Introduction to deep neural networks

Albert Montillo, Ph.D.

Albert.Montillo@UTSouthwestern.edu

Deep Learning for Precision Health Lab
Web: utsouthwestern.edu/labs/montillo

Special thanks

- This talk consists of original slides and slides I've adapted from several sources. Special thanks to slides with content curtesy of those listed below.

  – Dr Andrew Ng, CS Dept, Stanford University

  – Dr Hugo Larochelle, Université de Sherbrooke

  – Dr Aarti Singh, CMU

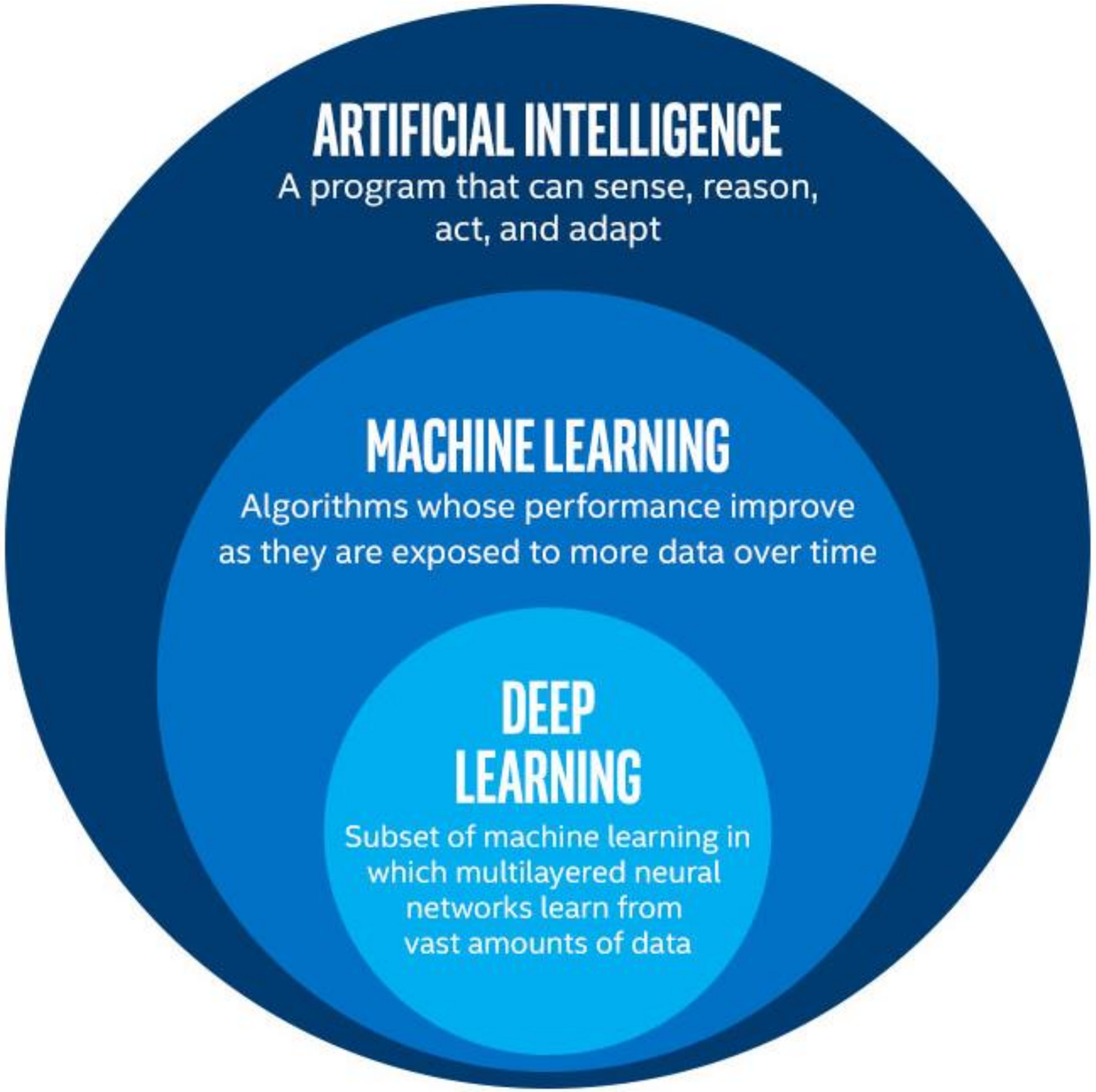
  – Adam Harley, Ryerson University

  – Lily Peng, MD/PhD, Google

  – Brett Kuprel, EE dept, Stanford

  – Weifeng Li, Victor Benjamin, Xiao Liu, and Hsinchun Chen, University of Arizona

What is machine and deep learning? Where do they fit in?



Definition of artificial neuron

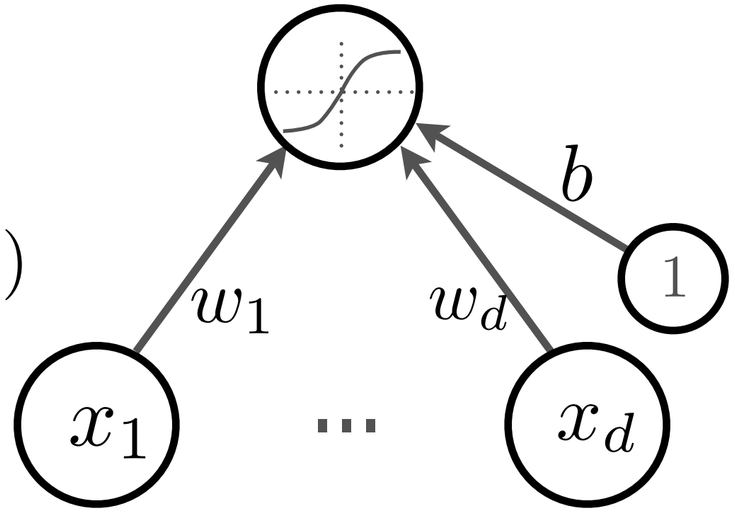
Topics: connection weights, bias, activation function

- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (output) activation

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

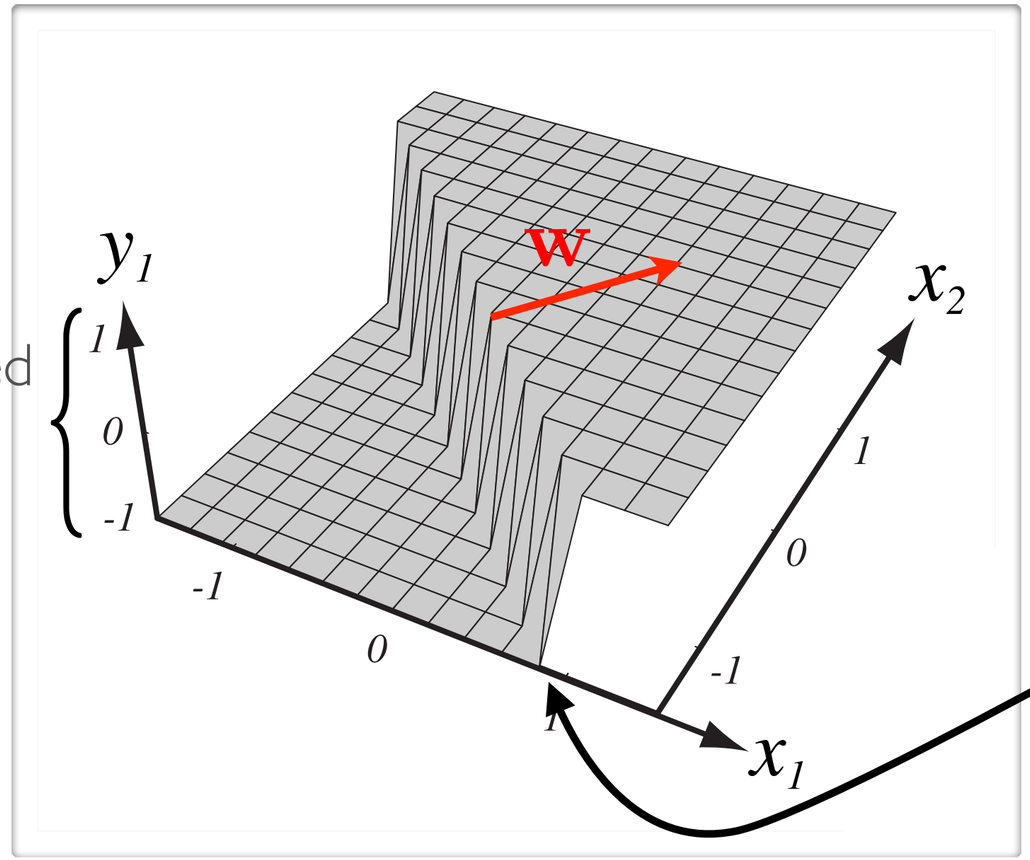


- $\mathbf{w}$ are the connection weights
- b is the neuron bias
- $g(\cdot)$ is called the activation function

Definition of artificial neuron

Topics: connection weights, bias, activation function

range determined by $g(\cdot)$



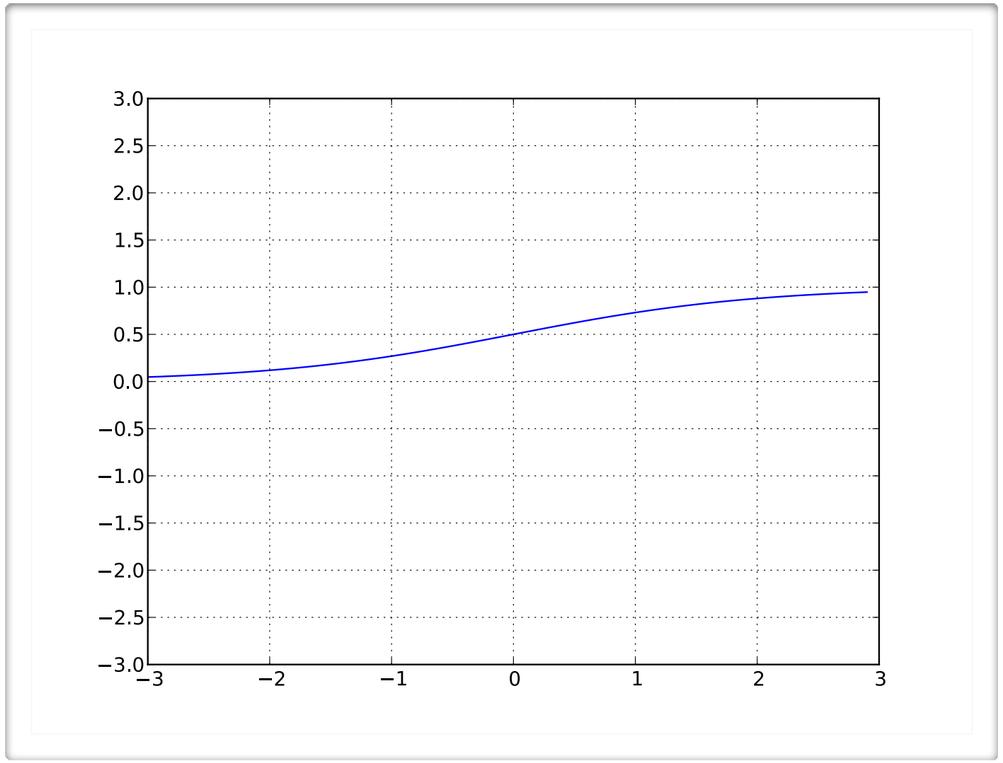
bias b only changes the position of the riff

(from Pascal Vincent's slides)

Artificial neuron activation function: sigmoid

Topics: sigmoid activation function

- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

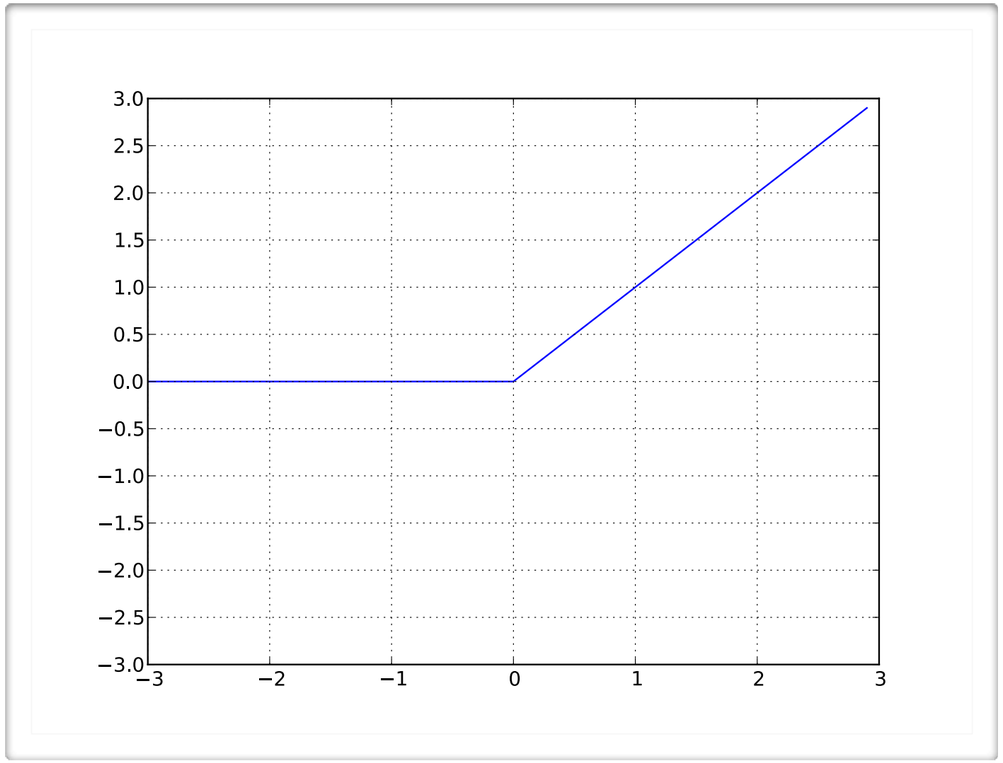


$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

Artificial neuron activation function: reclin

Topics: rectified linear activation function

- Bounded below by 0 (always non-negative)
- Not upper bounded
- Strictly increasing
- Tends to give neurons with sparse activities



$$g(a) = \text{reclin}(a) = \max(0, a)$$

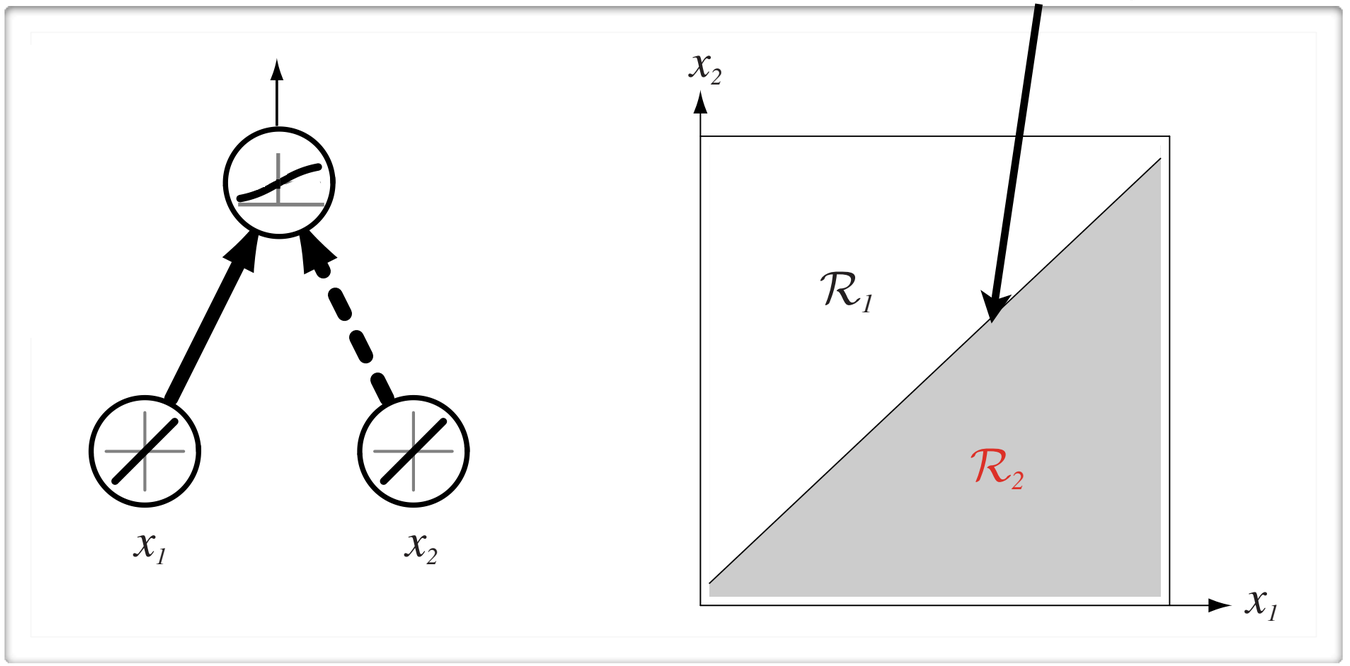
Artificial neuron: capacity

Topics: the artificial neurons's capacity to form complex decision boundaries

- Could do binary classification:
  - with sigmoid, can interpret neuron as estimating $p(y = 1|\mathbf{x})$
  - also known as logistic regression classifier
  - if greater than 0.5, predict class 1
  - otherwise, predict class 0

(similar idea can apply with tanh)

decision boundary is linear

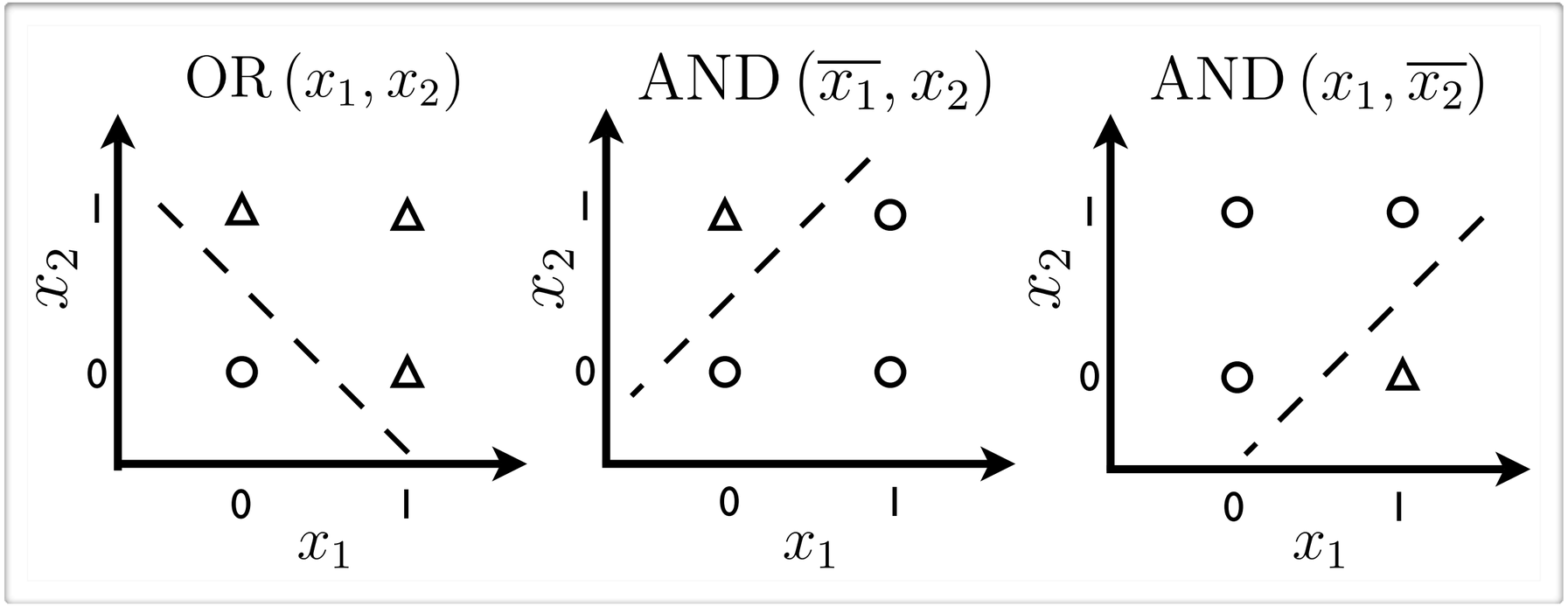


(from Pascal Vincent's slides)

Single artificial neuron: capacity

Topics: capacity of single neuron

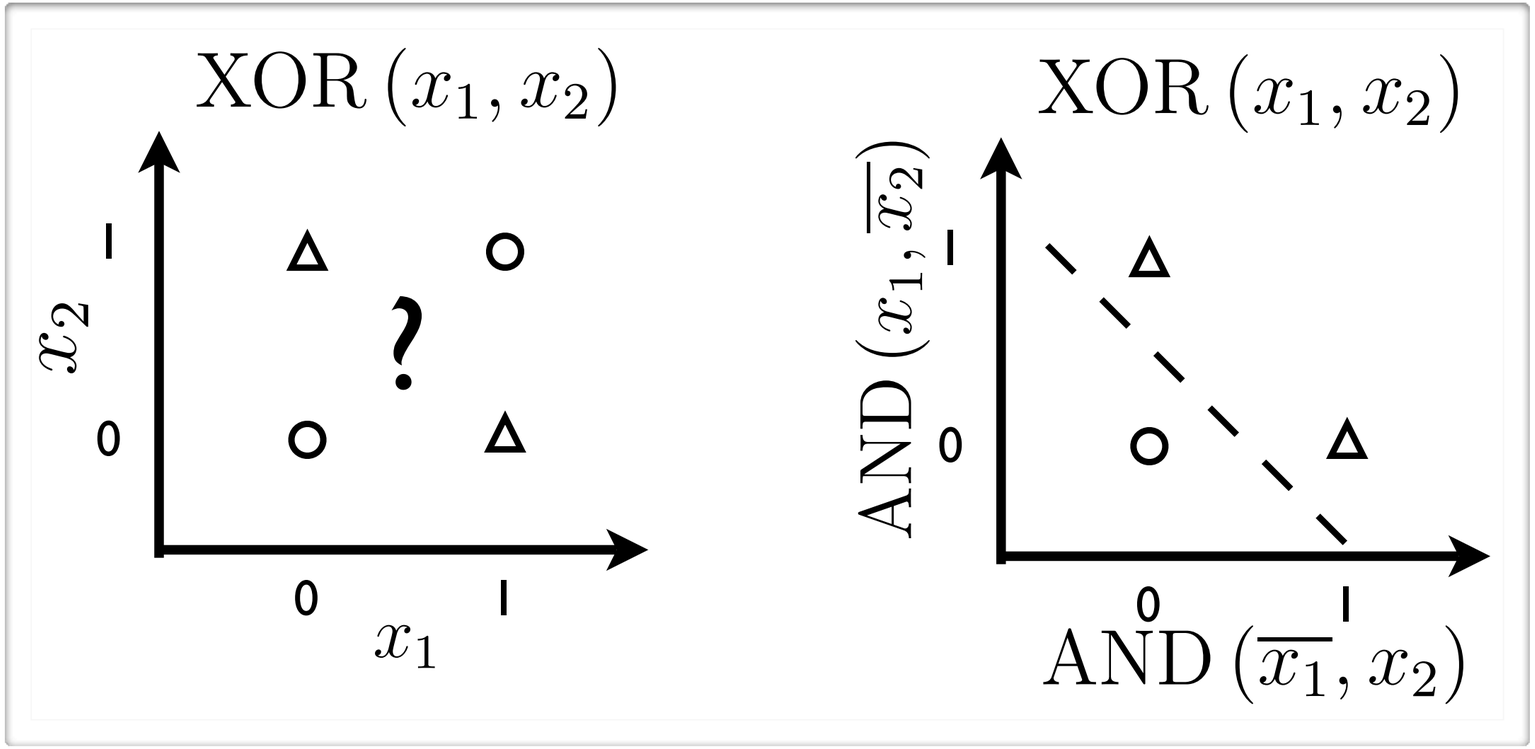
- Can solve linearly separable problems



Single artificial neuron: capacity

Topics: capacity of single neuron

- Can't solve non linearly separable problems...



- ... unless the input is transformed in a better representation

Neuron combinations: Neural network with 1 hidden layer

Topics: single hidden layer neural network

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

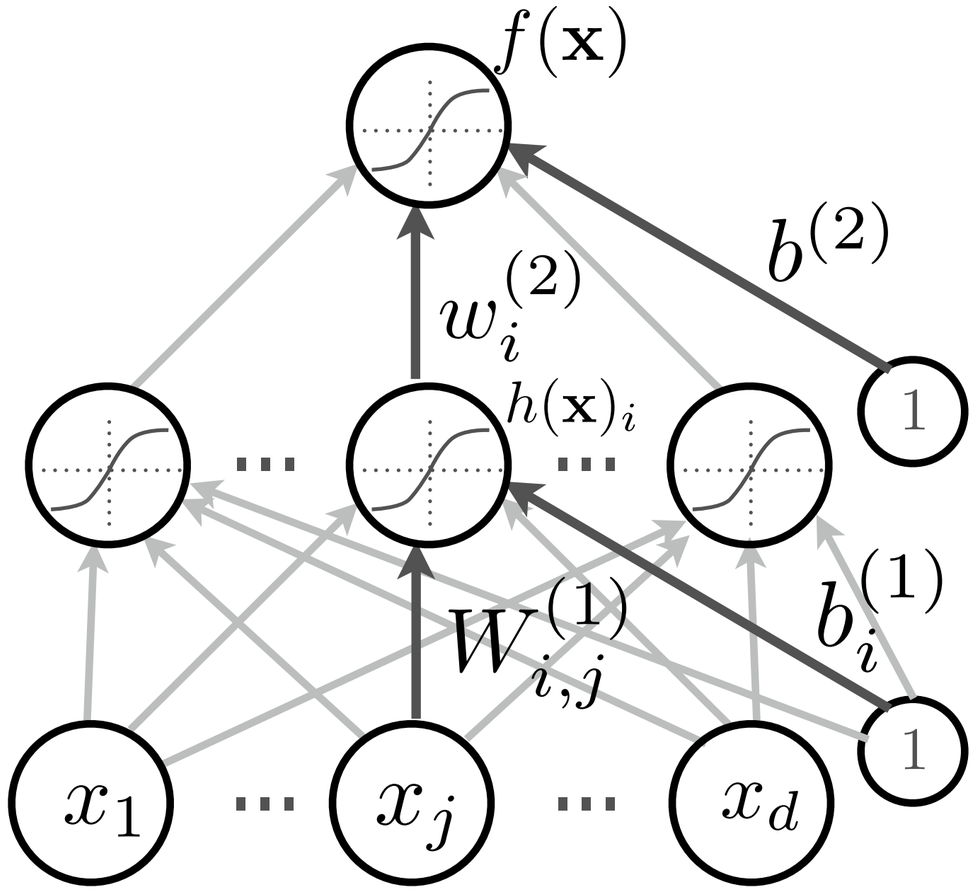
- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$

output activation function



Neural network: output layer's softmax activation function

Topics: softmax activation function

- For multi-class classification: i.e. > 2 classes
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability $p(y = c|\mathbf{x})$

  we will use a set of output neurons, 1 per class. Jth neuron indicates the probability of class J

- We use the softmax activation function at the output:

$$\mathbf{o}(\mathbf{a}) = \text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)} \right]^\top$$

  - strictly positive
  - sums to one

- Predicted class is the one with highest estimated probability

Neuron combinations: Neural network with >1 hidden layer

Topics: multilayer neural network

- Could have L hidden layers:
  - layer pre-activation for $k>0$ $(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

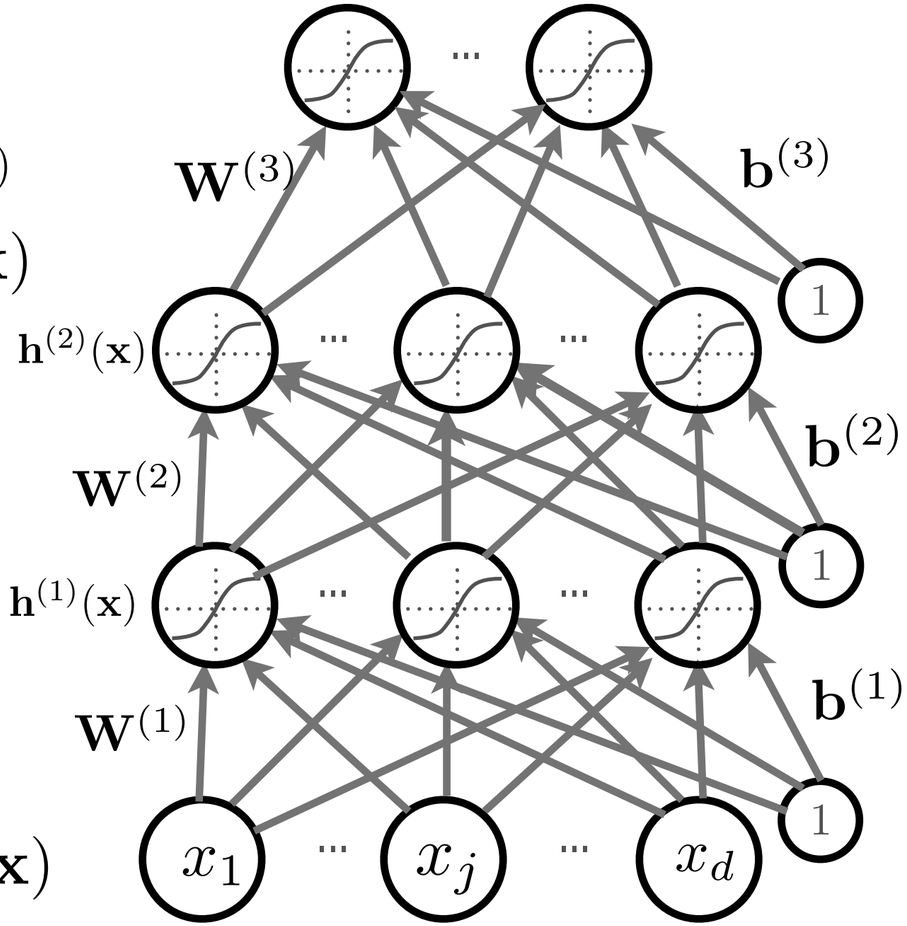
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

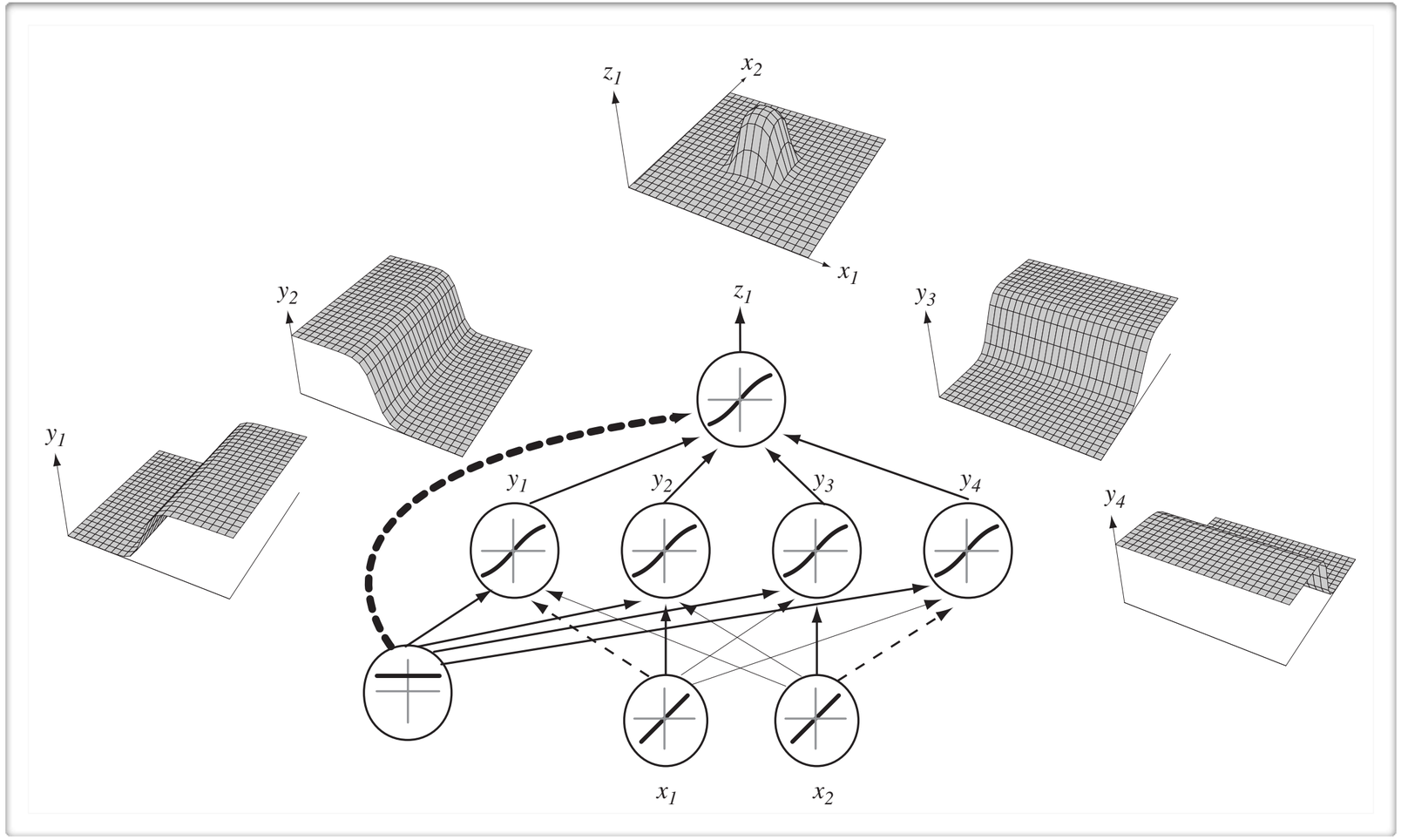
  - output layer activation ($k=L+1$):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



Capacity of single hidden layer network

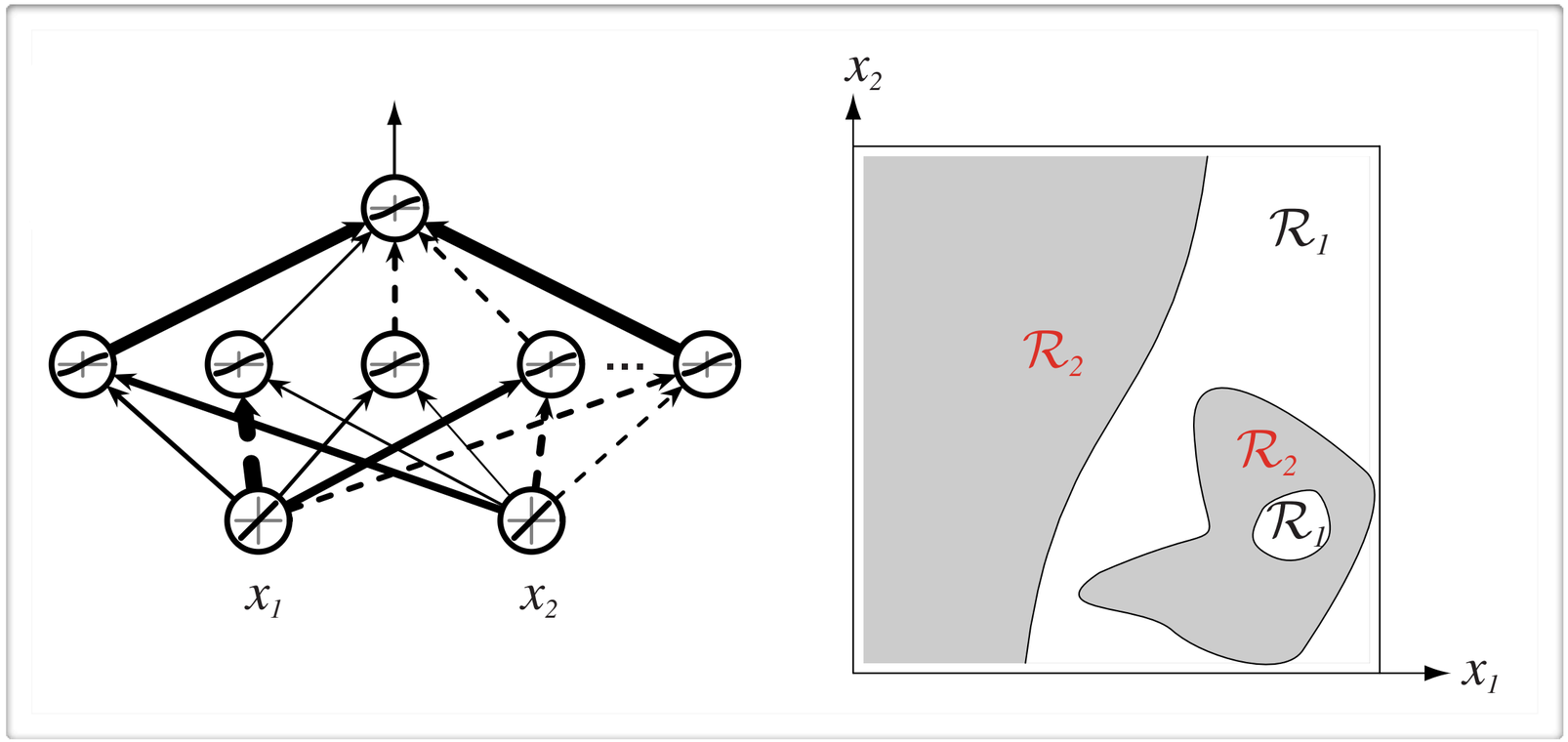
Topics: complexity of decision boundaries: using sigmoid activation function



(from Pascal Vincent's slides)

Capacity of single hidden layer network

Topics: single hidden layer neural network



(from Pascal Vincent's slides)

Any decision boundary using 1 layer

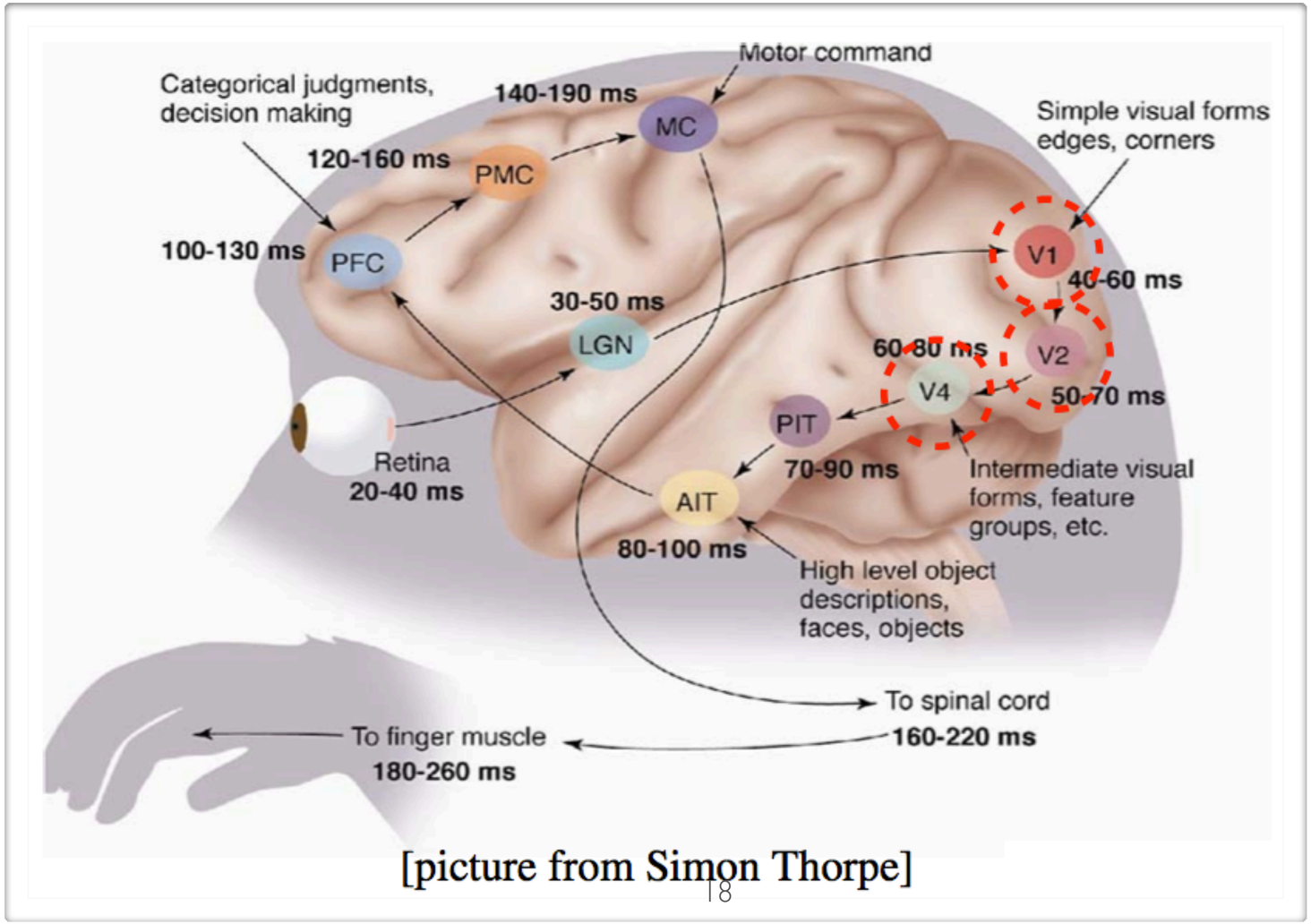
Topics: universal approximation

- Universal approximation theorem (Hornik, 1991):
  - "a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units' '
- The result applies for sigmoid, tanh and many other hidden layer activation functions

- This is a good result, but it doesn't mean there is a lear ning algorithm that can find the necessary parameter values!

Inspiration from neurophysiology

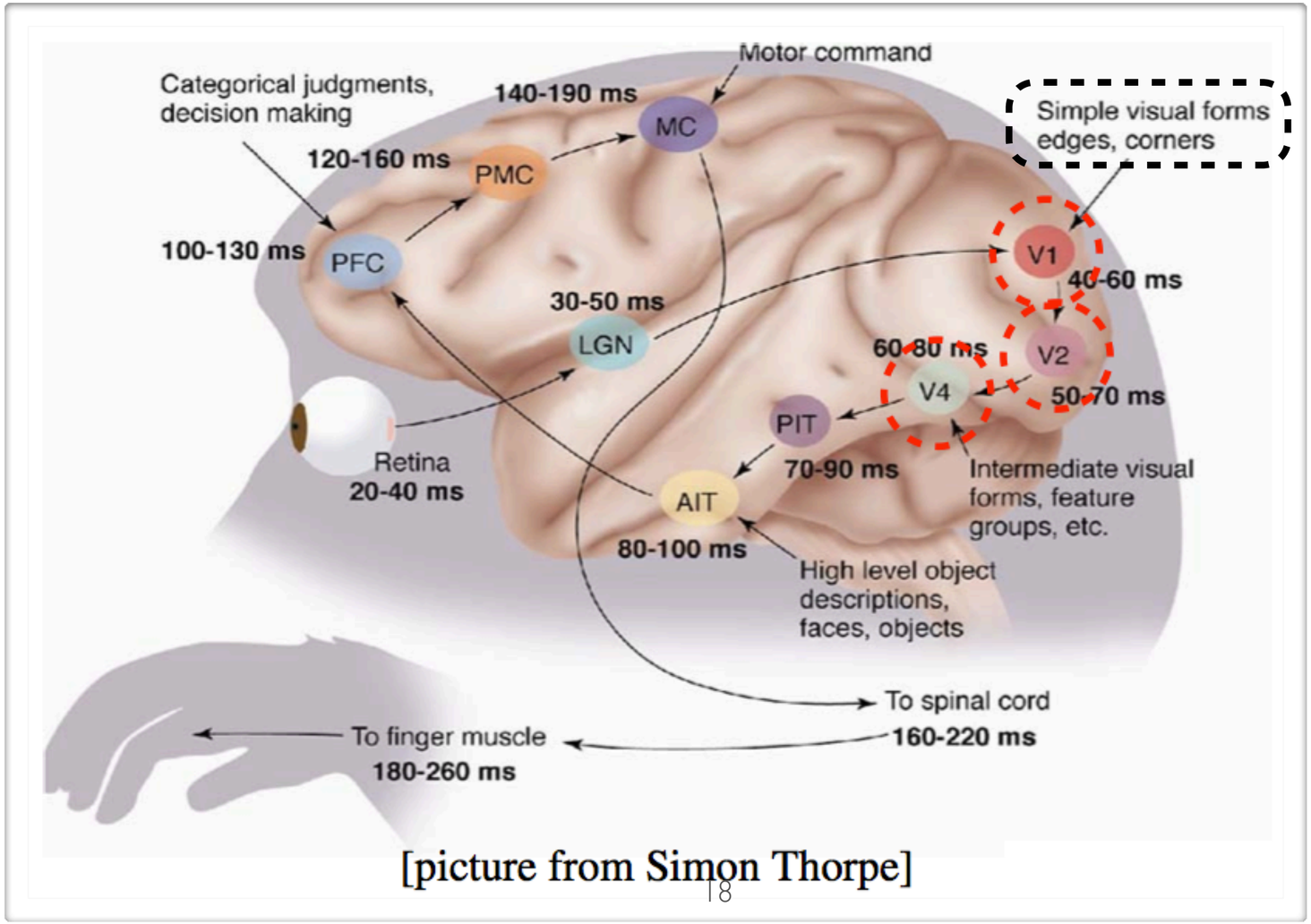
Topics: deep processing within human visual system, visual cortex



[picture from Simon Thorpe]

Inspiration from neurophysiology

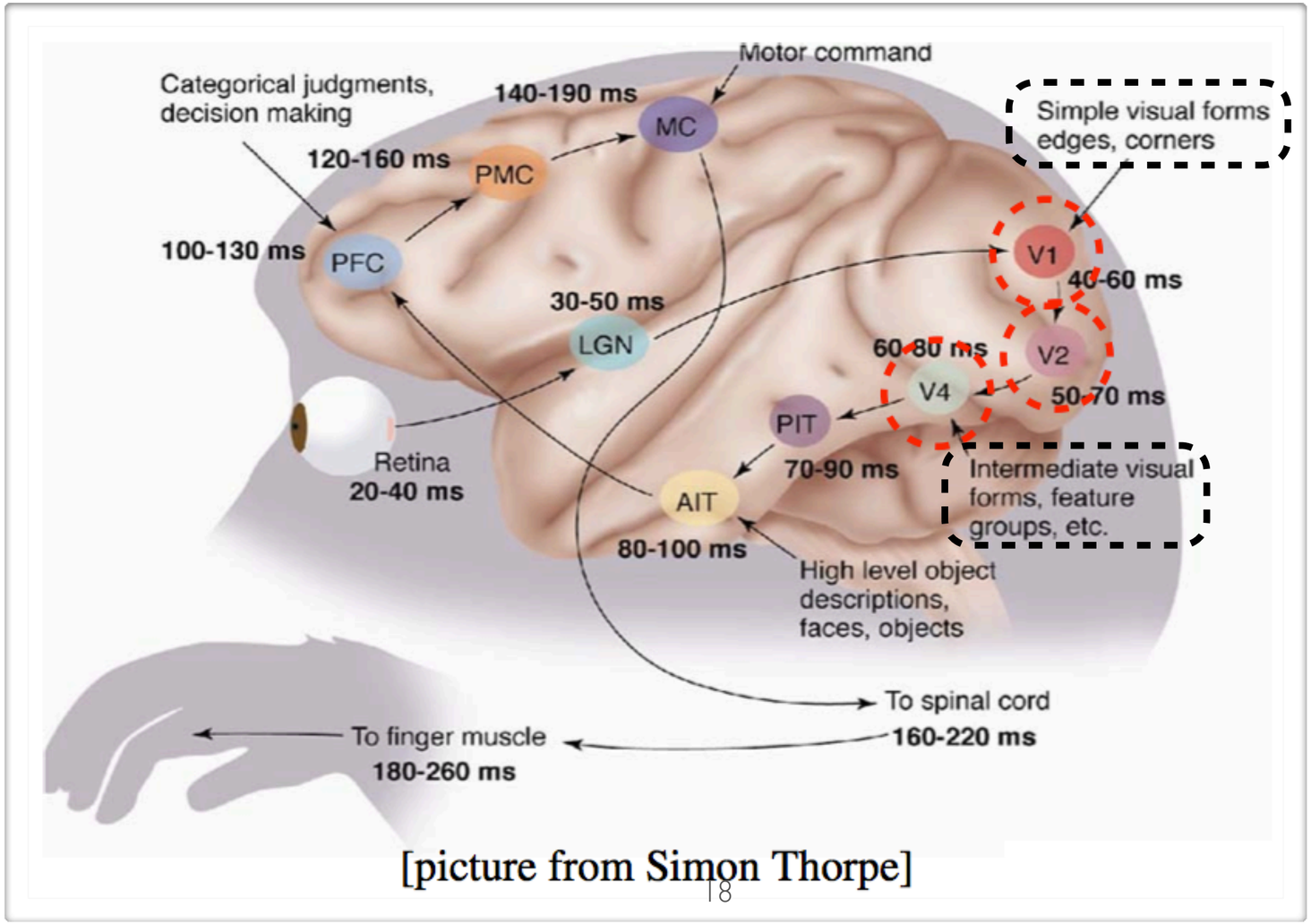
Topics: deep processing within human visual system, visual cortex



[picture from Simon Thorpe]

Inspiration from neurophysiology

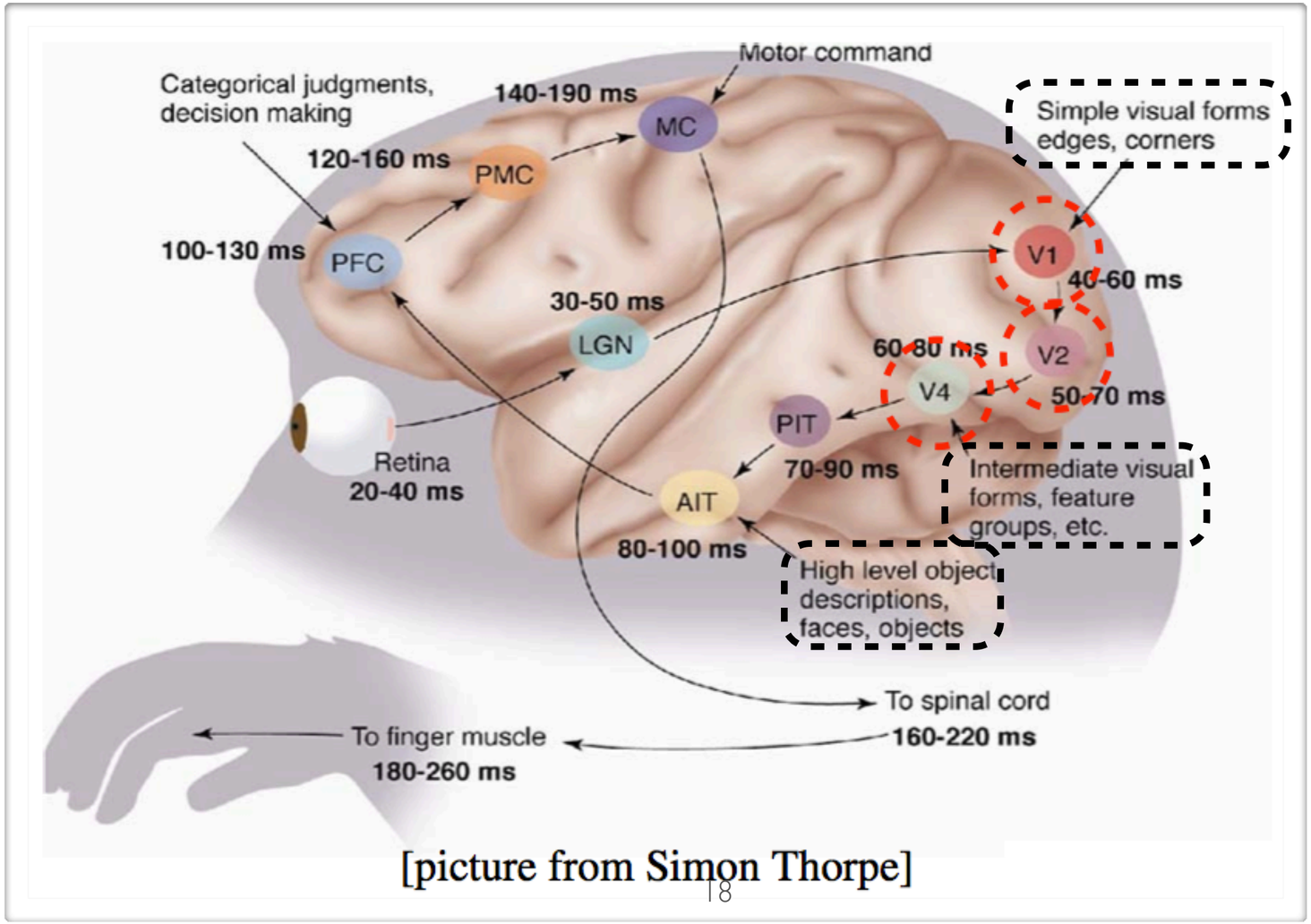
Topics: deep processing within human visual system, visual cortex



[picture from Simon Thorpe]

Inspiration from neurophysiology

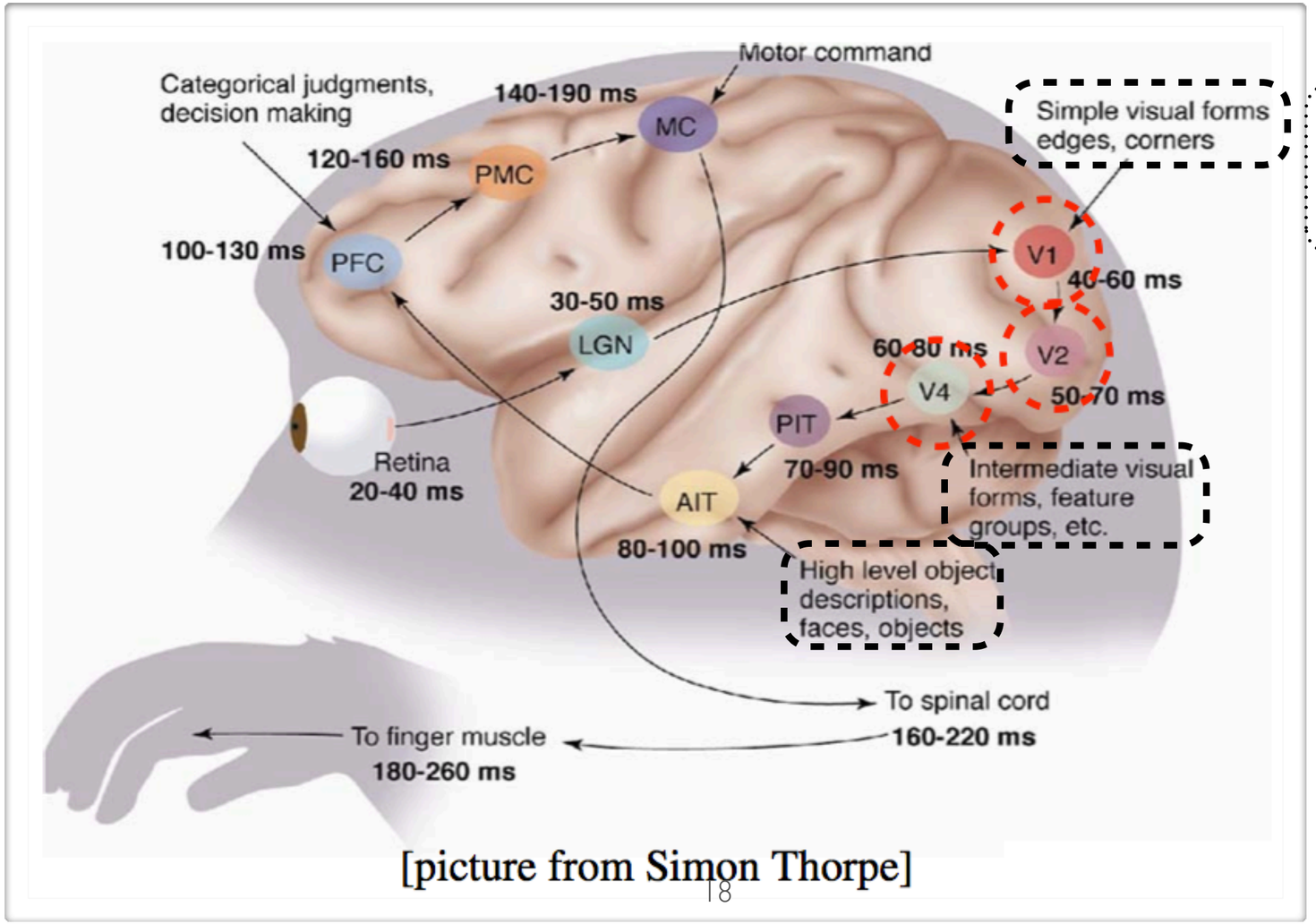
Topics: deep processing within human visual system, visual cortex



[picture from Simon Thorpe]

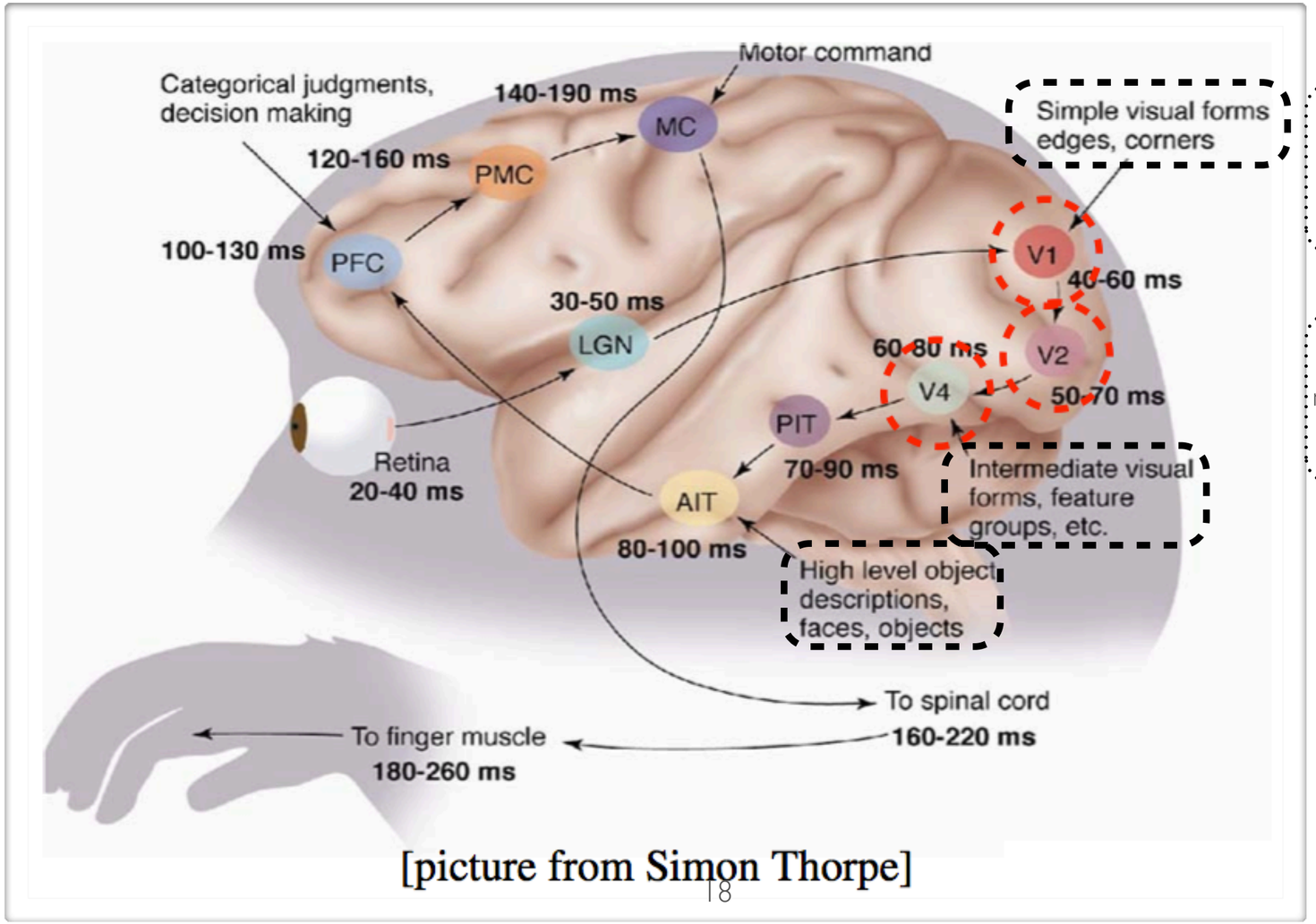
Inspiration from neurophysiology

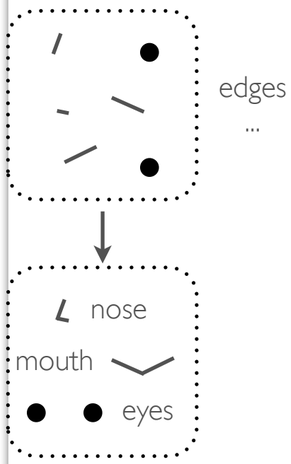
Topics: deep processing within human visual system, visual cortex



Inspiration from neurophysiology

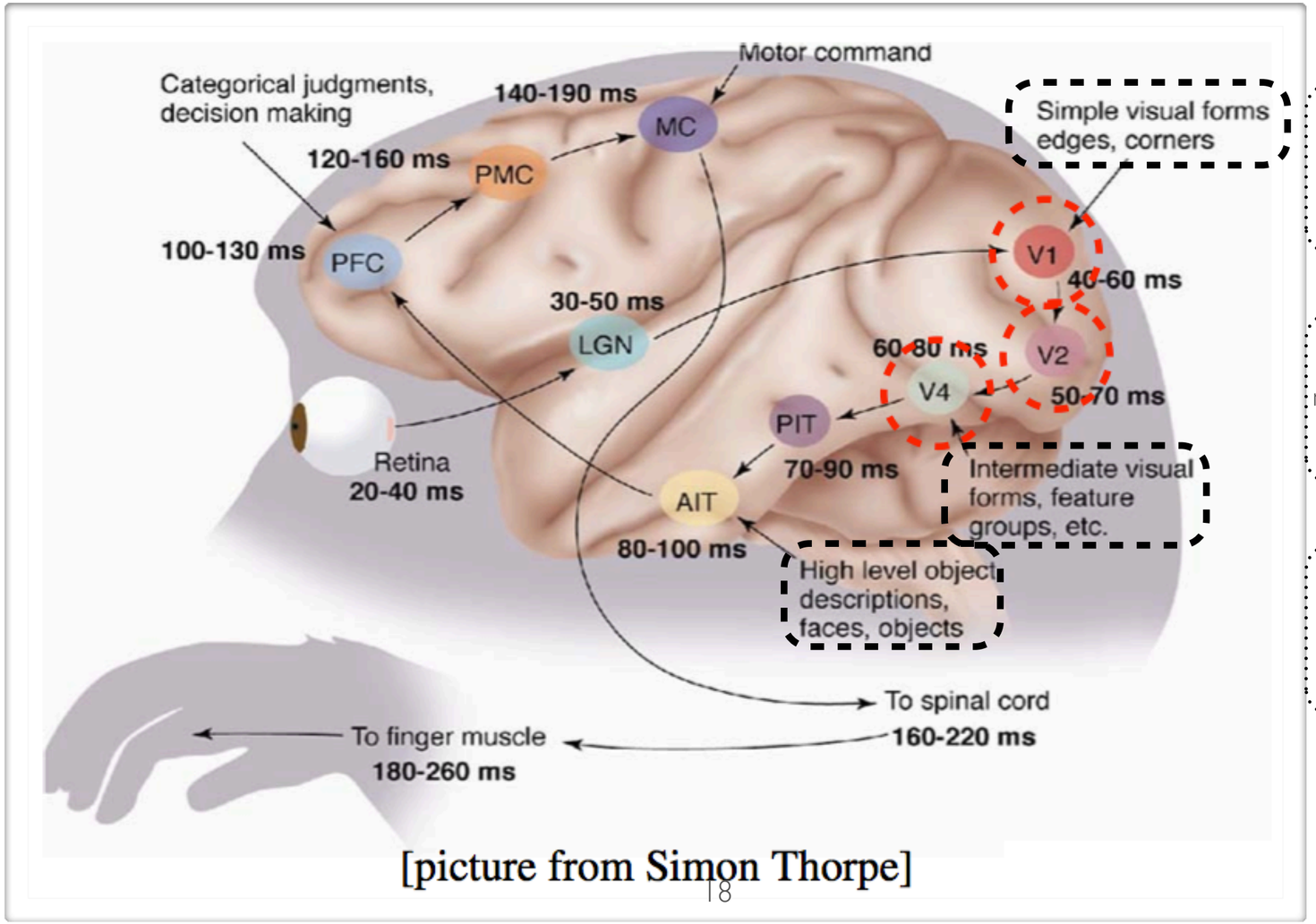
Topics: deep processing within human visual system, visual cortex

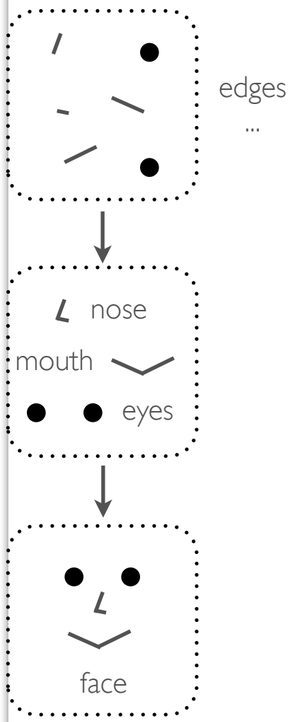




Inspiration from neurophysiology

Topics: deep processing within human visual system, visual cortex



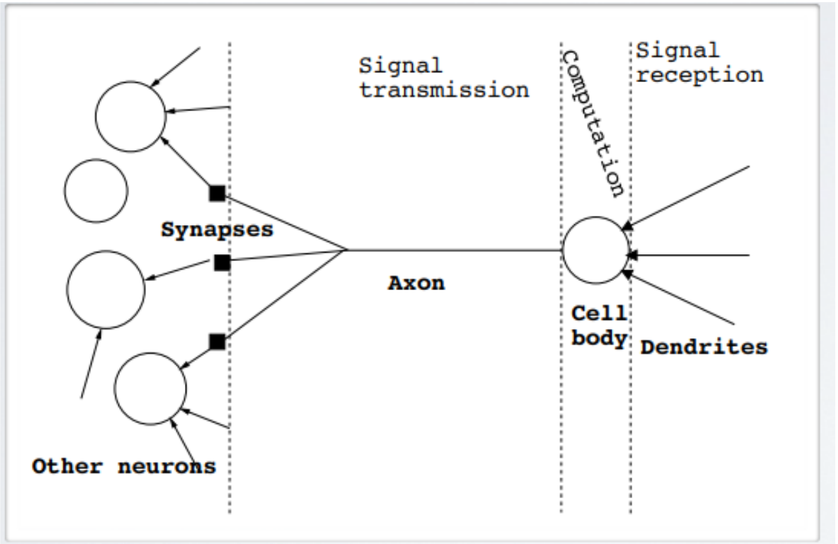


Artificial and biological neurons

- Firing rates of different input neurons combine to influence the firing rate of other neurons:
  - depending on the dendrite and axon, a neuron can either work to increase (excite) or decrease (inhibit) the firing rate of another neuron

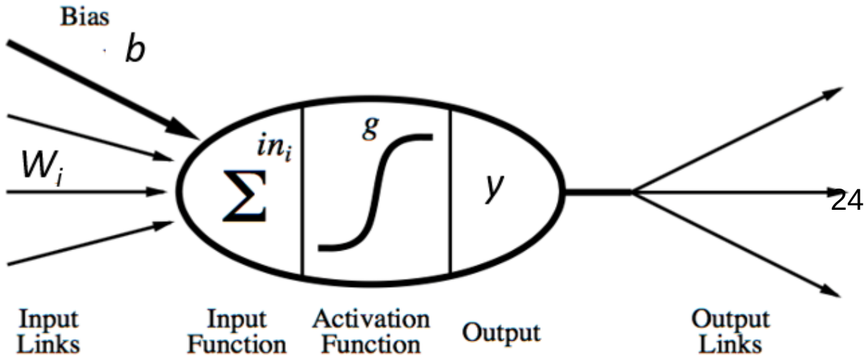
- This is what artificial neurons approximate:
  - the activation corresponds to a "sort of" firing rate
  - the weights between neurons model whether neurons excite or inhibit each other
  - the activation function and bias model the thresholded behavior of action potentials
- Do we need to simulate exactly? Do airplanes have to flap to fly?

Biological neuron



(from Hyvärinen, Hurri and Hoyer's book)

Artificial neuron



MACHINE LEARNING

Topics: stochastic gradient descent (SGD) is one way to speed up learning

- Algorithm performs updates after each example
  - initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
  - for N iterations
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
      - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
      - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \, \Delta$

    } training epoch = iteration over **all** examples

- **Outline of following sections** of this talk. To apply this algorithm we need:
  1. the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  2. a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  3. the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
  4. initialization method

1. Loss function

Topics: loss function for classification

- Neural network estimates $f(\mathbf{x})_c = p(y = c|\mathbf{x})$
  - we could maximize the probabilities of $y^{(t)}$ given $\mathbf{x}^{(t)}$ in the training set

- To frame as minimization, we minimize the negative log-likelihood

natural log (ln)

$$l(\mathbf{f}(\mathbf{x}), y) = -\sum_c 1_{(y=c)} \log f(\mathbf{x})_c = -\log f(\mathbf{x})_y$$

  - we take the log to simplify for numerical stability and math simplicity
  - sometimes referred to as cross-entropy

2. Intro to gradient Backpropagation

Purpose: adjust model parameters **w** and **b** to better fit training data

- This assumes a forward propagation has been made before
  - compute output gradient (before activation)

$$\nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y \quad \Longleftarrow \quad -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x}))$$

  - for k from $L+1$ to 1 (**propagate gradient backwards from last layer to the first**):
    - compute gradients of hidden layer parameter

$$\nabla_{\mathbf{W}^{(k)}} - \log f(\mathbf{x})_y \quad \Longleftarrow \quad \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \quad \mathbf{h}^{(k-1)}(\mathbf{x})^\top$$

$$\nabla_{\mathbf{b}^{(k)}} - \log f(\mathbf{x})_y \quad \Longleftarrow \quad \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$

    - compute gradient of hidden layer below

$$\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \quad \Longleftarrow \quad \mathbf{W}^{(k)^\top} \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right)$$

    - compute gradient of hidden layer below (before activation)

$$\nabla_{\mathbf{a}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \quad \Longleftarrow \quad \left(\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \odot [\ldots, g'(a^{(k-1)}(\mathbf{x})_j), \ldots]$$

4. Model parameter initialization

From an initial conditions, gradients refine params to min. loss on training data.
Define the initial conditions:

- For biases
  - initialize all to 0
- For weights
  - Can't initialize weights to 0 with tanh activation
    - we can show that all gradients would then be 0 (saddle point)
  - Can't initialize all weights to the same value
    - we can show that all hidden units in a layer will always behave the same
    - need to break symmetry
  - Recipe: sample $\mathbf{W}_{i,j}^{(k)}$ from $U\left[-b, b\right]$ where $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$
    - the idea is to sample around 0 but break symmetry
    - other values of b could work well (not an exact science) (see Glorot & Bengio, 2010)

H_k=size of $\mathbf{h}^{(k)}(\mathbf{x})$

Example: b is small value near zero:
e.g. b=0.14 if H_k=100 H_{k-1}=200

MODEL SELECTION

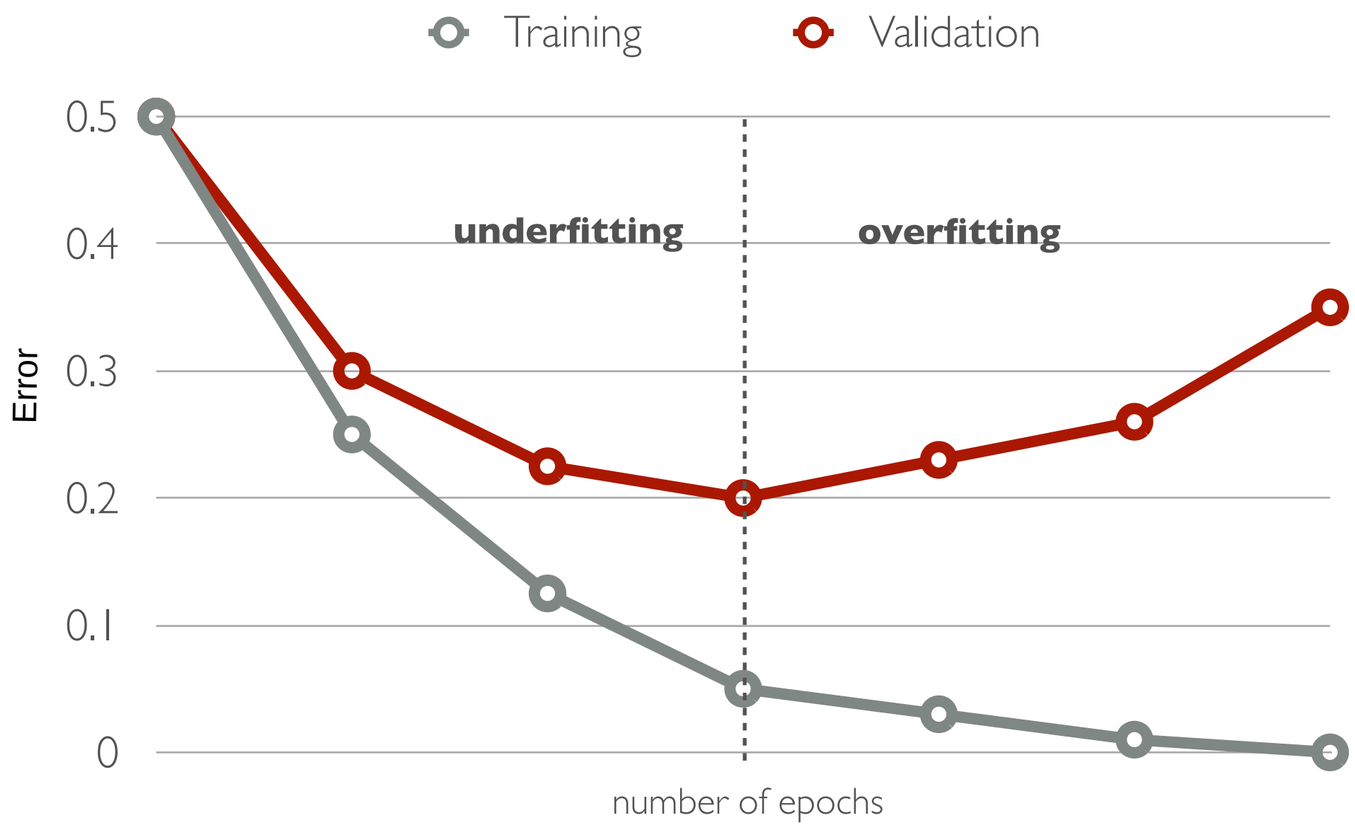
Gradients refine params of given model (e.g. wts W) but not hyper-params (e.g # layers, #neurons/layer, choice of activation function, regularization method)

- To search for the best configuration of the hyper-parameters:
  - you can perform a grid search
    - specify a set of values you want to test for each hyper-parameter
    - try all possible configurations of these values
  - you can perform a random search
    - specify a distribution over the values of each hyper-parameters (e.g. uniform in some range)
    - sample independently each hyper-parameter to get a configuration, and repeat as many times as wanted
- Use a validation set performance to select the best configuration
- You can go back and refine the grid/distributions if needed

KNOWING WHEN TO STOP

Rather than training a fixed number of epochs, use early stopping

- Stop training at the epoch when validation set error starts increasing (with some look ahead). This prevents overfitting the model to the training data.

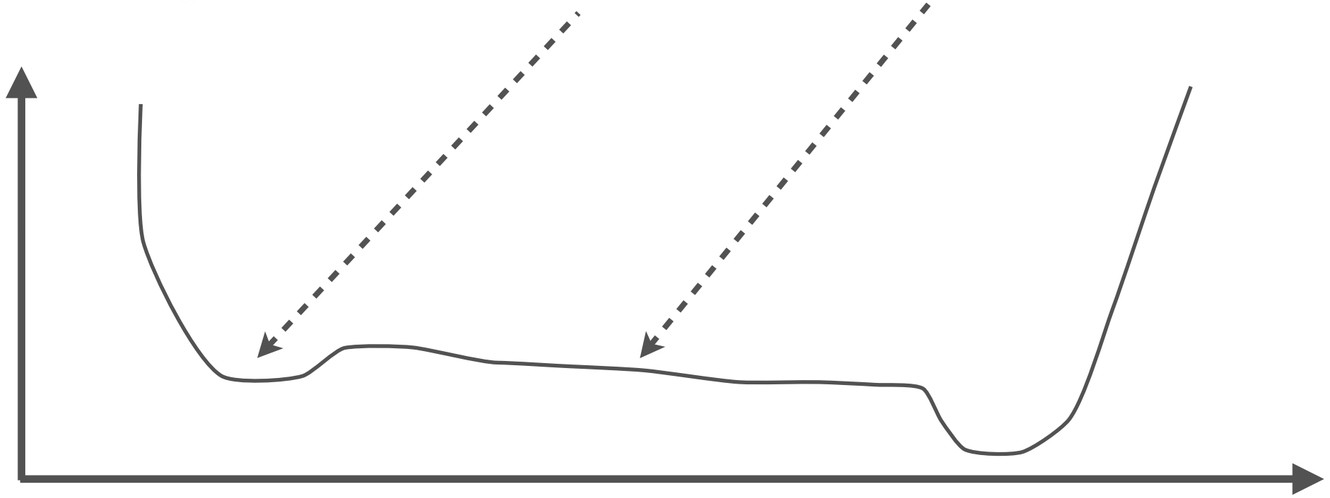


OPTIMIZATION

Topics: local optimum, global optimum, plateau

- Notes on the optimization problem
  - there isn't a single global optimum (non-convex optimization)
    - e.g. we can permute the hidden units (with their connections) and get the same function
    - we say that the hidden unit parameters are not identifiable

- Optimization can get stuck in local minimum or plateaus



Selecting the rule for the learning rate α_t

- Decreasing strategies: (δ is the decrease constant)
  - $\alpha_t = \frac{\alpha}{1+\delta t}$
  - $\alpha_t = \frac{\alpha}{t^\delta}$ (where $0.5 < \delta \leq 1$)
- Better to use a fixed learning rate for the first few updates

Questions

Albert.Montillo@UTSouthwestern.edu