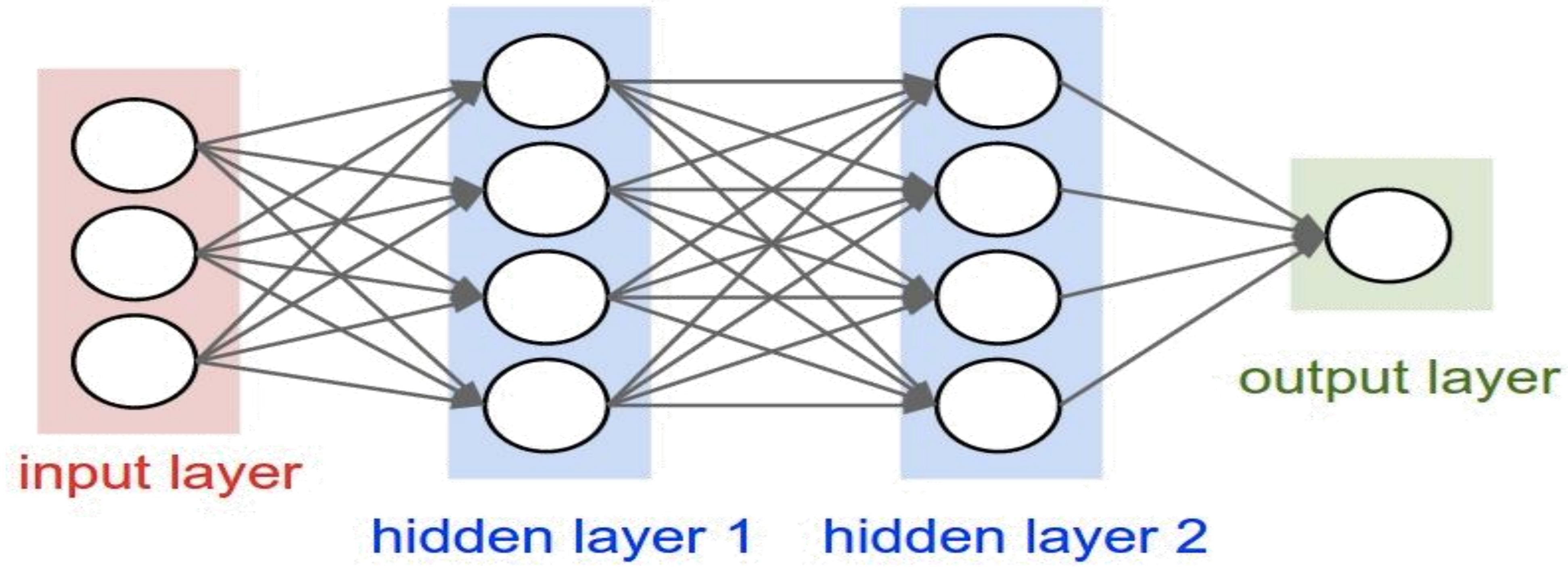
Deep learning: A sub-area of machine learning, that is today understood as representation learning



Deep Learning

Introduction

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



History of Deep Learning

Golden Age | *"Dark Age"* | *"Dark Age"* | *Golden Age*

Electronic Brain 1943

Perceptron 1957

Delta Learning Rule 1960

XOR Problem 1969

Neocognitron 1980

Backpropagation 1986

SVMs 1996

Convolutional Neural Nets 1998

Hierarchical Feature Learning 2006

AlexNet 2012

1940 1950 1960 1970 1980 1990 2000 2010

McCulloch-Pitts

Rosenblatt

Widrow-Hoff

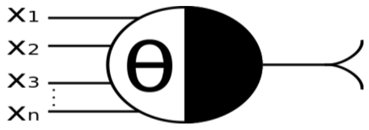
Minsky-Papert

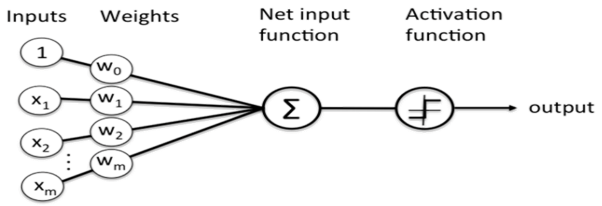
Rumelhart-Hinton-Williams

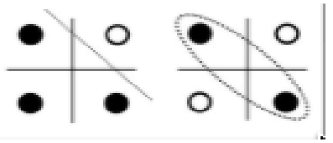
LeCun

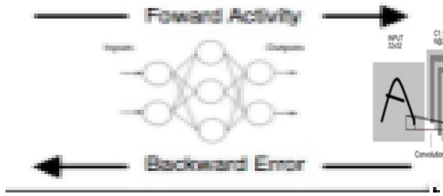
Hinton-Ruslan

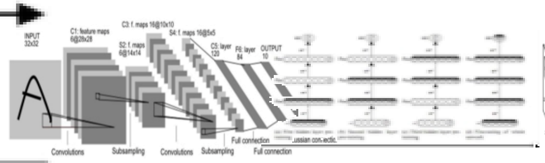
Krizhevsky-Sutskever-Hinton

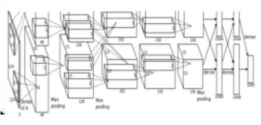






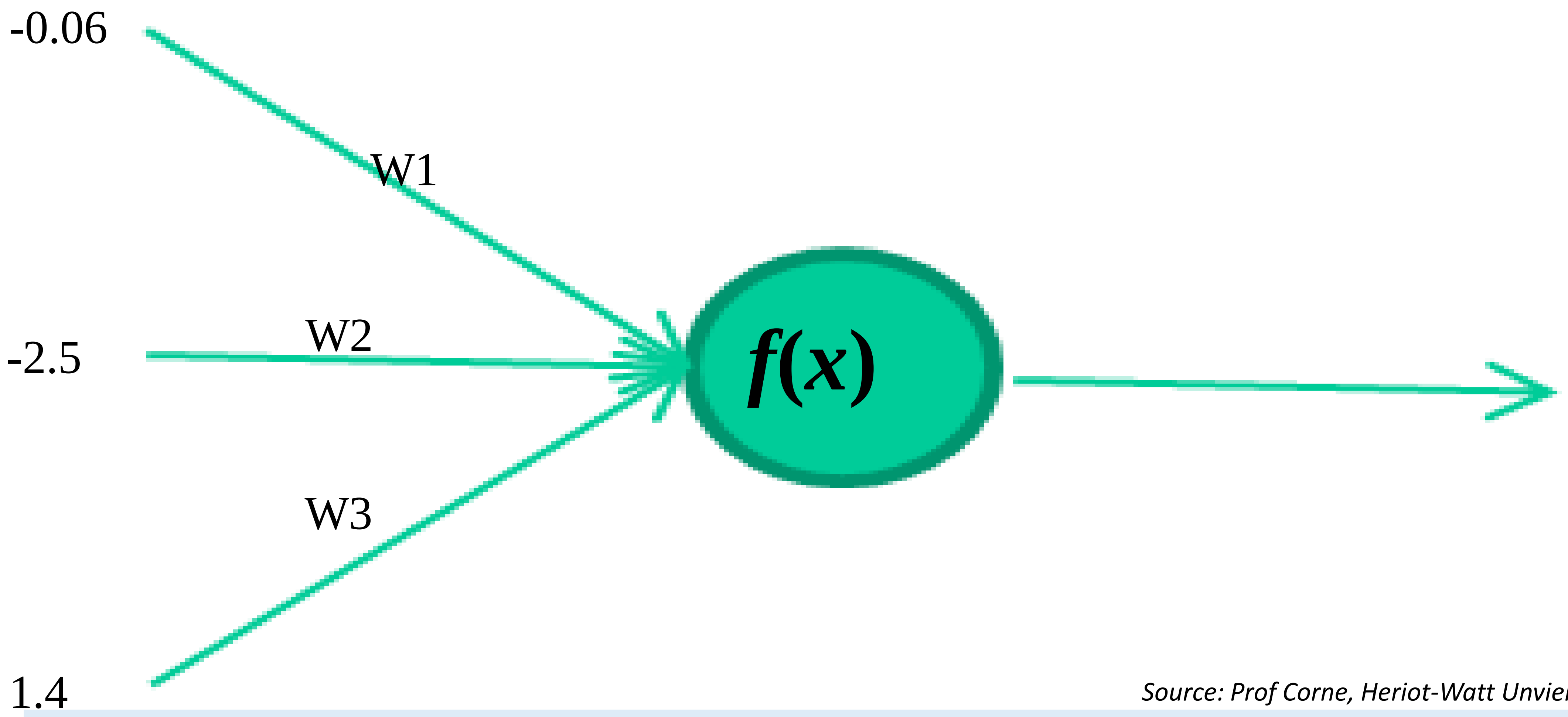






Deep Learning

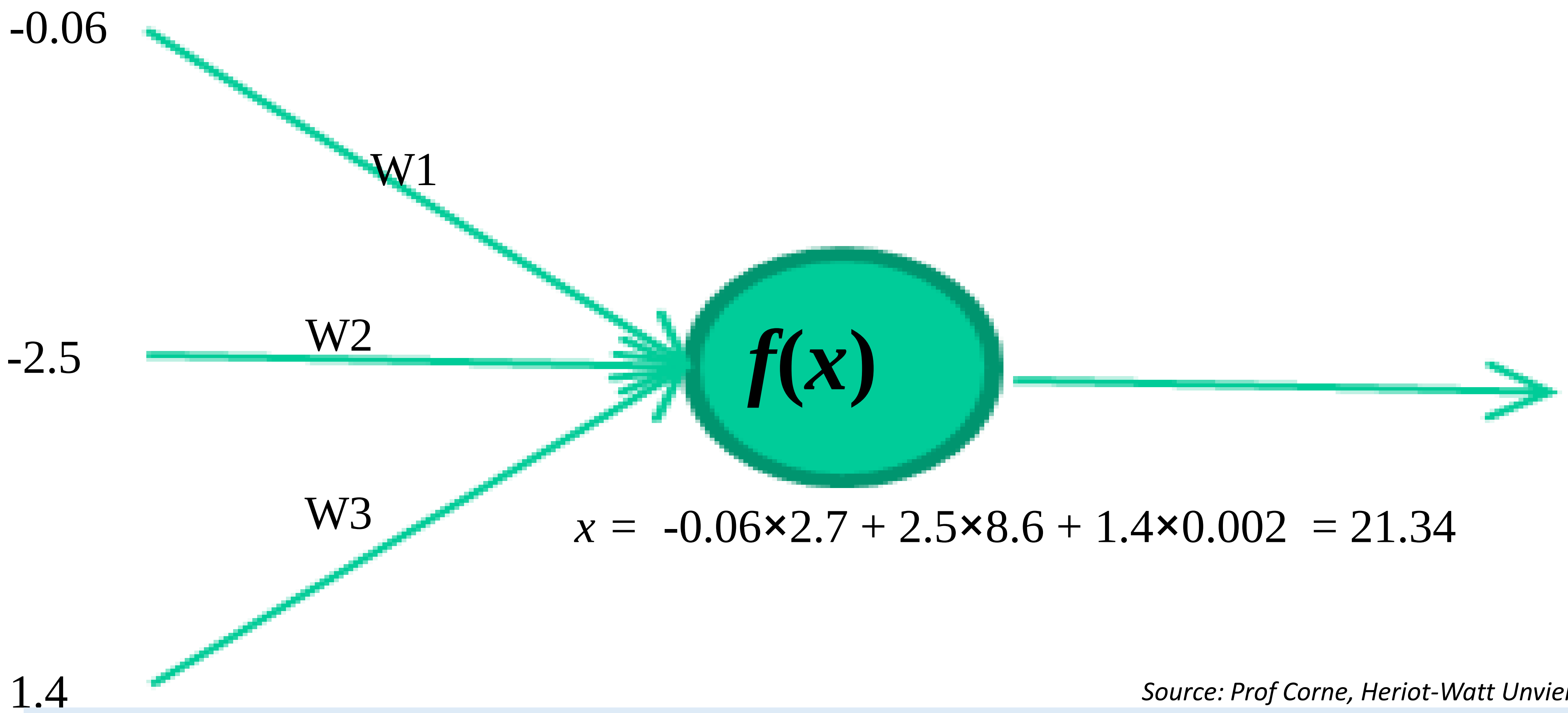
How do they learn?



Source: Prof Corne, Heriot-Watt Unviersity, UK

Deep Learning

How do they learn?



$x = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 = 21.34$

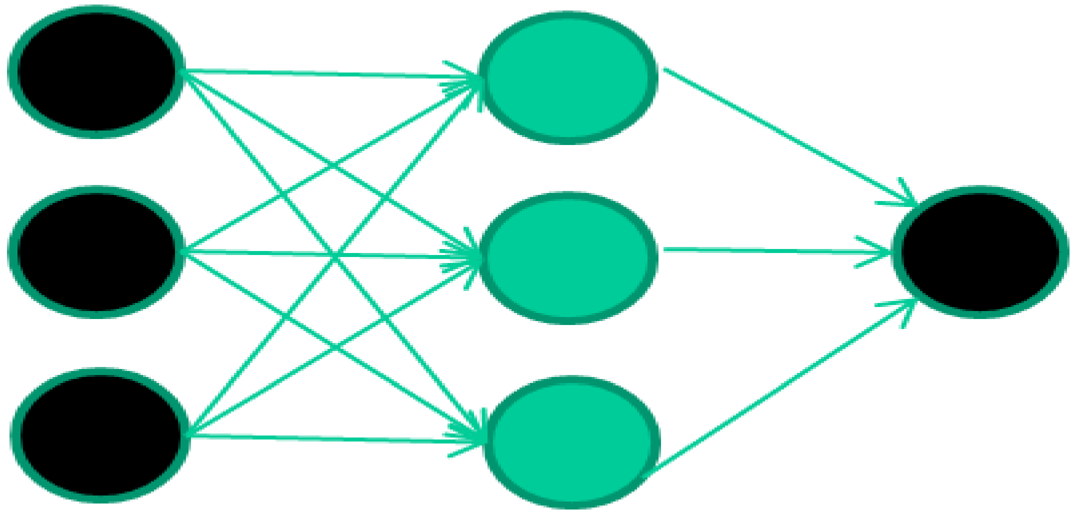
Source: Prof Corne, Heriot-Watt Unviersity, UK

Deep Learning

How do they learn?

A dataset

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 …			



Source: Prof Corne, Heriot-Watt Unviersity, UK

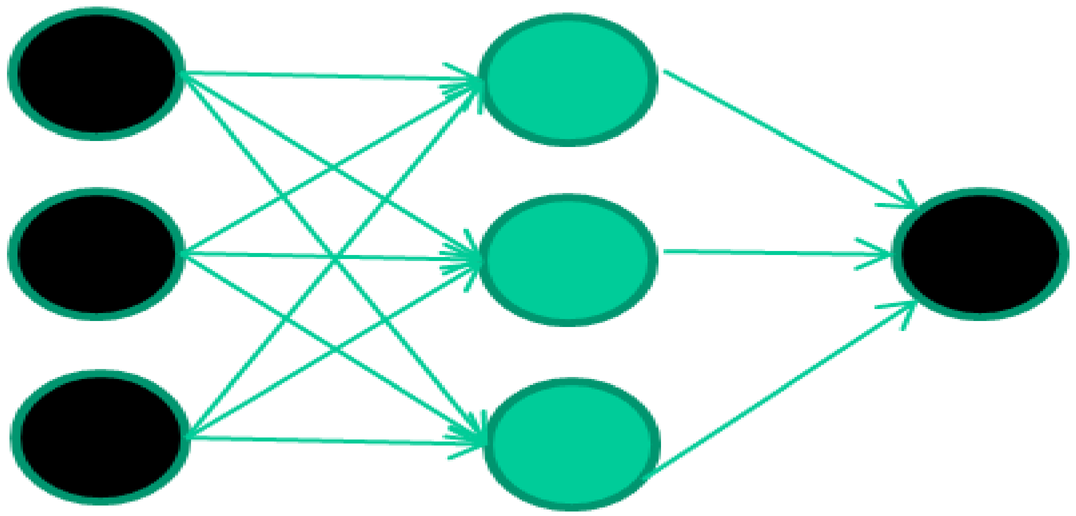
Deep Learning

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 …			

Initialise with random weights



Source: Prof Corne, Heriot-Watt Unviersity, UK

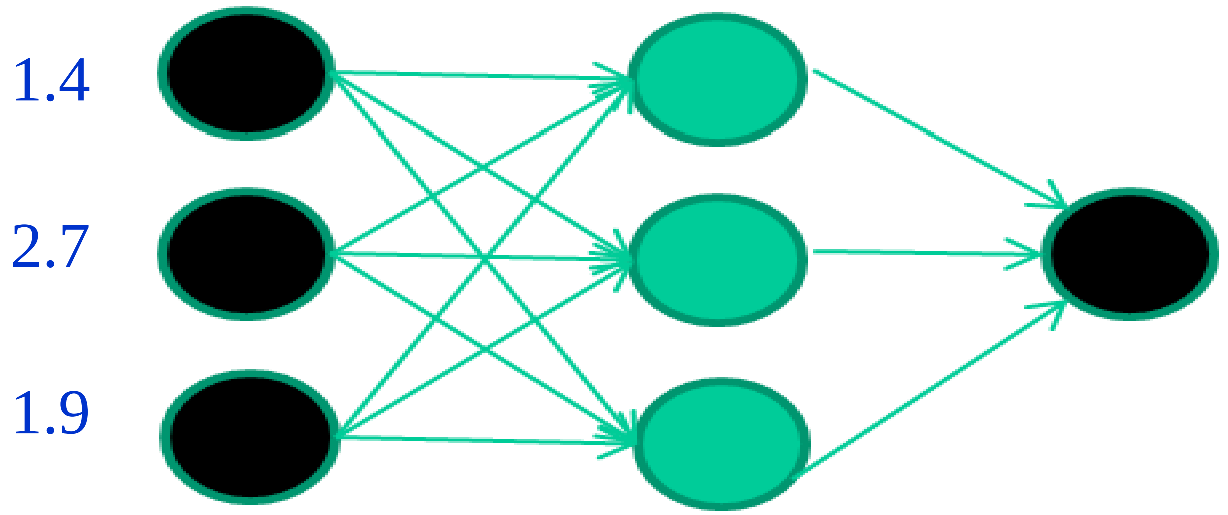
Deep Learning

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 Unviersity, UK

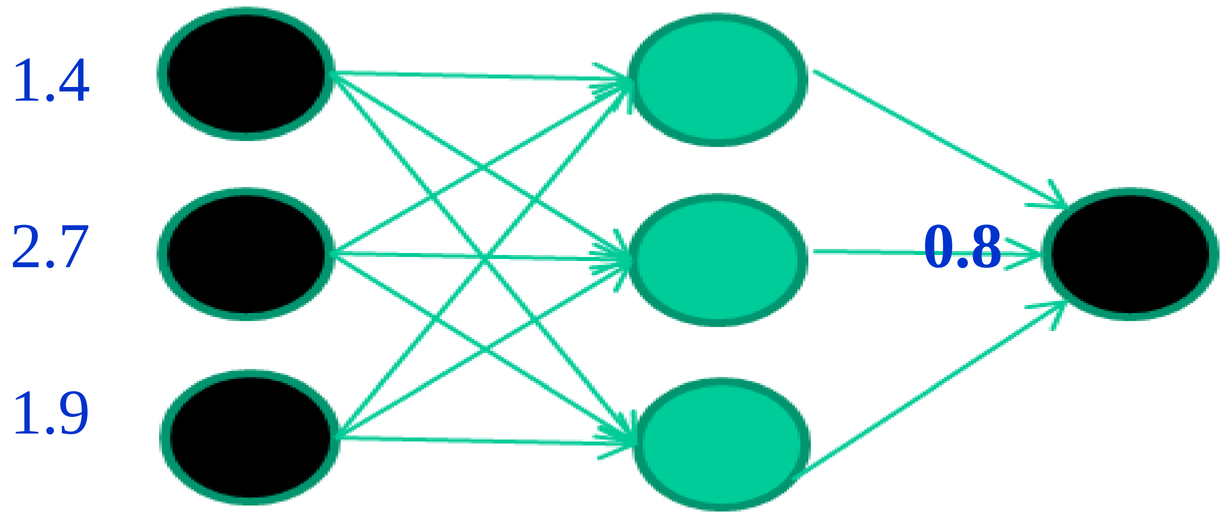
Deep Learning

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 Unviersity, UK

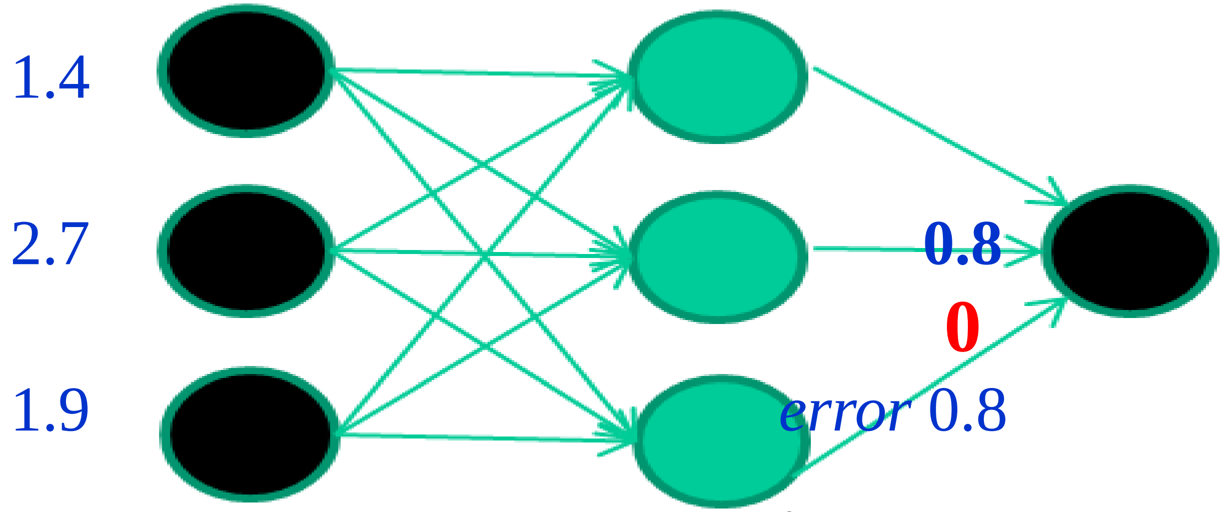
Deep Learning

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 Unviersity, UK

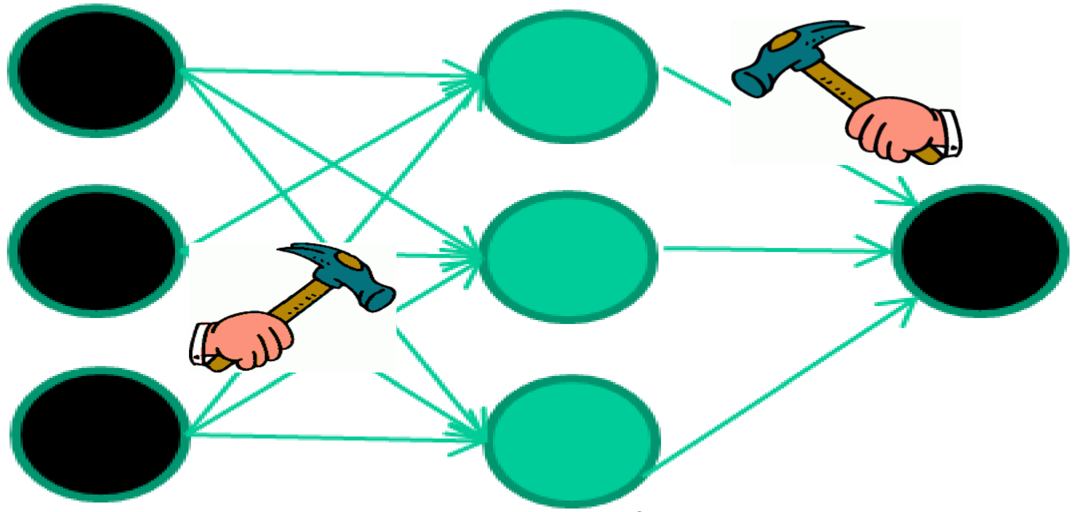
Deep Learning

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 Unviersity, UK

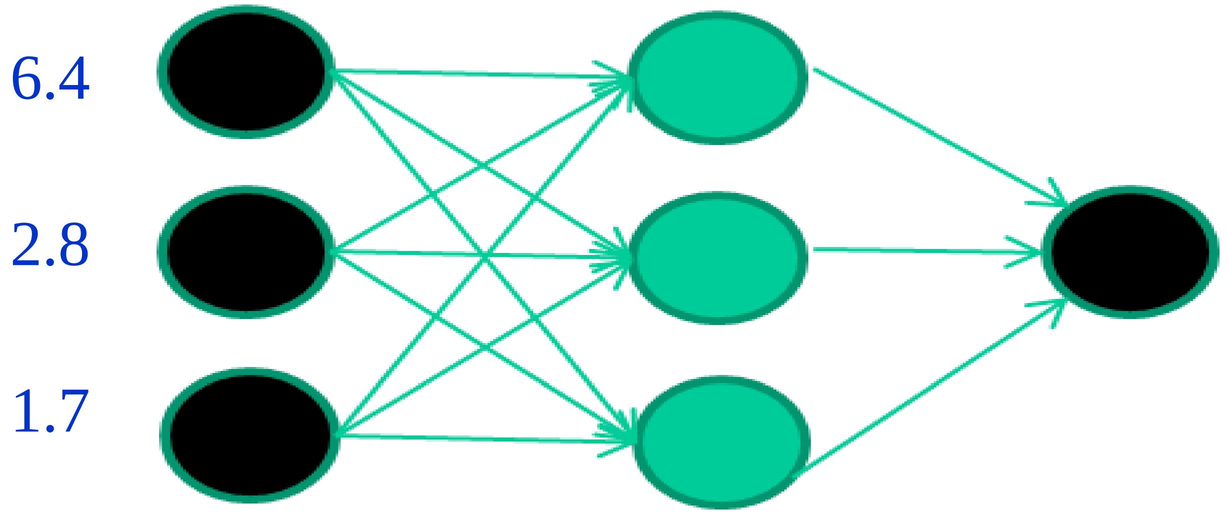
Deep Learning

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 Unviersity, UK

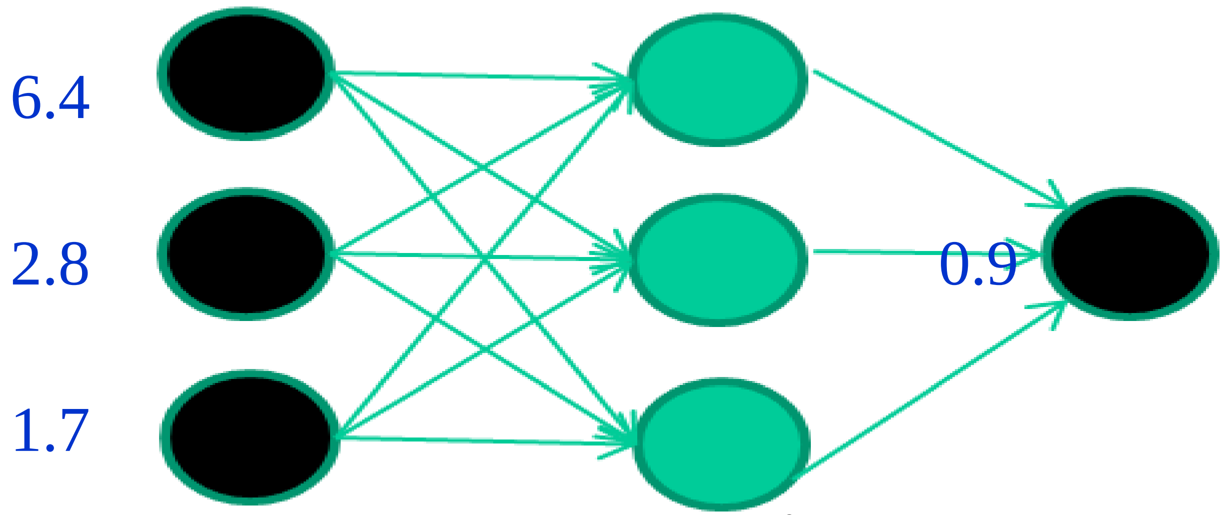
Deep Learning

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 Unviersity, UK

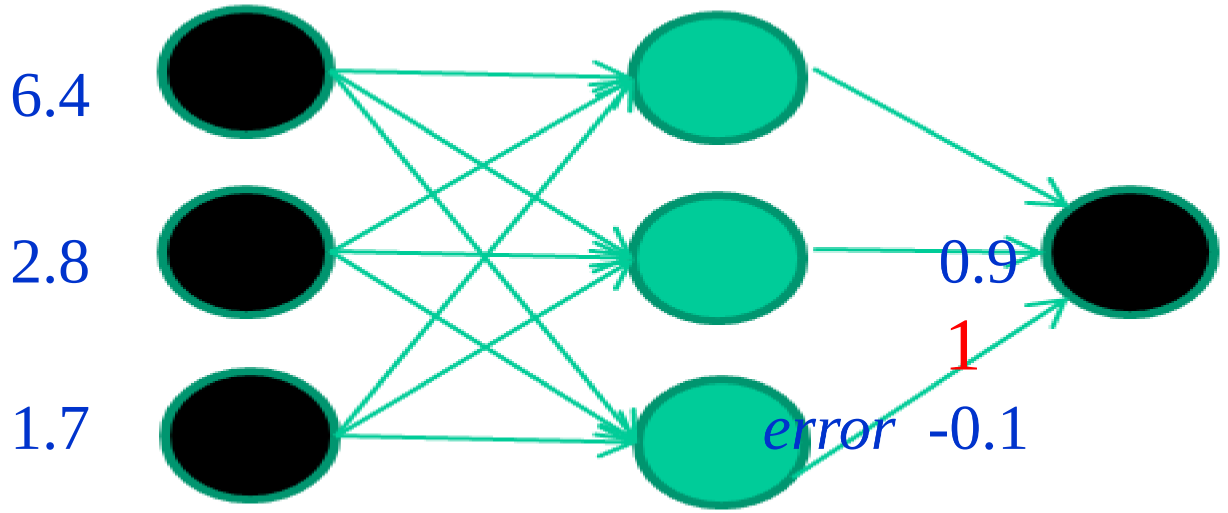
Deep Learning

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 Unviersity, UK

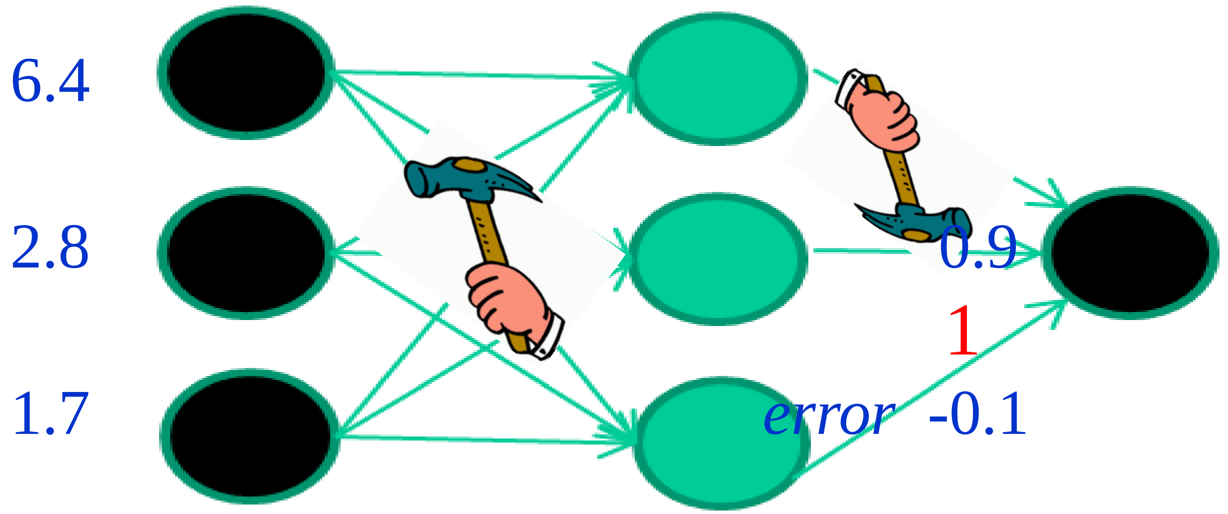
Deep Learning

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 Unviersity, UK

Deep Learning

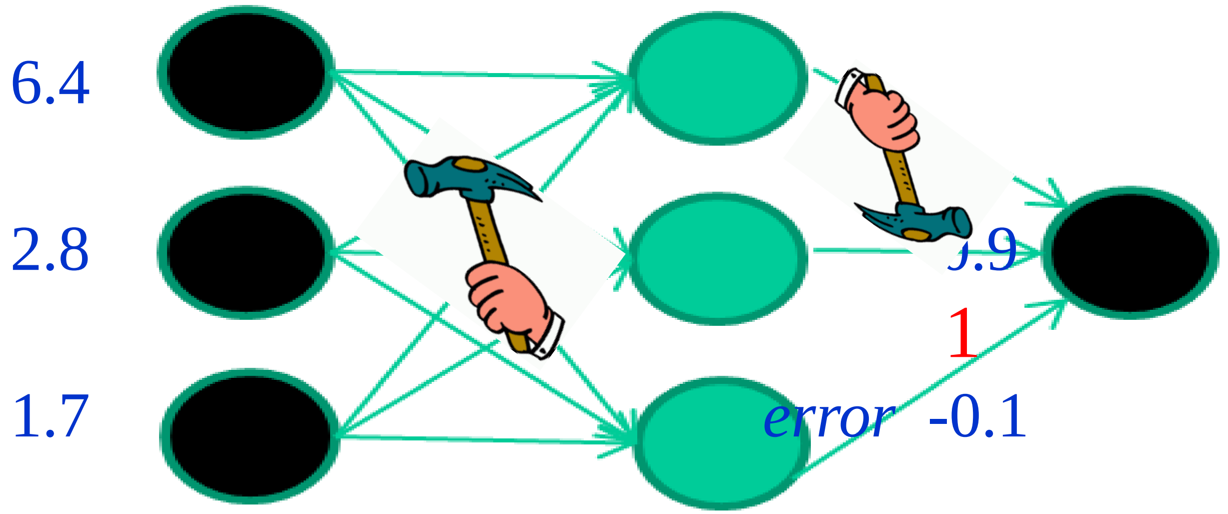
How do they learn?

Source: Prof Corne, Heriot-Watt Unviersity, UK

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 …			

And so on ….

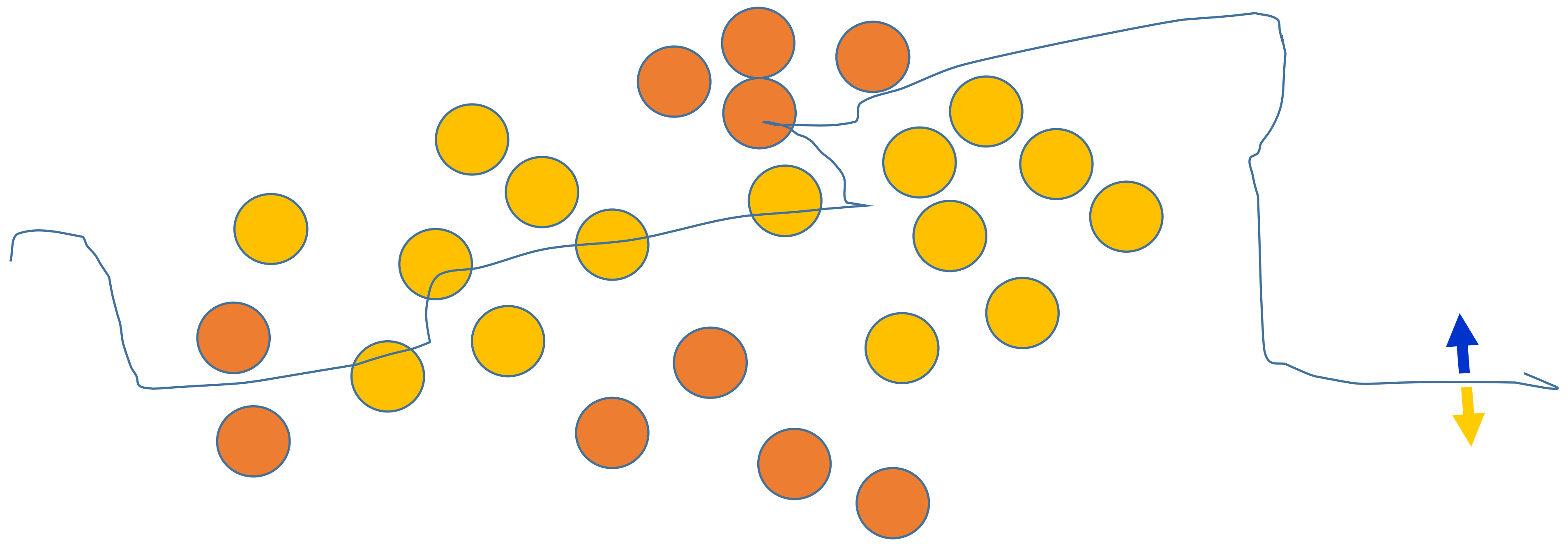


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments, reduce the error

Called "Gradient Descent"

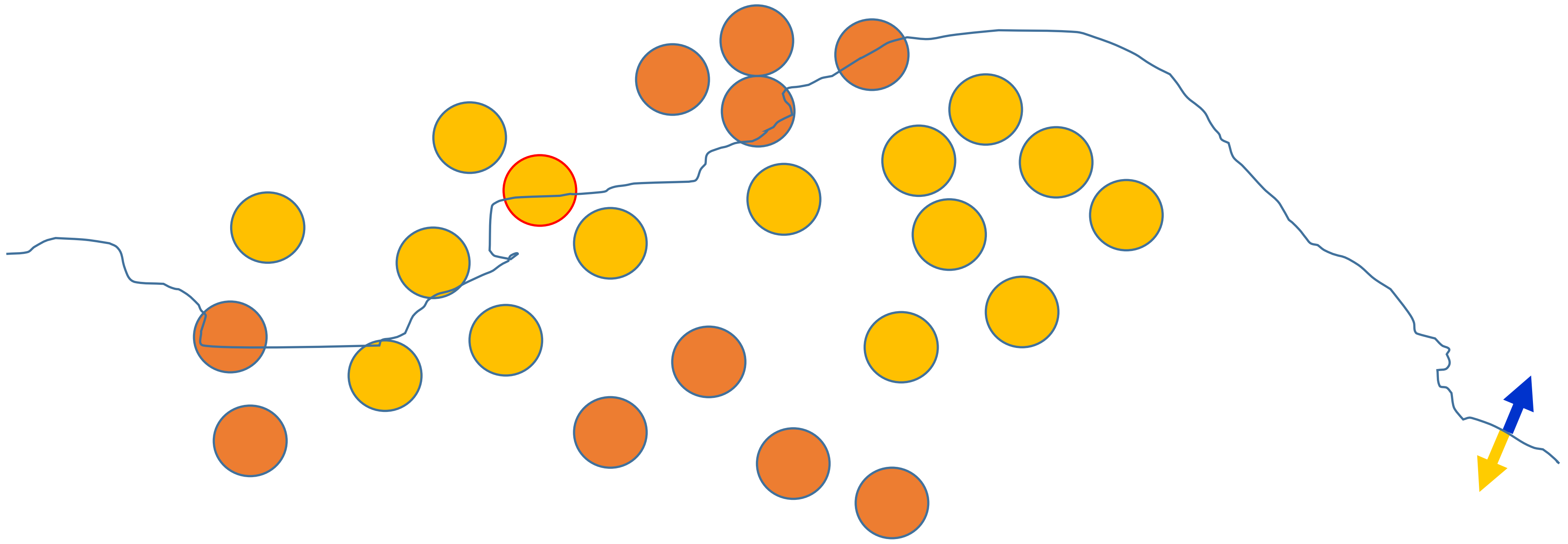
The decision boundary perspective…

Initial random weights



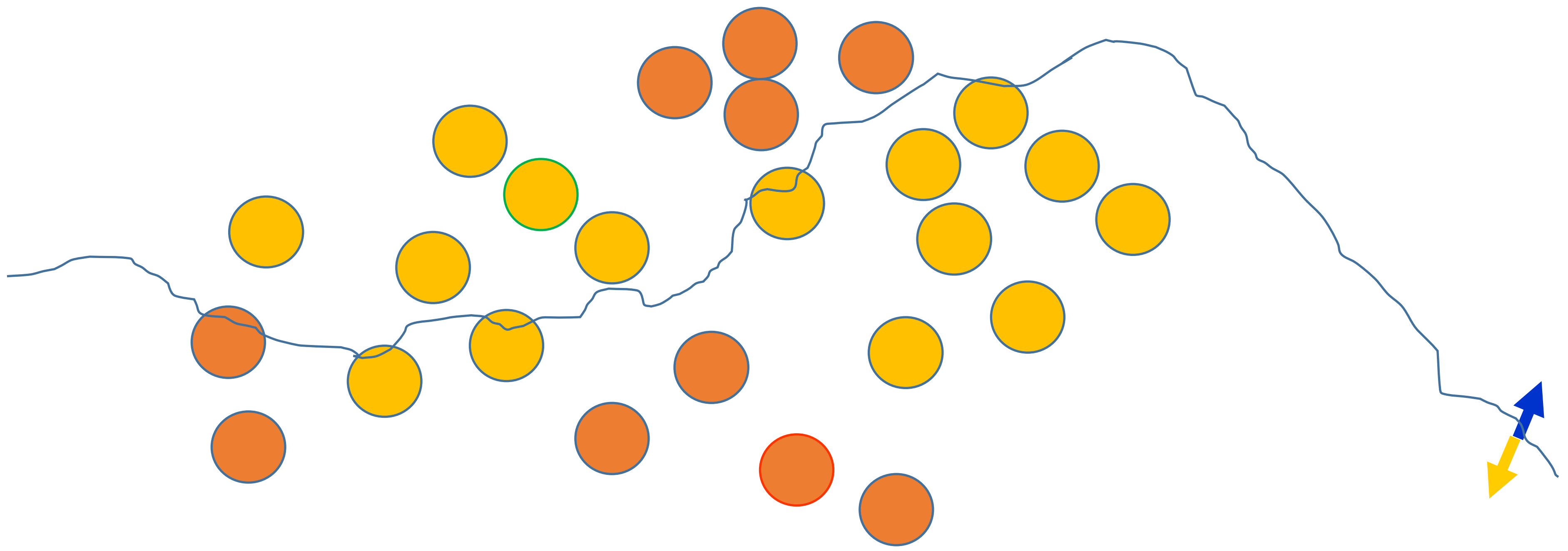
The decision boundary perspective…

Present a training instance / adjust the weights



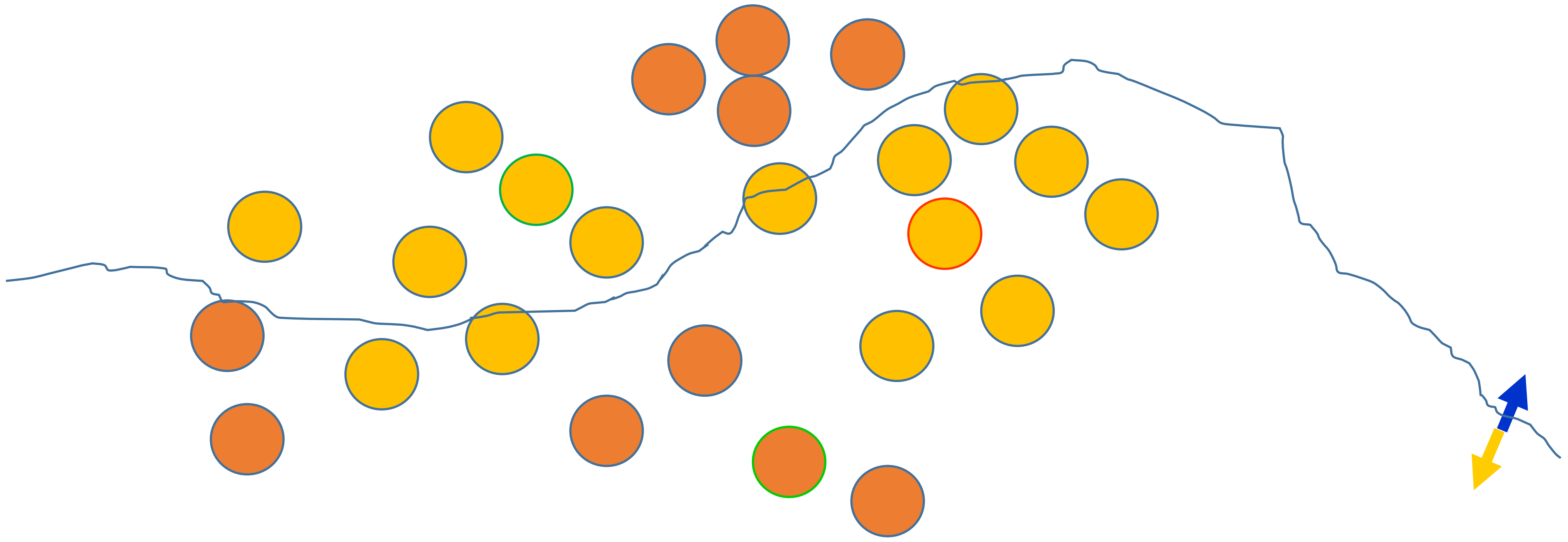
The decision boundary perspective…

Present a training instance / adjust the weights



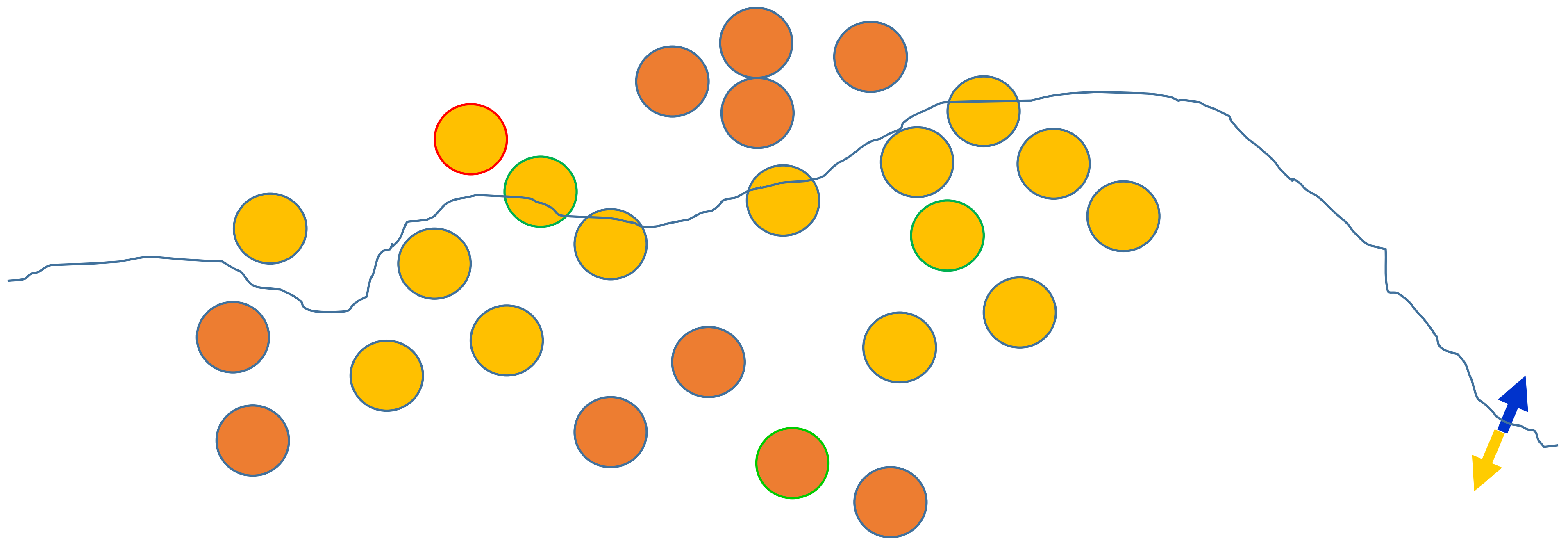
The decision boundary perspective…

Present a training instance / adjust the weights



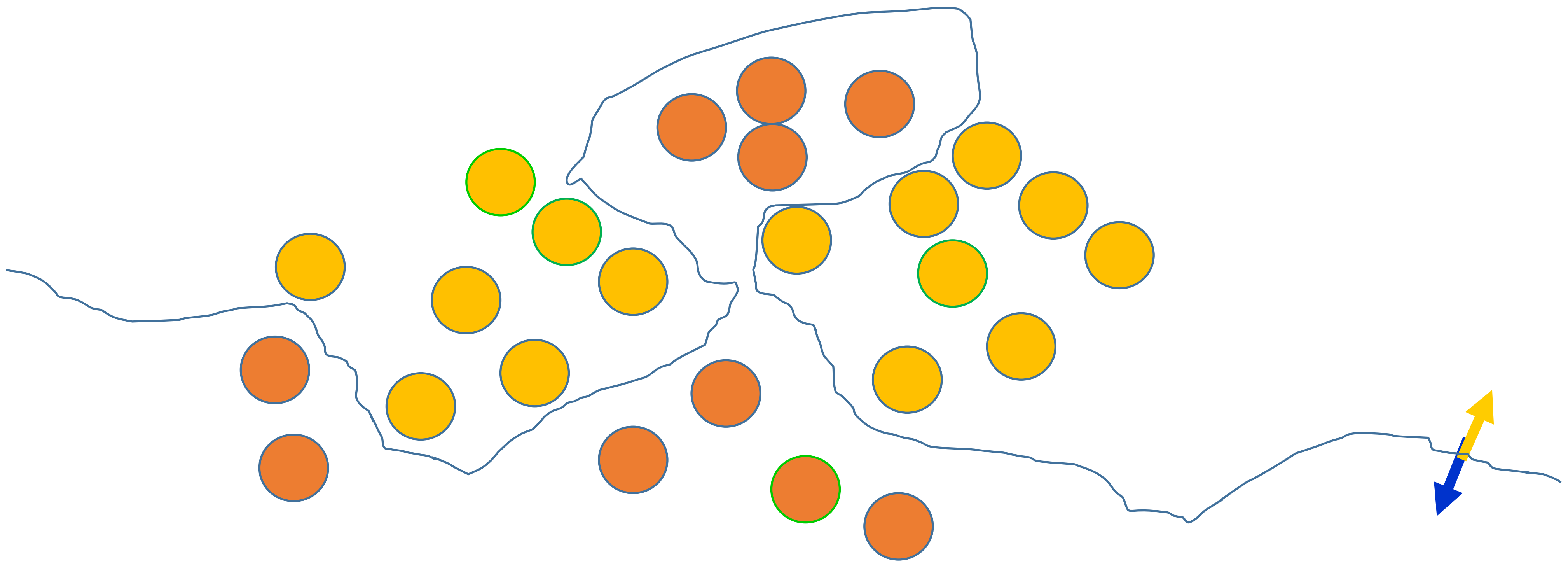
The decision boundary perspective…

Present a training instance / adjust the weights



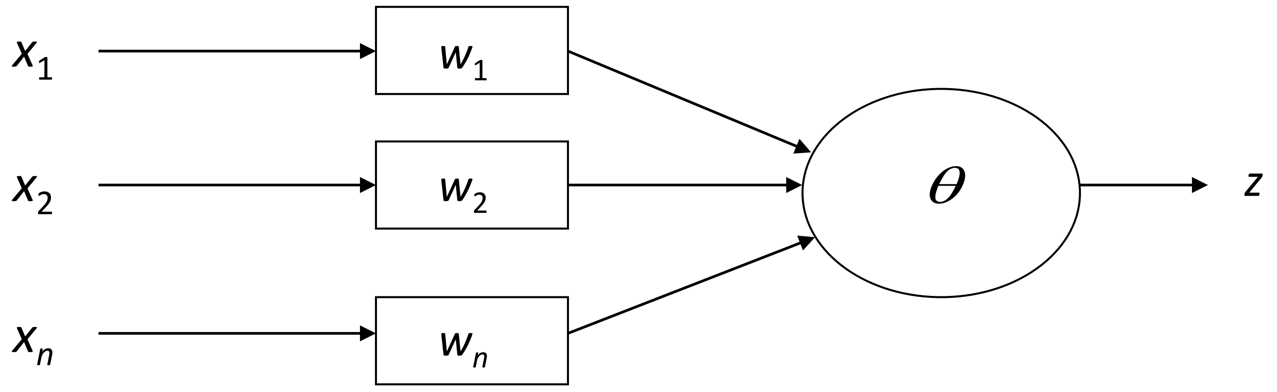
The decision boundary perspective…

Eventually ….



Neural Networks Training: Backpropagation

Perceptrons

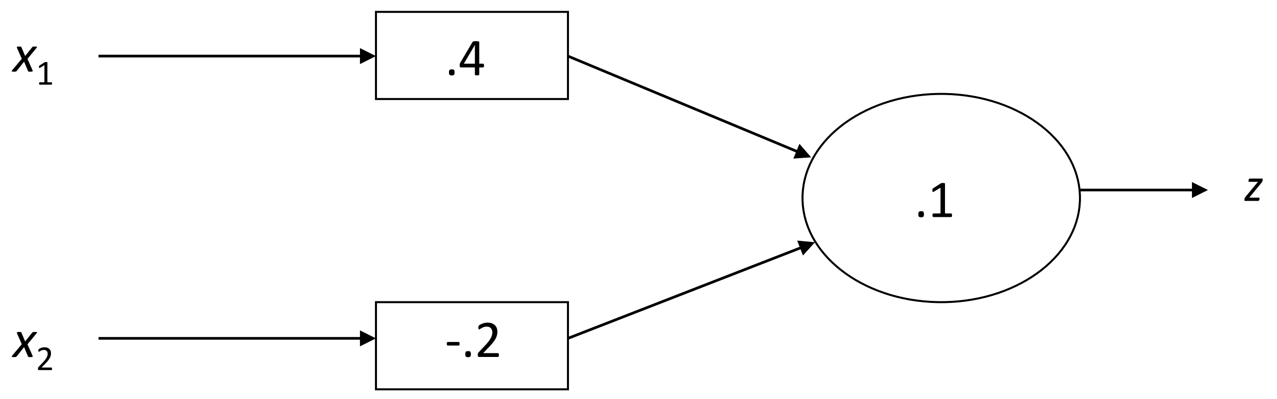


$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

- 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

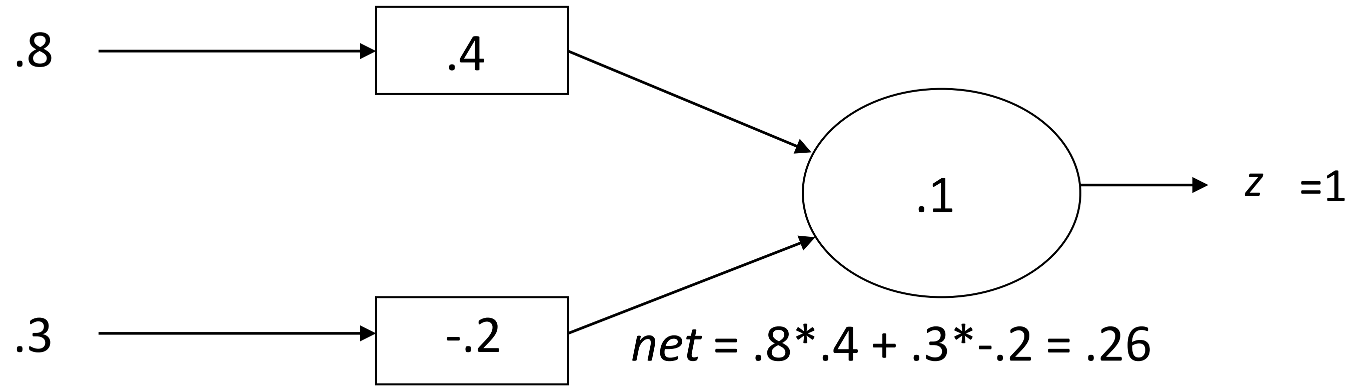


x_1	x_2	t
.8	.3	1
.4	.1	0

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

Neural Networks Training: Backpropagation

First Training Instance



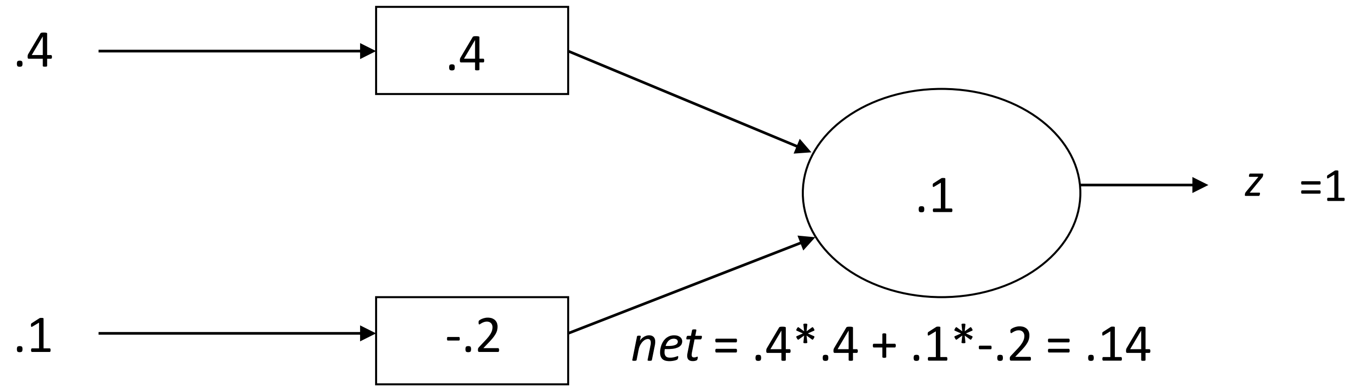
net = .8*.4 + .3*-.2 = .26

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



net = .4*.4 + .1*-.2 = .14

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*th input
- Least perturbation principle
  - Only change weights if there is an error
  - small *c* 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

- 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

- 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

- 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

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

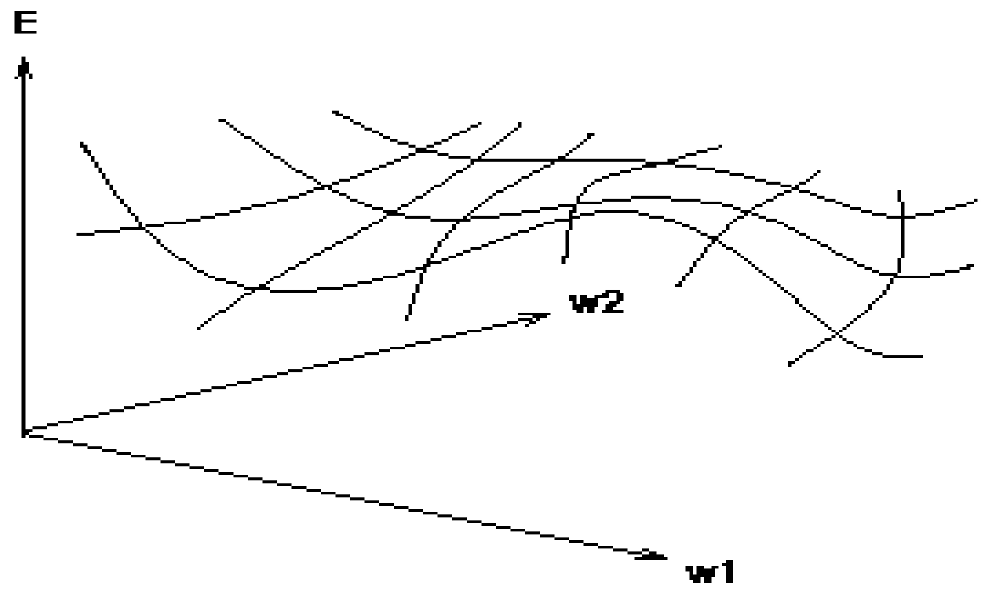
Forward pass

Backward pass explained in next few slides

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"



Compute deltas

Neural Networks Training: Backpropagation

Gradient Descent

- Trying to make error decrease the fastest
- Compute:
  - Grad_E = [dE/dw1, dE/dw2, . . ., dE/dwn]
- Change i-th weight by
  - delta_{wi} = -*alpha* * dE/dwi

Derivatives of error for weights

- We need a derivative!
- Activation function must be continuous, differentiable, non-decreasing, and easy to compute

Neural Networks Training: Backpropagation

Updating Hidden-to-Output

$$\min_{\mathbf{W},v} \sum_n \frac{1}{2}\left(y_n - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x}_n)\right)^2$$

$$\nabla_v = -\sum_n e_n \boldsymbol{h}_n$$

Neural Networks Training: Backpropagation

Updating Hidden

$$\mathcal{L}(\mathbf{W}) = \frac{1}{2}\left(y - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x})\right)^2$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i} = \frac{\partial \mathcal{L}}{\partial f_i}\frac{\partial f_i}{\partial \boldsymbol{w}_i}$$

$$\frac{\partial \mathcal{L}}{\partial f_i} = -\left(y - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x})\right) v_i = -e v_i$$

$$\frac{\partial f_i}{\partial \boldsymbol{w}_i} = f'(\boldsymbol{w}_i \cdot \boldsymbol{x})\boldsymbol{x}$$

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

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

$$\nabla_{\boldsymbol{w}_i} = -e v_i f'(\boldsymbol{w}_i \cdot \boldsymbol{x})\boldsymbol{x}$$

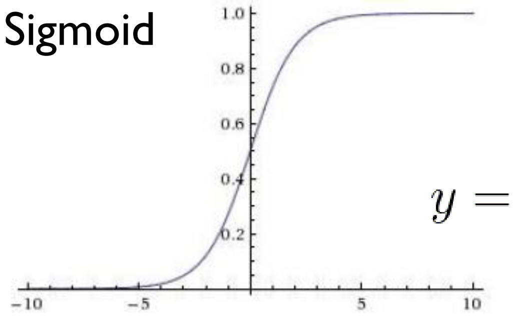
Derivative of activation function

Making Choices

- Number of hidden layers – *empirically determined*
  - Too few ==> can’t learn
  - Too many ==> poor generalization
- Number of neurons in each hidden layer – *empirically determined*
- Activation functions
- Error/loss functions
- Learning rate
- Gradient descent methods

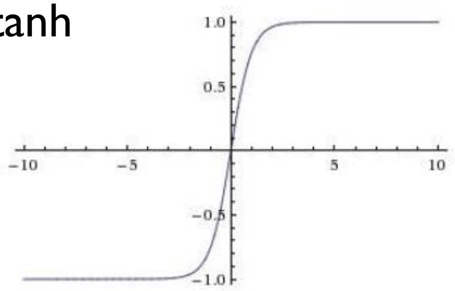
Activation Functions

Sigmoid



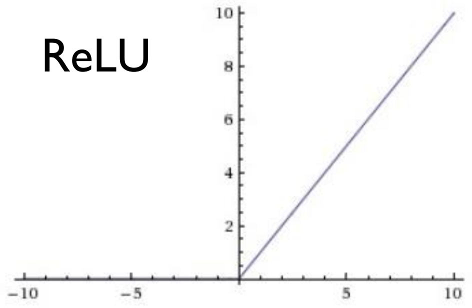
$$y = \frac{1}{1+e^{-x}}$$

tanh



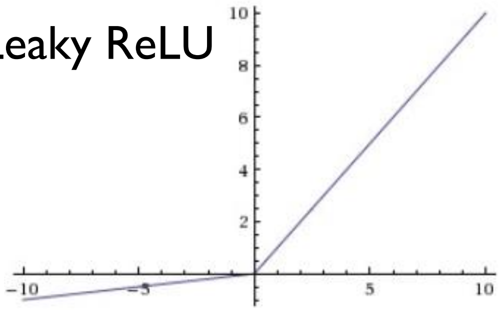
$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLU



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

Leaky ReLU



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

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

$$p_i = \frac{e^{x_i}}{\sum_k e^{x_k}} \qquad L = -\sum_i y_i log(p_i)$$

  - Derivative w.r.t. x_i $$\frac{\partial L}{\partial x_i} = p_i - y_i$$
  - Also called: Cross-entropy loss

neural networks loss function is non-convex and optimization is sensitive to their initialization.

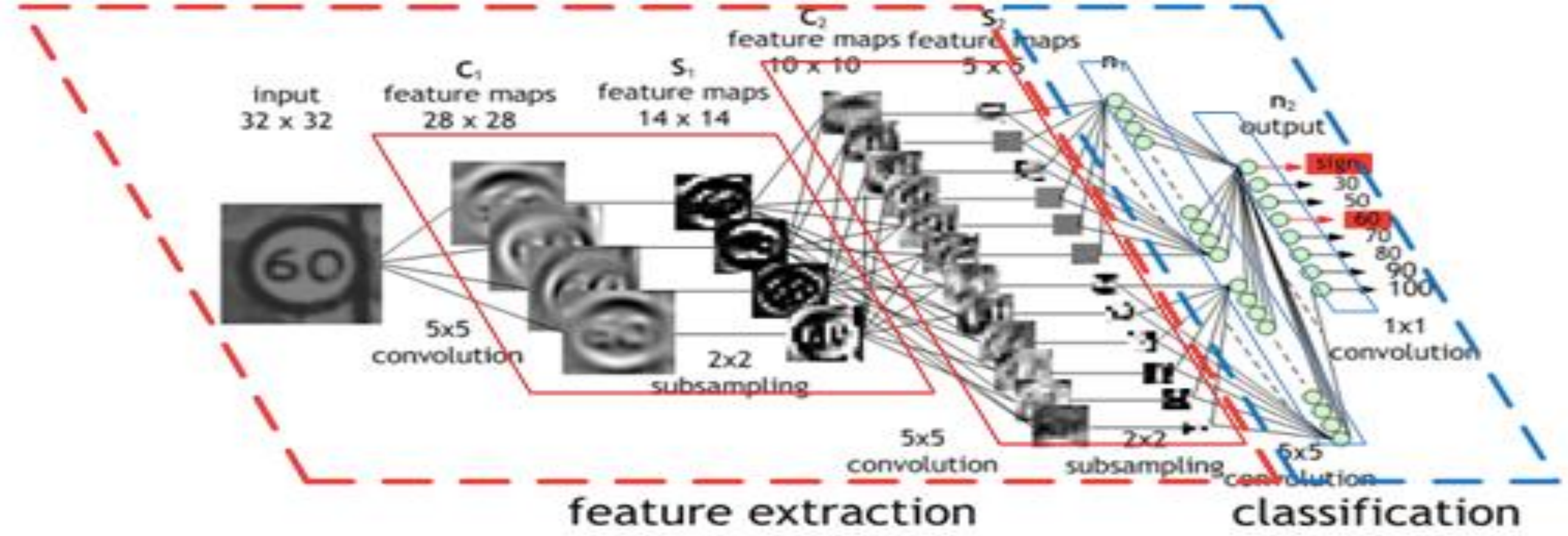
Gradient Descent Methods

- 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

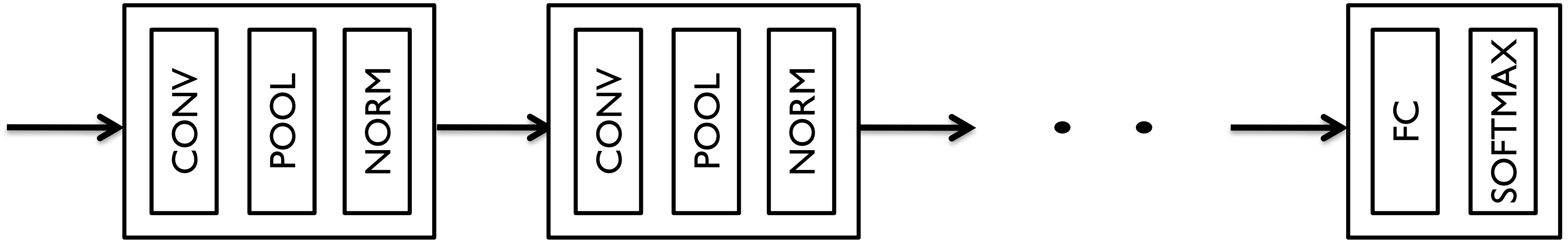
Variants

Convolutional Neural Networks for Image and Video Understanding



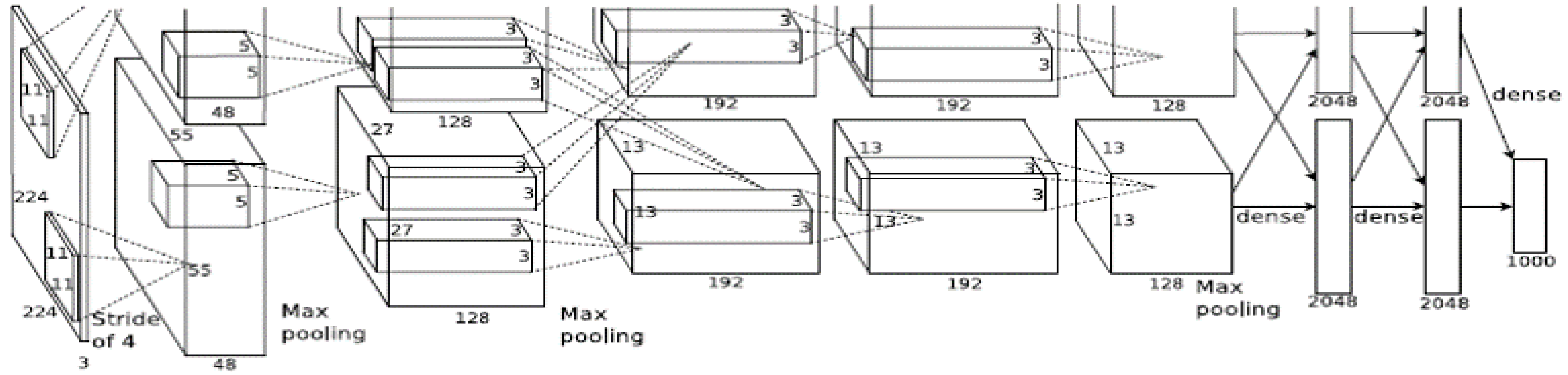
CNNs: Typical Architecture

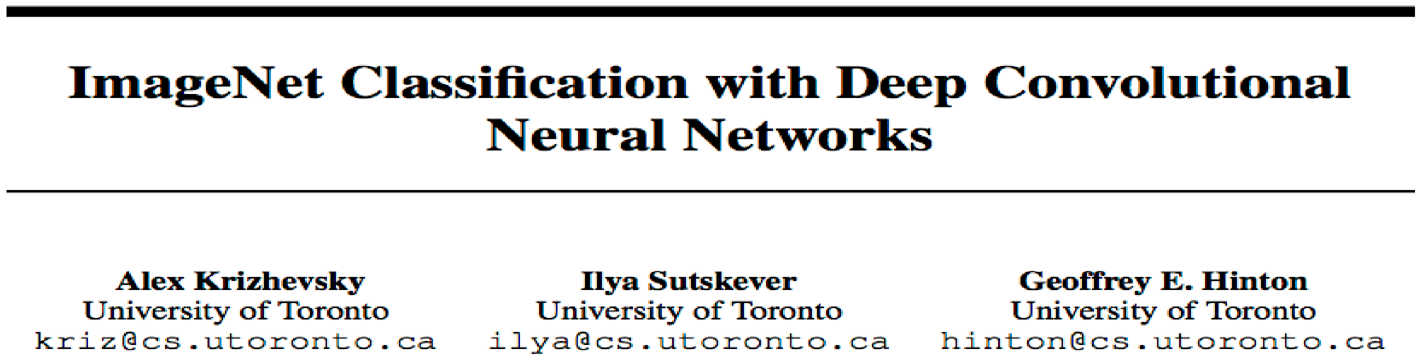
- A typical deep convolutional network



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

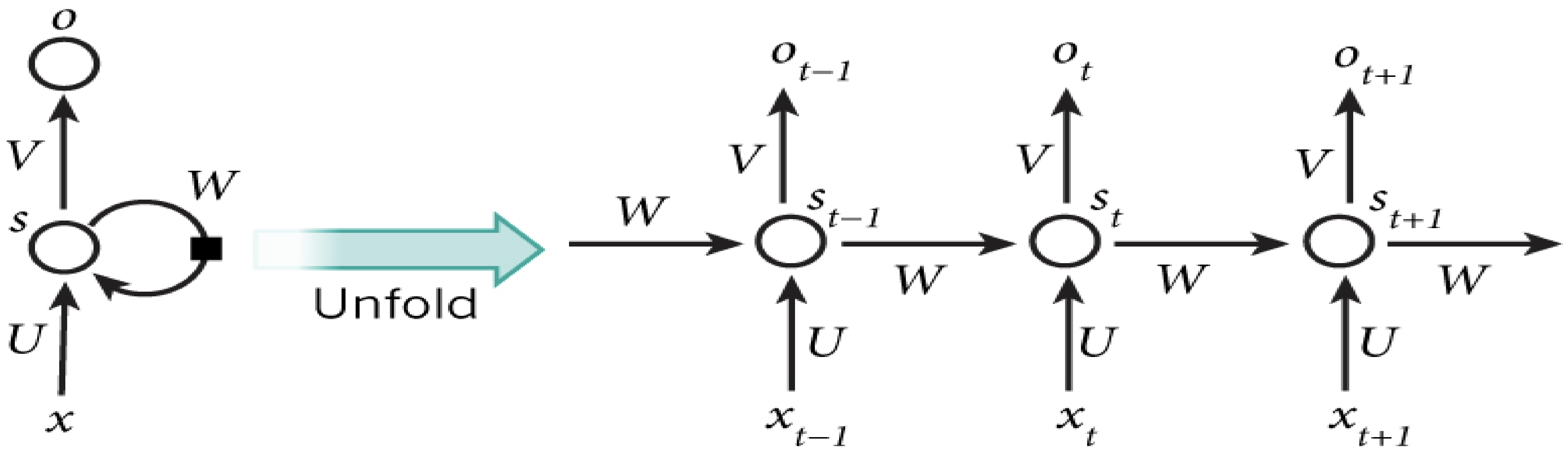
ImageNet Classification Task:

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

Deep Learning Architectures

Variants

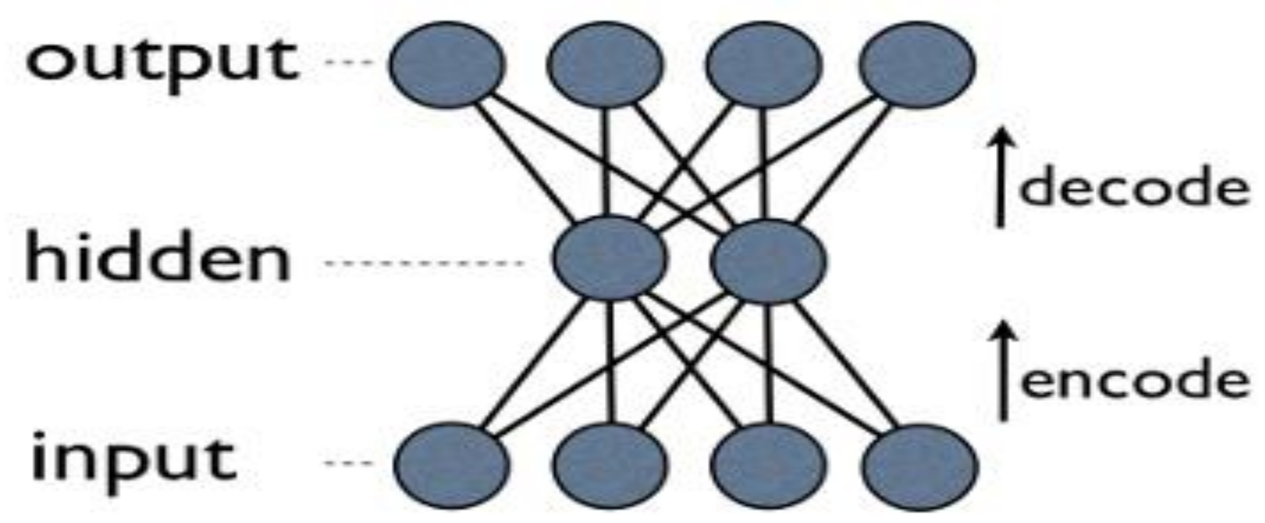
Recurrent Neural Networks for Time Series and Sequence Data Understanding



Deep Learning Architectures

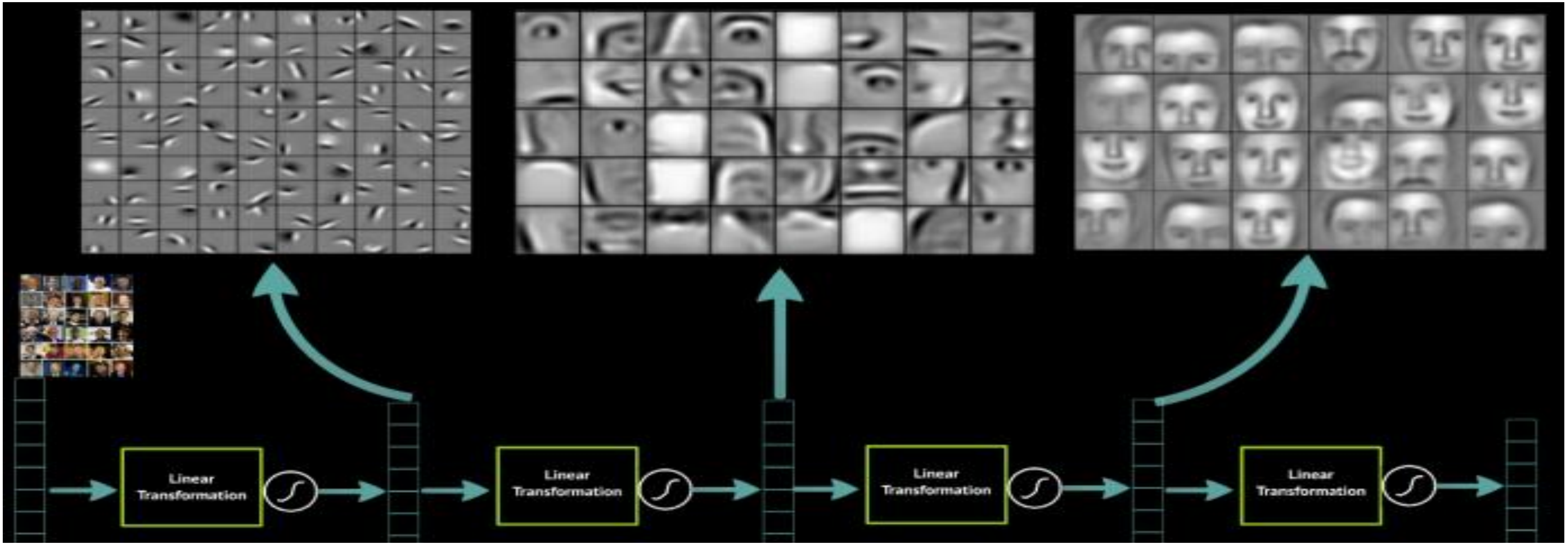
Variants

Deep Autoencoders for Dimensionality Reduction



Deep Learning

Why is it successful?



Deep Learning

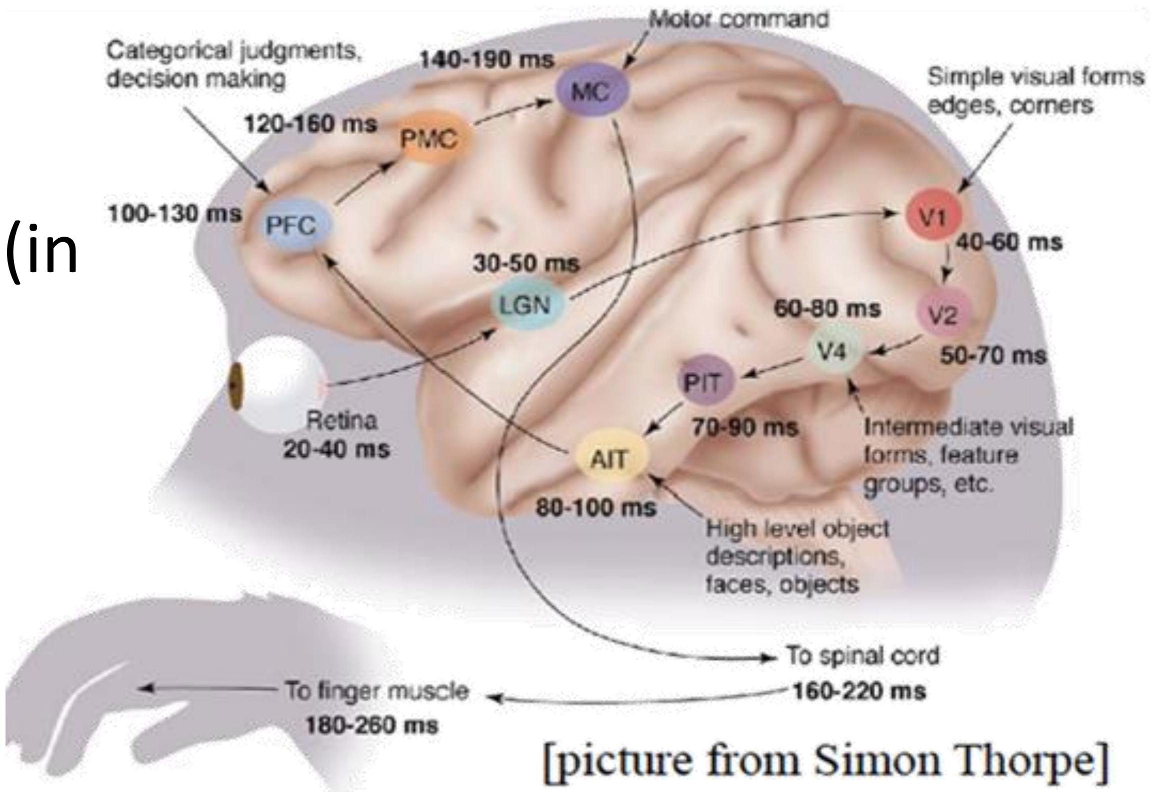
Why has use of multiple layers led to success?

- 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

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

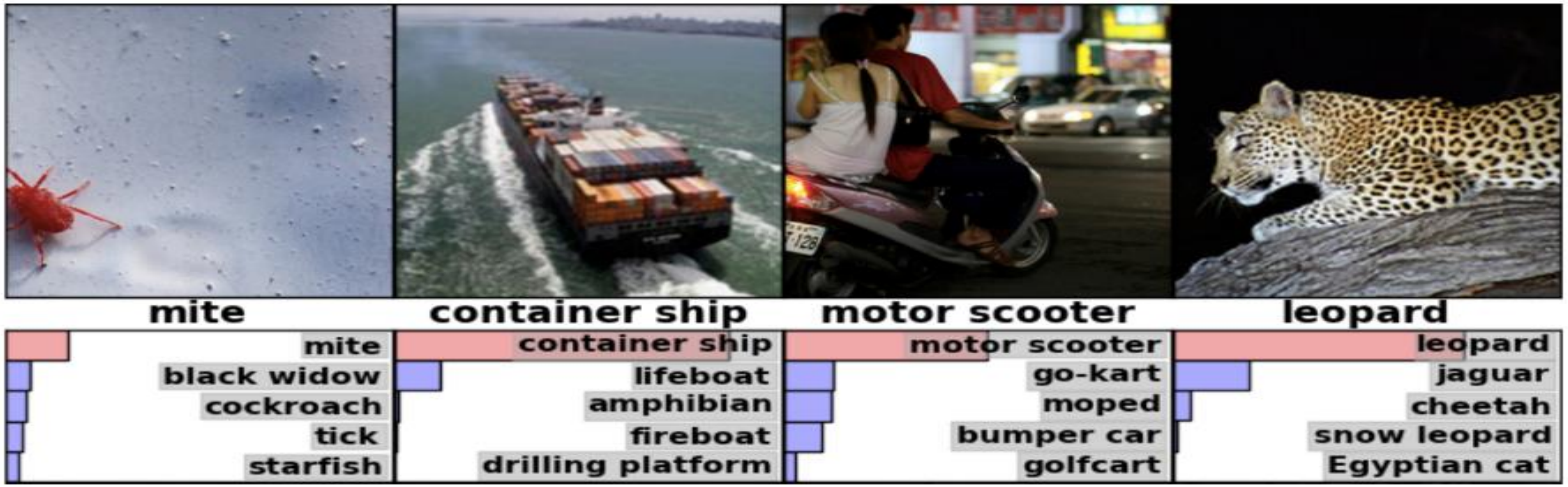


[picture from Simon Thorpe]

Deep Learning

Applications and Successes

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



Deep Learning

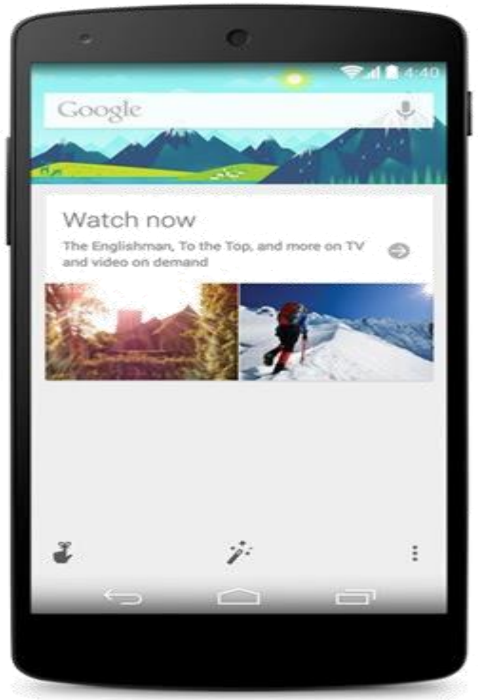
Applications and Successes

- Speech understanding and natural language processing

Apple Siri

I found quite a number of movies playing today near San Francisco:

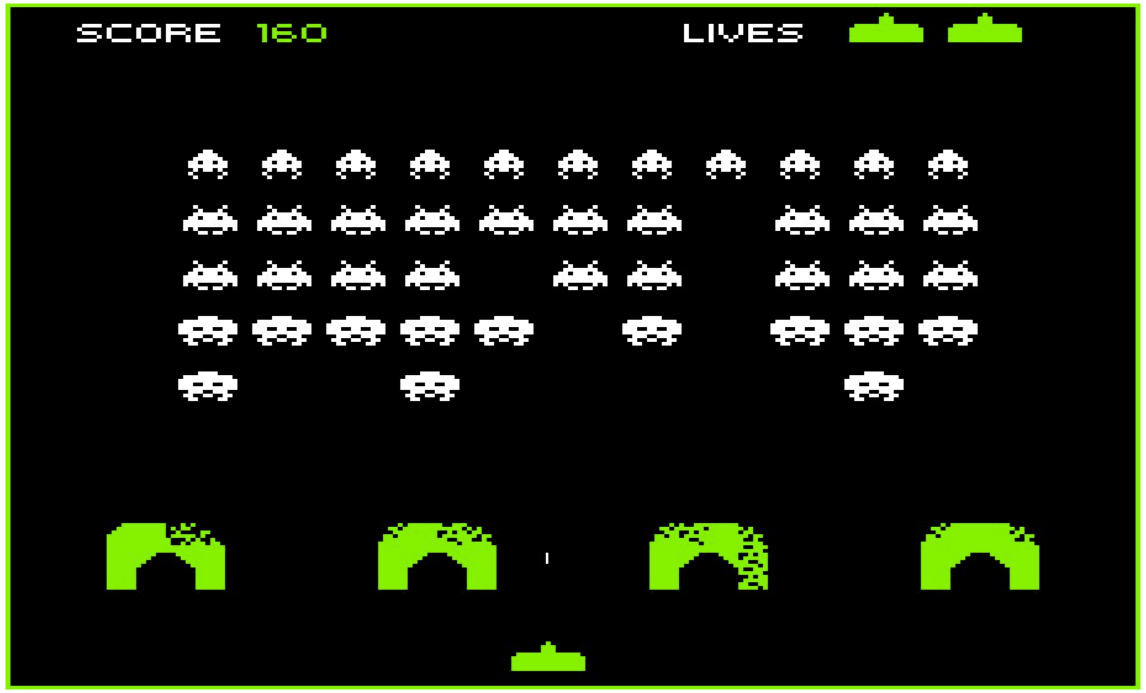
Google Now


Google
Watch now
The Englishman, To the Top, and more on TV and video on demand

Windows Cortana

Applications and Successes

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



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 11.1-11.11
- Bishop, PRML, Sec 5.1-5.3, 5.5