EECE 5644: More Neural Networks (NNs)

Mark Zolotas

E-mail: m.zolotas@northeastern.edu
Webpage: https://coe.northeastern.edu/people/zolotas-mark/

Tentative Course Outline (Wks. 5-6*)

Topics	Dates	Assignments	Additional Reading
Neural Networks: Multilayer Perceptrons & Backpropagation	08/01-03	**Homework 3** released on Canvas on 08/01 **Due 08/10**	Chpts. 13.1-13.5 Murphy 2022
HW1 Review	08/02		N/A
Support Vector Machines (SVMs)	08/04		Burges Tutorial
Clustering: K-means, Gaussian Mixture Models (GMMs)	08/08	**Homework 4** released on Canvas on 08/08 **Due 08/17**	Chpt. 21 Murphy 2022
More on Deep Learning (CNNs & RNNs)	08/09		Deep Learning Goodfellow et al. 2016

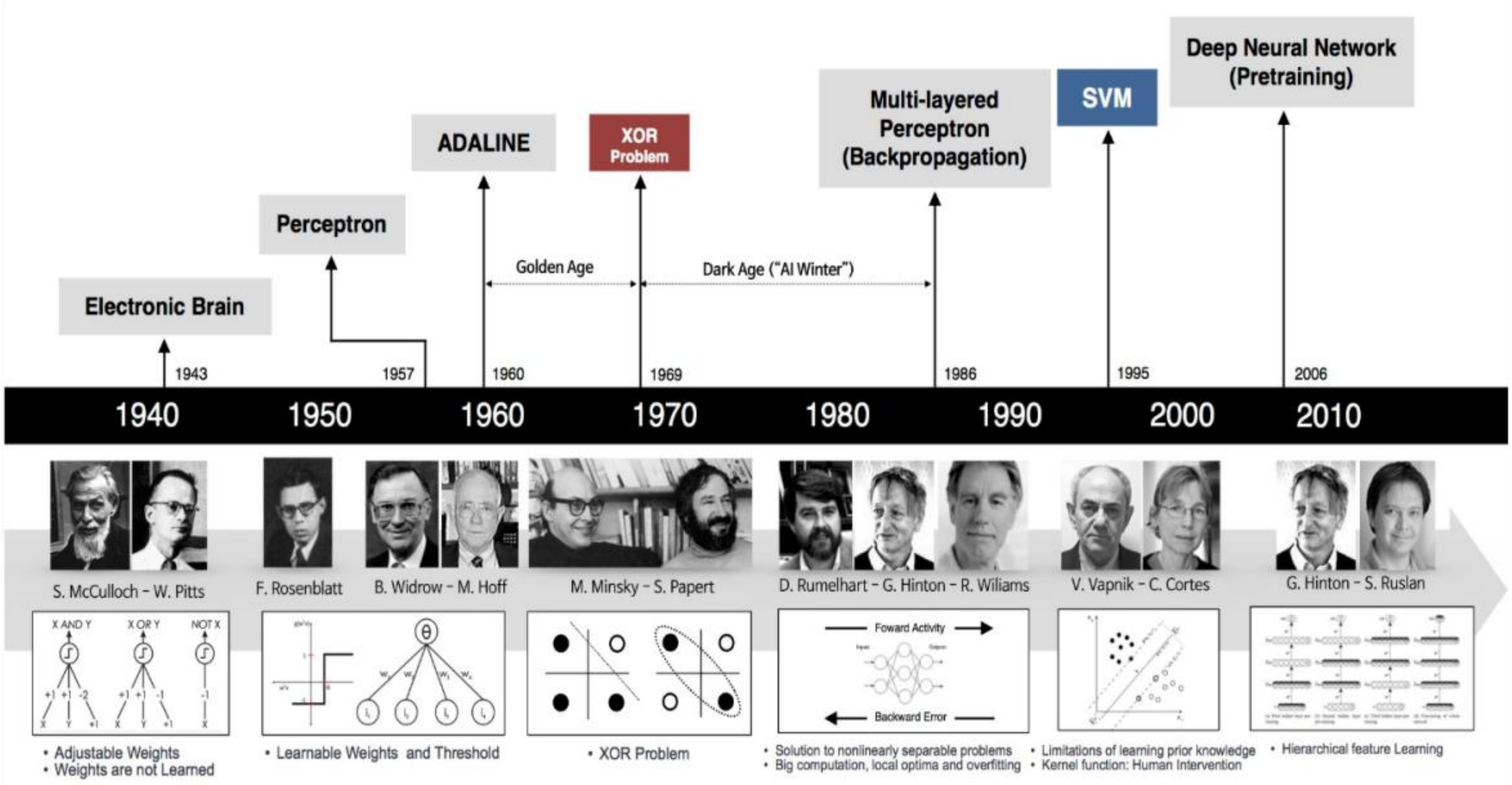
Tentative Course Outline (Wks. 6*-8)

<table>
<tr><th>Topics</th><th>Dates</th><th>Assignments</th><th>Additional Reading</th></tr>
<tr><td>Project + Practical Tips</td><td>08/10</td><td rowspan="6">Project teams (2-3 ppl. strict) are fully formed by 08/12

Final Project Reports & Code Due 08/22

Presentations on 08/22-23 in normal lecture hours and office hours depending on no. of groups</td><td>N/A</td></tr>
<tr><td>Ensemble Methods: Decision Trees, Boosting & Bagging</td><td>08/11</td><td>Chpt. 18
Murphy 2022</td></tr>
<tr><td>Model Predictive Control (MPC)</td><td>08/15-16</td><td>TBD</td></tr>
<tr><td>Gaussian Processes</td><td>08/17</td><td>TBD</td></tr>
<tr><td>Representation Learning (Autoencoders)</td><td>08/18</td><td>Chpt. 20
Murphy 2022</td></tr>
<tr><td>Project Presentations</td><td>08/22-23</td><td>N/A</td></tr>
</table>

History of NNs



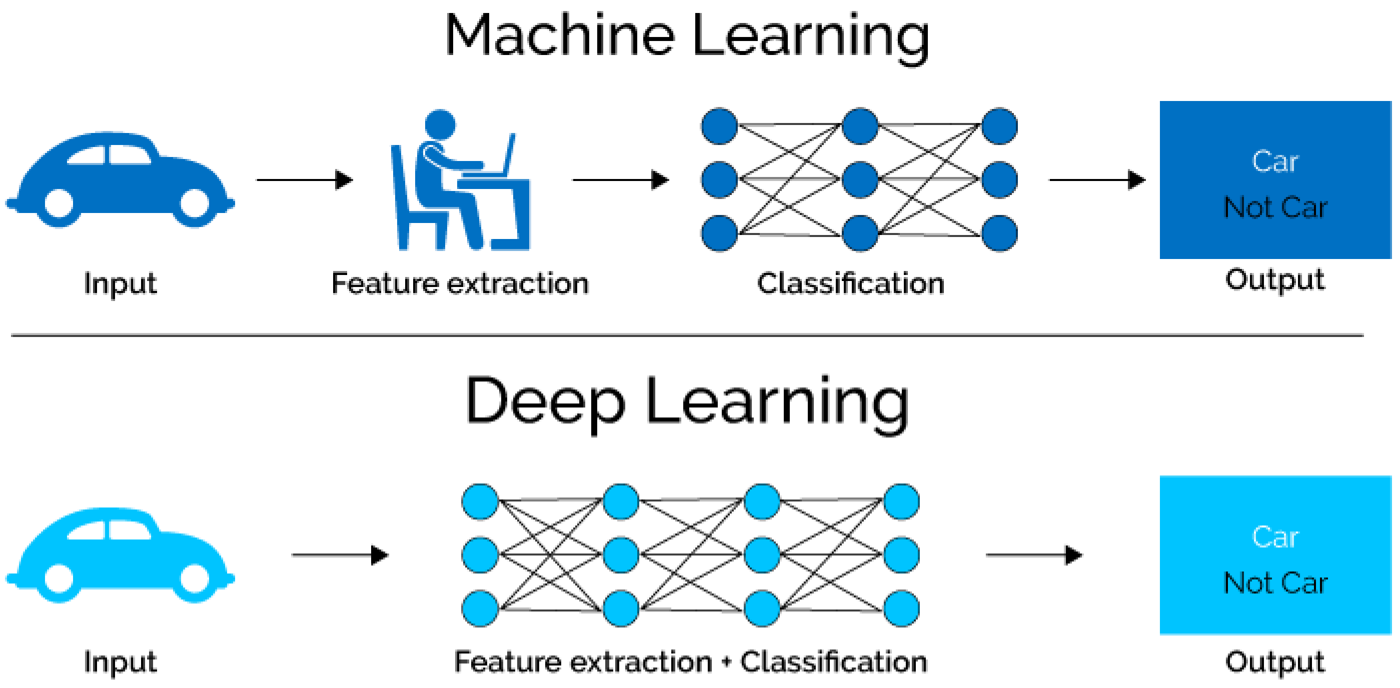
In Summary

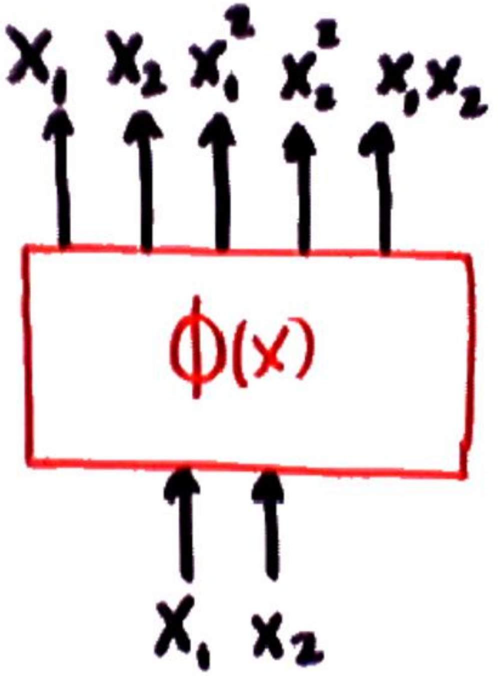
- **1st generation NNs:** Perceptron 1957 – 1969
  - ❖ Only useful for linearly separable examples

- **2nd generation NNs:** Feedforward networks and variants (convolutional, recurrent), beginning of 1980s to middle 1990s… <u>difficult to train</u>
  - ❖ Wrong activation functions
  - ❖ Subpar weight initialization
  - ❖ Too many parameters to train when computers were slower
  - ❖ Datasets were too small

- **3rd generation NNs:** Deep networks 2006-?
  - ❖ Newer approaches to train networks with multiple layers
  - ❖ Reap the rewards of flexible function approximators…

Flexible Function Approximators





Remember the idea of increasing model flexibility through **feature transformation**, *i.e.* replace $\mathbf{x}$ with $\phi(\mathbf{x})$?

Automatic Feature Extraction

Example **basis function expansion** in polynomial regression:

$$f(\mathbf{x};\boldsymbol{\theta}) = \underbrace{\mathbf{W}}_{\text{Weights}}\phi(\mathbf{x}) + \underbrace{\mathbf{b}}_{\text{Bias}} \qquad \theta = [W, b]$$

Handcrafting transformations is limiting; parameterize the feature extractor:

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathbf{W}\phi(\mathbf{x};\boldsymbol{\theta}^{(2)}) + \mathbf{b}$$

Repeat recursively to create increasingly more complex function hierarchies:

Deep NNs

$$f(\mathbf{x};\boldsymbol{\theta}) = f^{(L)}(f^{(L-1)}(\cdots(f^{(1)}(\mathbf{x}))\cdots))$$

Composition of L functions, where $f^{(l)}(\mathbf{x}) = f(\mathbf{x};\boldsymbol{\theta}^{(l)})$ is the function at layer l.

Feedforward Neural Networks

The following slides are taken and adapted from **Hugo Larochelle's** course:

https://info.usherbrooke.ca/hlarochelle/neural_networks/content.html

Artificial Neuron (1)

Topics: connection weights, bias, activation function

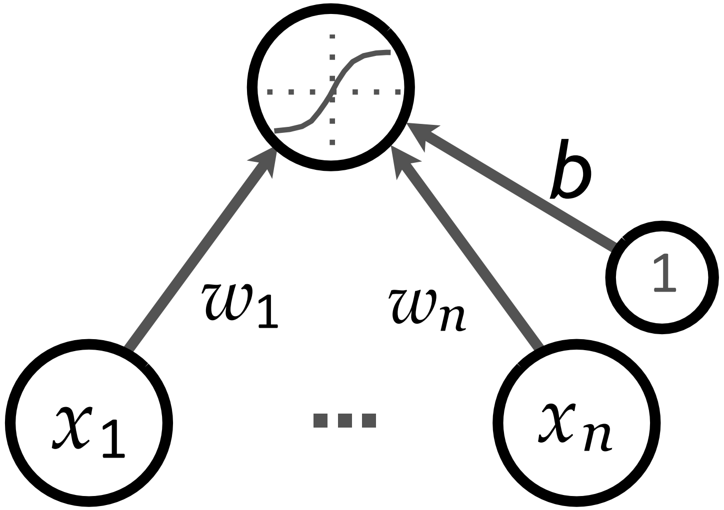
- Neuron **pre-activation** (or input activation):

$$a(\mathbf{x}) = b + \sum_{j}^{n} w_j x_j = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (**output**) activation

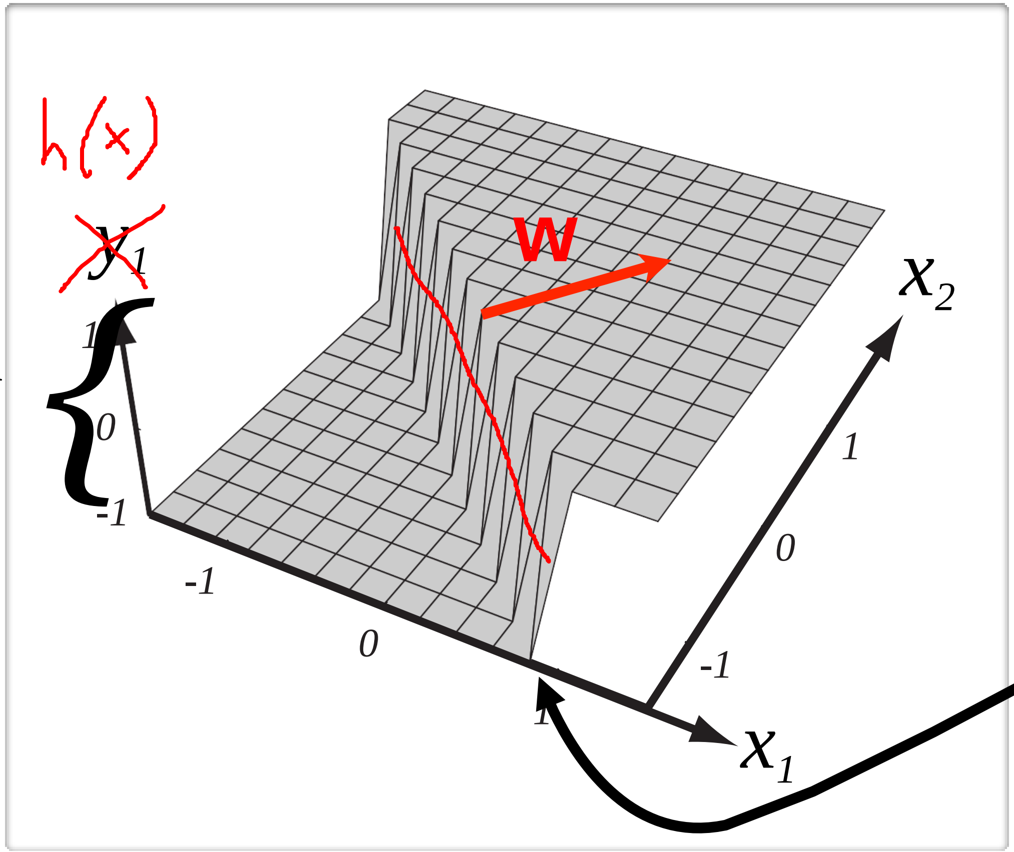
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \mathbf{w}^\top \mathbf{x})$$

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



Artificial Neuron (2)

Range determined by $g(\cdot)$

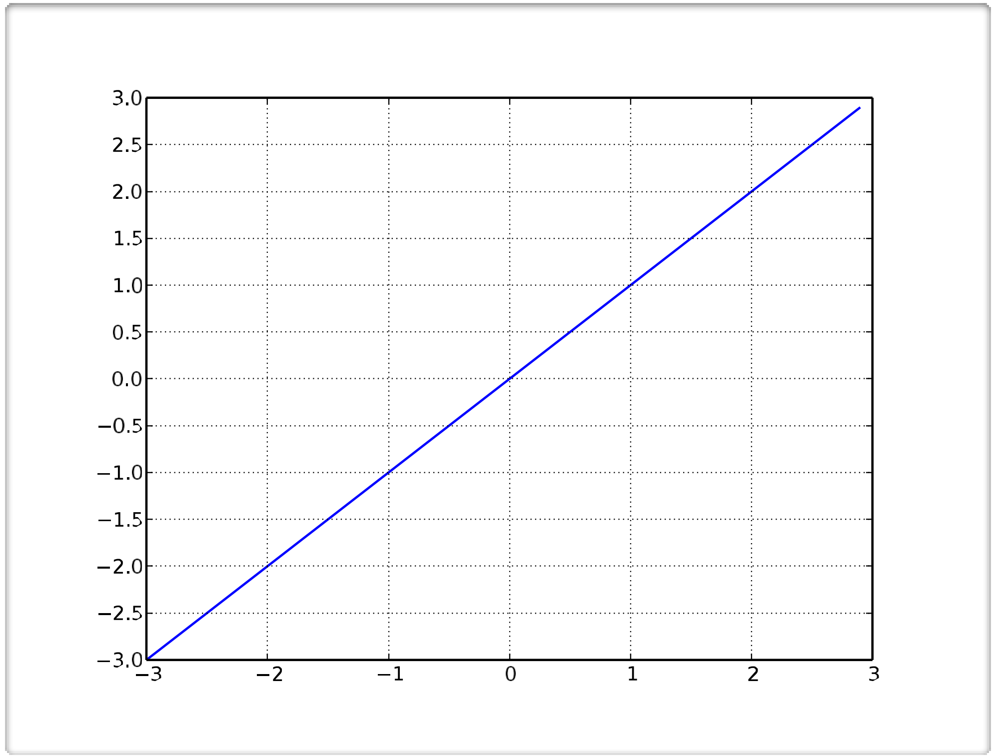


Bias b only changes the position of the ridge

Activation Functions – Linear

Linear activation function

- Performs no input squashing
- Not very interesting…

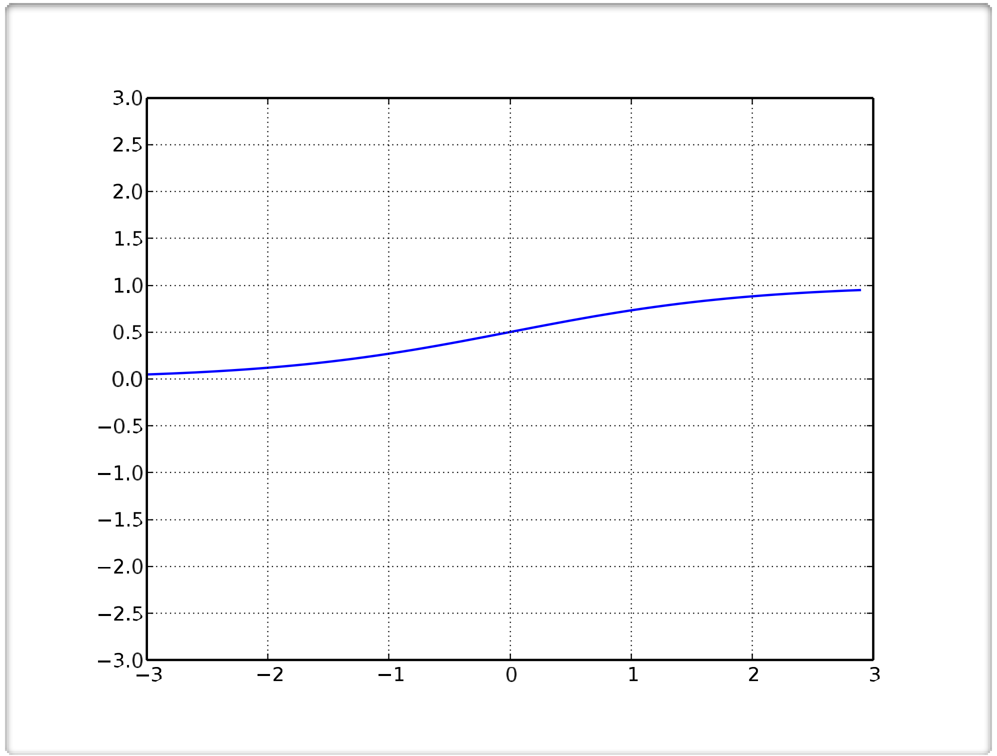


$$g(a) = a$$

Activation Functions – Sigmoid/Logistic

Sigmoid activation function

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

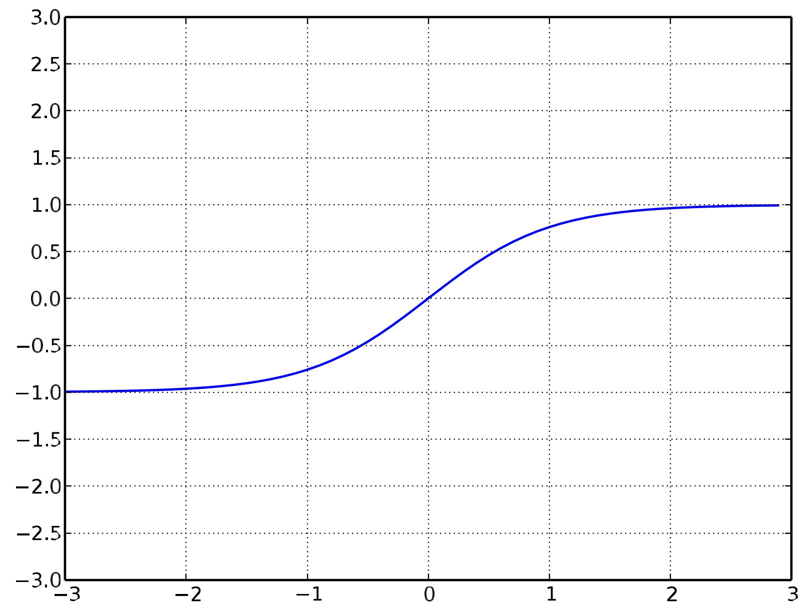


$$g(a) = \frac{1}{1 + e^{-a}}$$

Activation Functions – tanh

Hyperbolic tangent (**tanh**) activation function

- Squashes the neuron's pre-activation between -1 and 1
- Positive or negative
- Bounded
- Strictly increasing

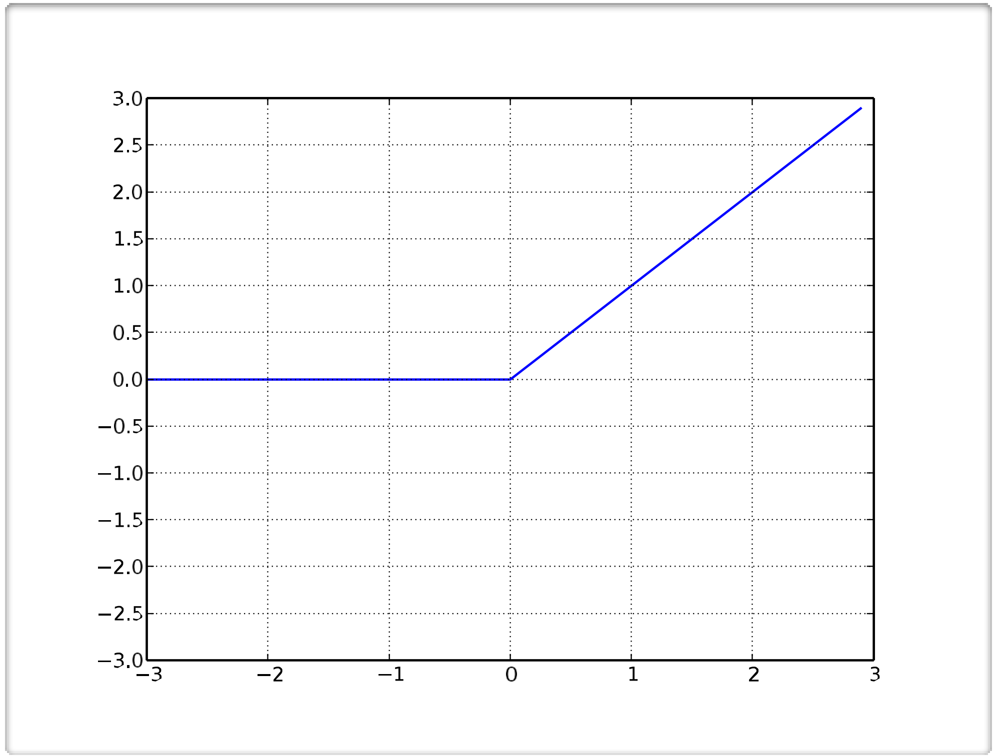


$$g(a) = \frac{e^{a} - e^{(-a)}}{e^{a} + e^{(-a)}} = \frac{e^{2a} - 1}{e^{2a} + 1}$$

Activation Functions – ReLU

Rectified linear activation function (ReLU)

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



$$g(a) = \mathrm{relu}(a) = \max(0, a)$$

Linear Capacity of an Artificial Neuron

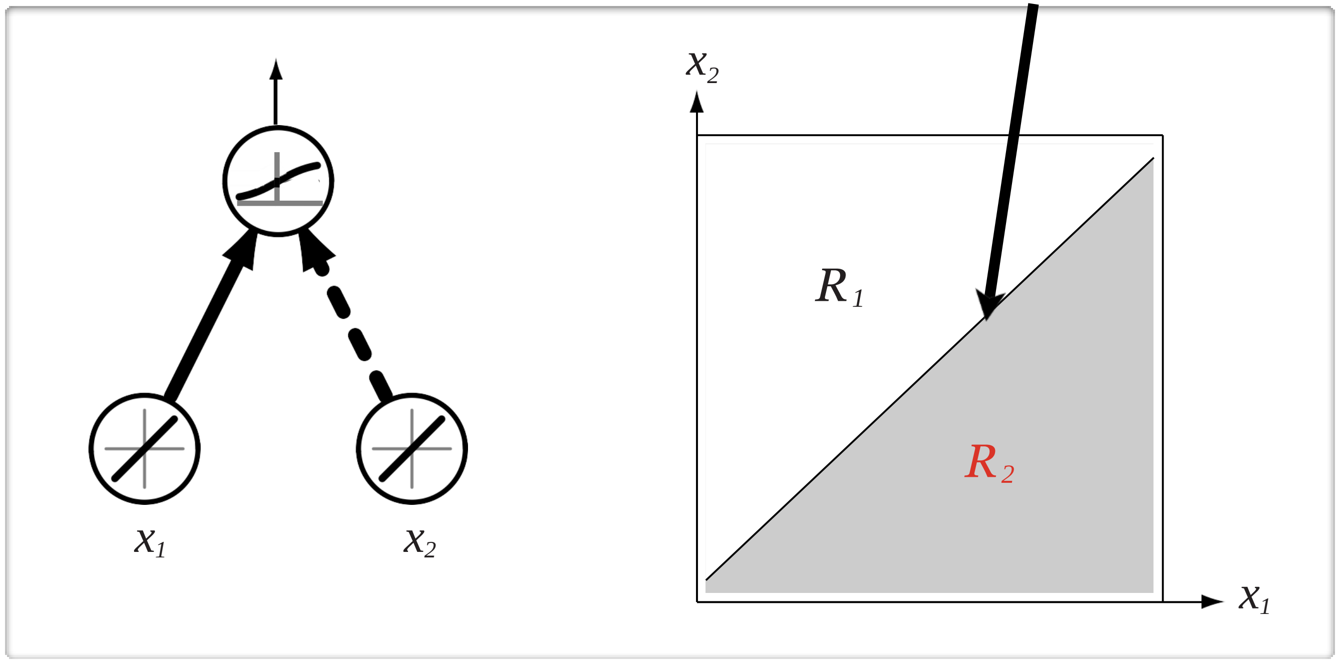
Topics: capacity, decision boundary of neuron

- Could do binary classification:
  - with sigmoid, interpret neuron as estimating $p(y = 1|\mathbf{x})$

decision boundary is linear

- Aka logistic regression
- if greater than 0.5, predict class 1
- otherwise, predict class 0

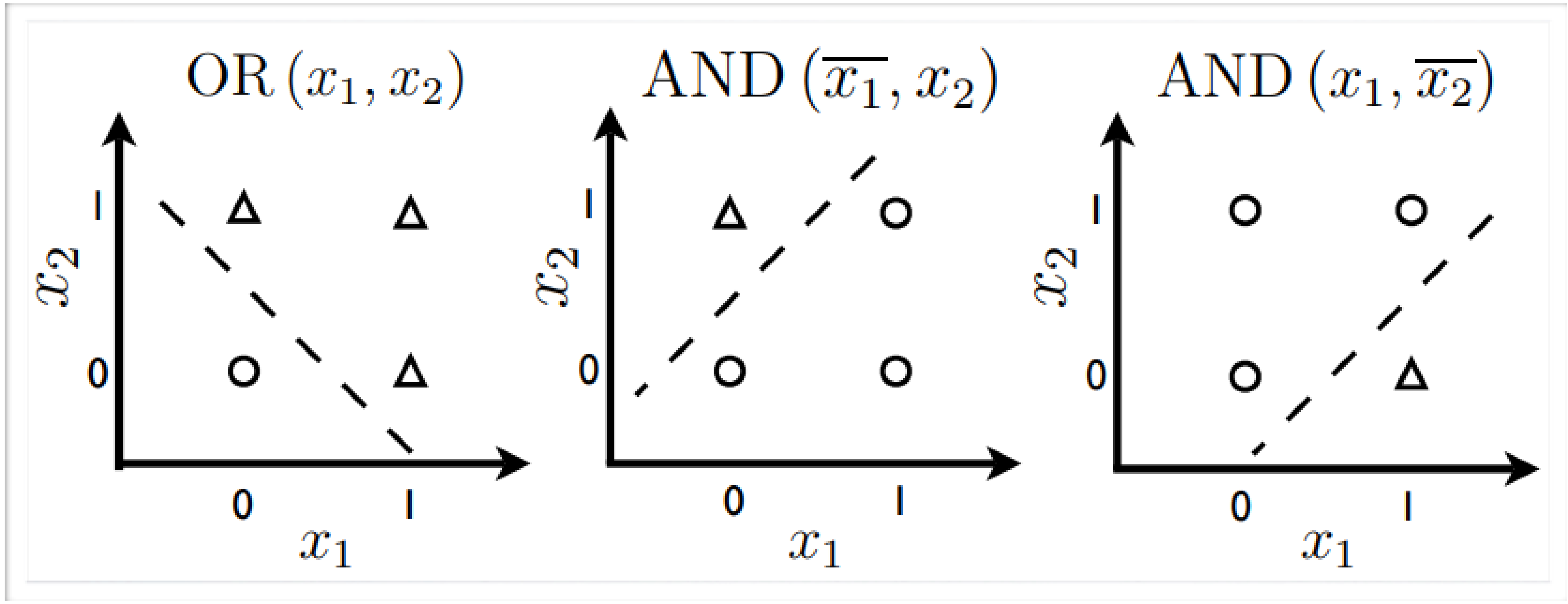
Similar idea can be applied with tanh



Artificial Neuron for Basic Logic Gates

Topics: capacity of a single neuron

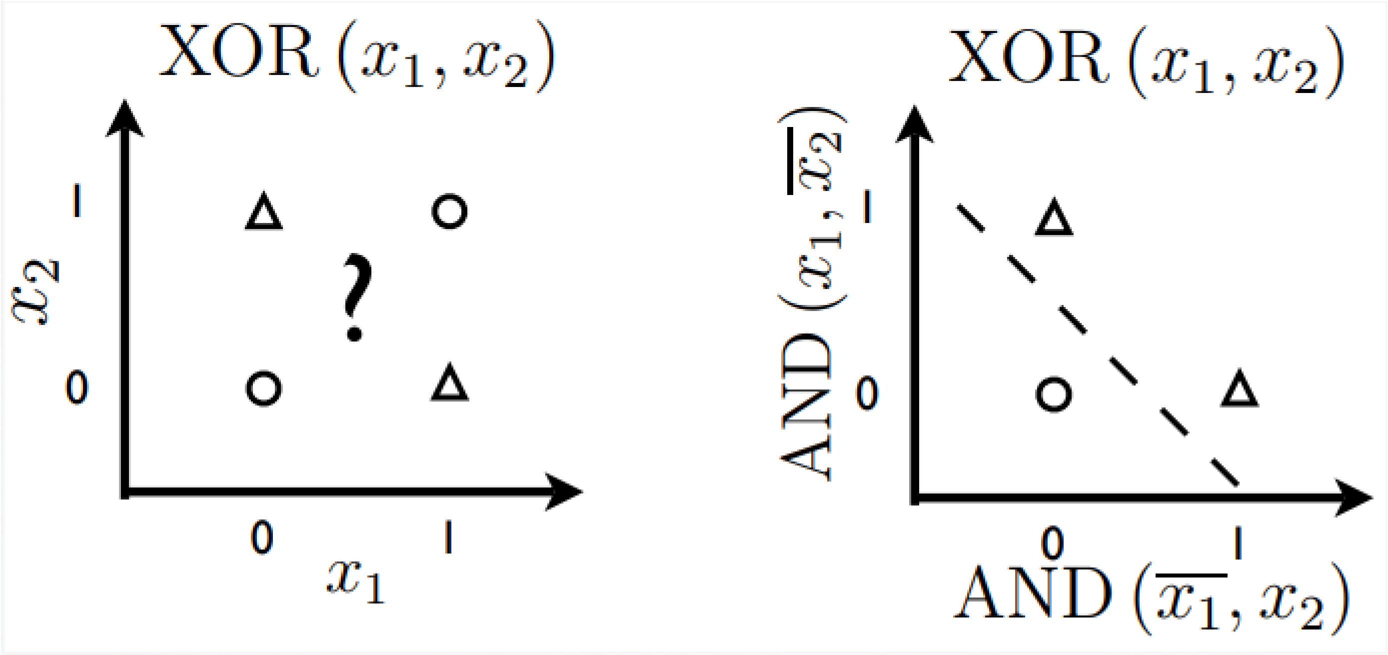
- Can solve linearly separable problems



Artificial Neuron NOT for Non-linear Problems

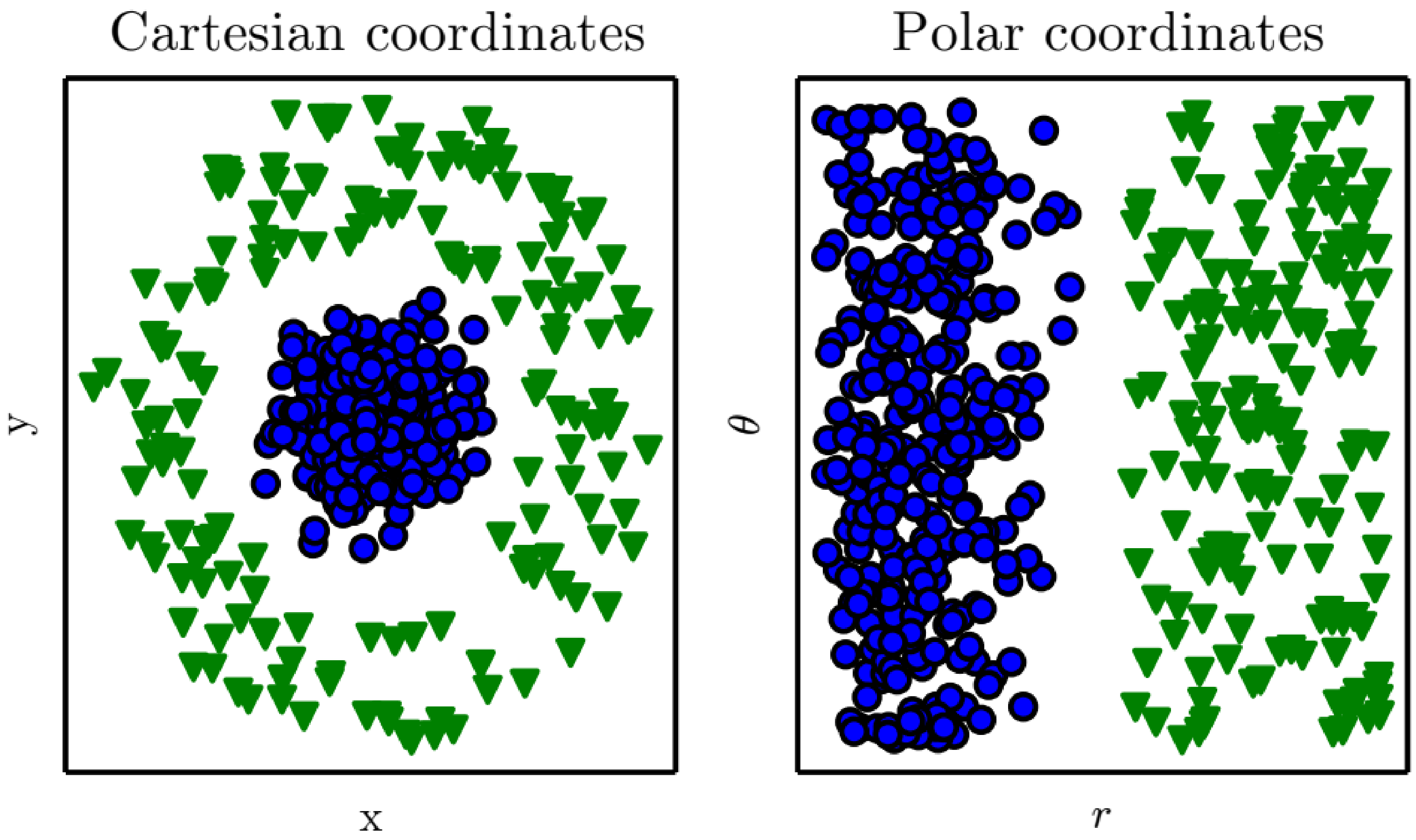
Topics: capacity of a single neuron

- CAN’T solve non-linear separable problems



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

Representation Matters



Goodfellow et al., *"Deep Learning"*, 2016

Neural Network – Single Layer

Topics: single-layer neural network (NN)

- Hidden layer pre-activation:

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

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

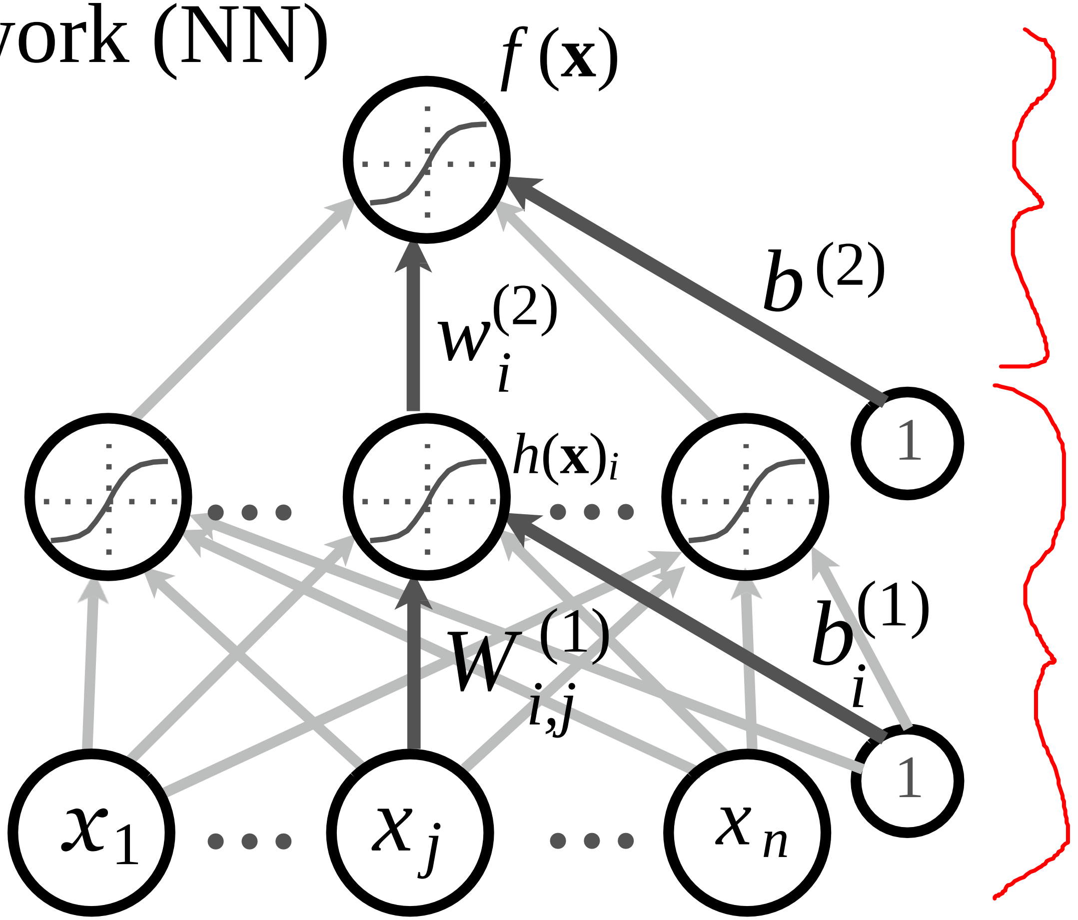
- Hidden layer activation:

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

- Output layer activation:

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

Not g



Neural Network – Softmax Activation

Topics: softmax activation function

- For multi-class classification:
  - need multiple outputs (one per class)
  - wish to estimate conditional probability $p(y = c \,|\mathbf{x})$

- Use **softmax** activation at the output:
  - strictly positive
  - sums to one

$$\mathcal{S}(\boldsymbol{a}) \triangleq \left[\frac{e^{a_1}}{\sum_{c'=1}^{C} e^{a_{c'}}}, \ldots, \frac{e^{a_C}}{\sum_{c'=1}^{C} e^{a_{c'}}} \right] = \sigma(a)$$

…Predicted class is one with highest estimated probability

Neural Network – Multilayer Neural Network

MLP

Topics: multilayer NN

- Could have L hidden layers
- Layer pre-activation for $l > 0$:

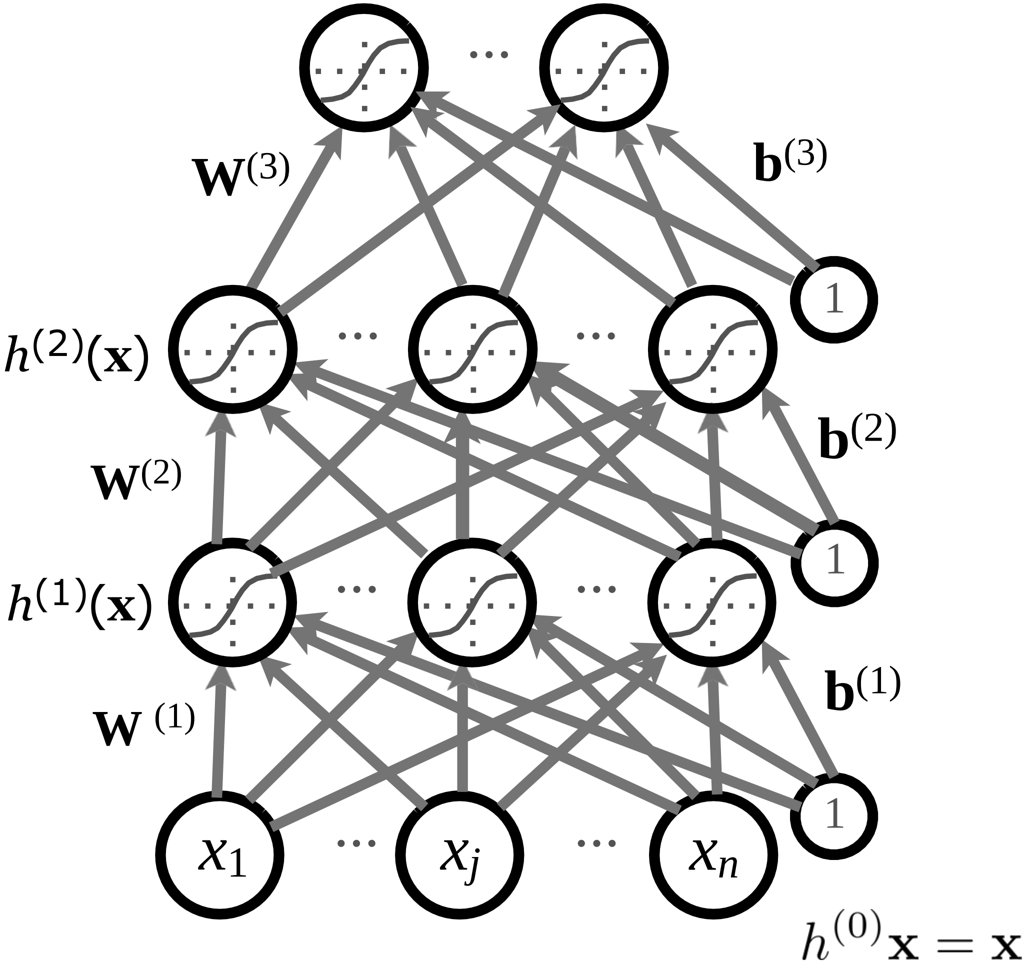
$$a^{(l)}(\mathbf{x}) = \mathbf{b}^{(l)} + \mathbf{W}^{(l)} h^{(l-1)} \mathbf{x}$$

- Hidden layer activation ($l = 1, \ldots, L$):

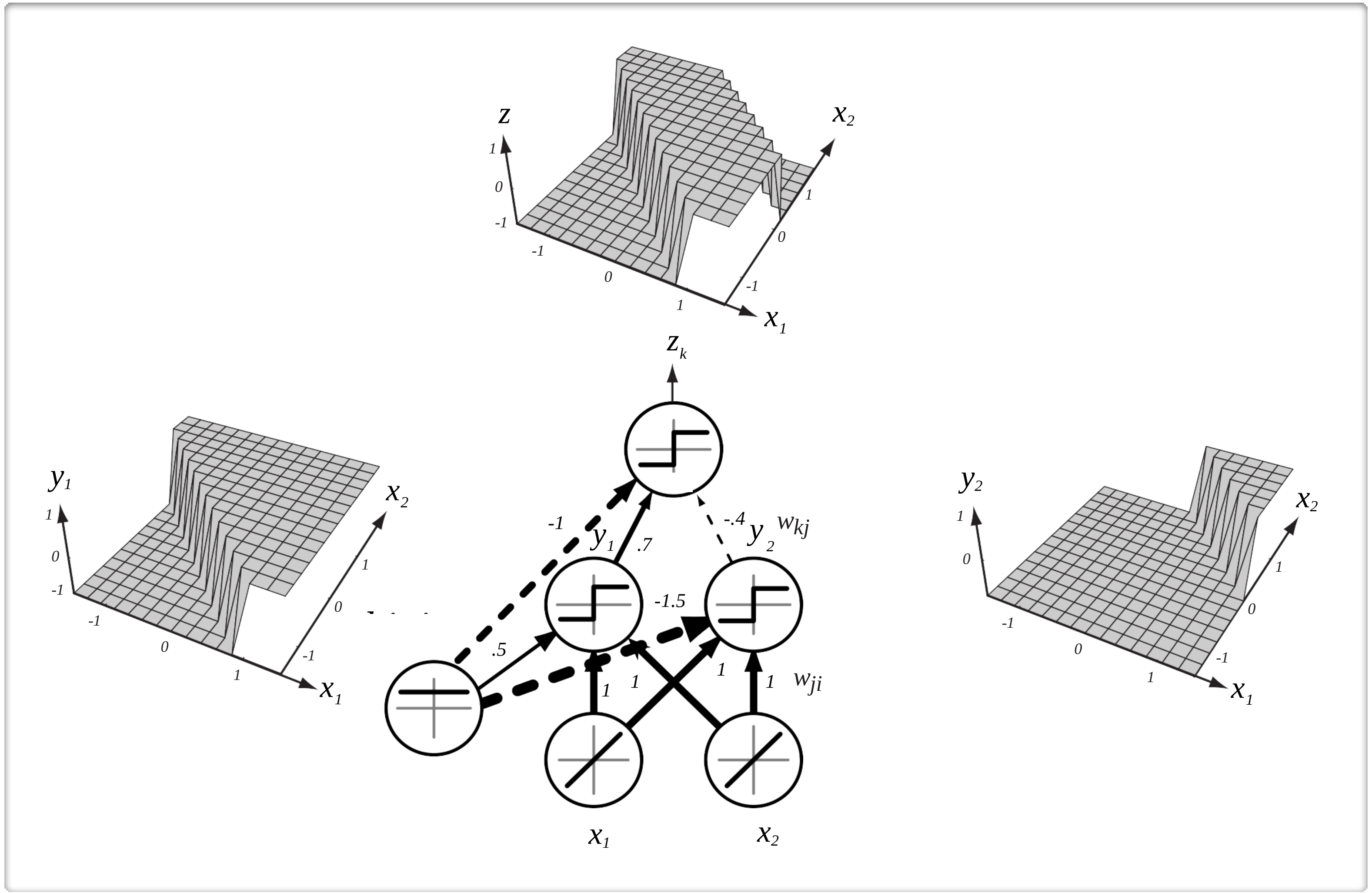
$$h^{(l)}(\mathbf{x}) = g(a^{(l)}(\mathbf{x}))$$

- Output layer activation ($l = L + 1$):

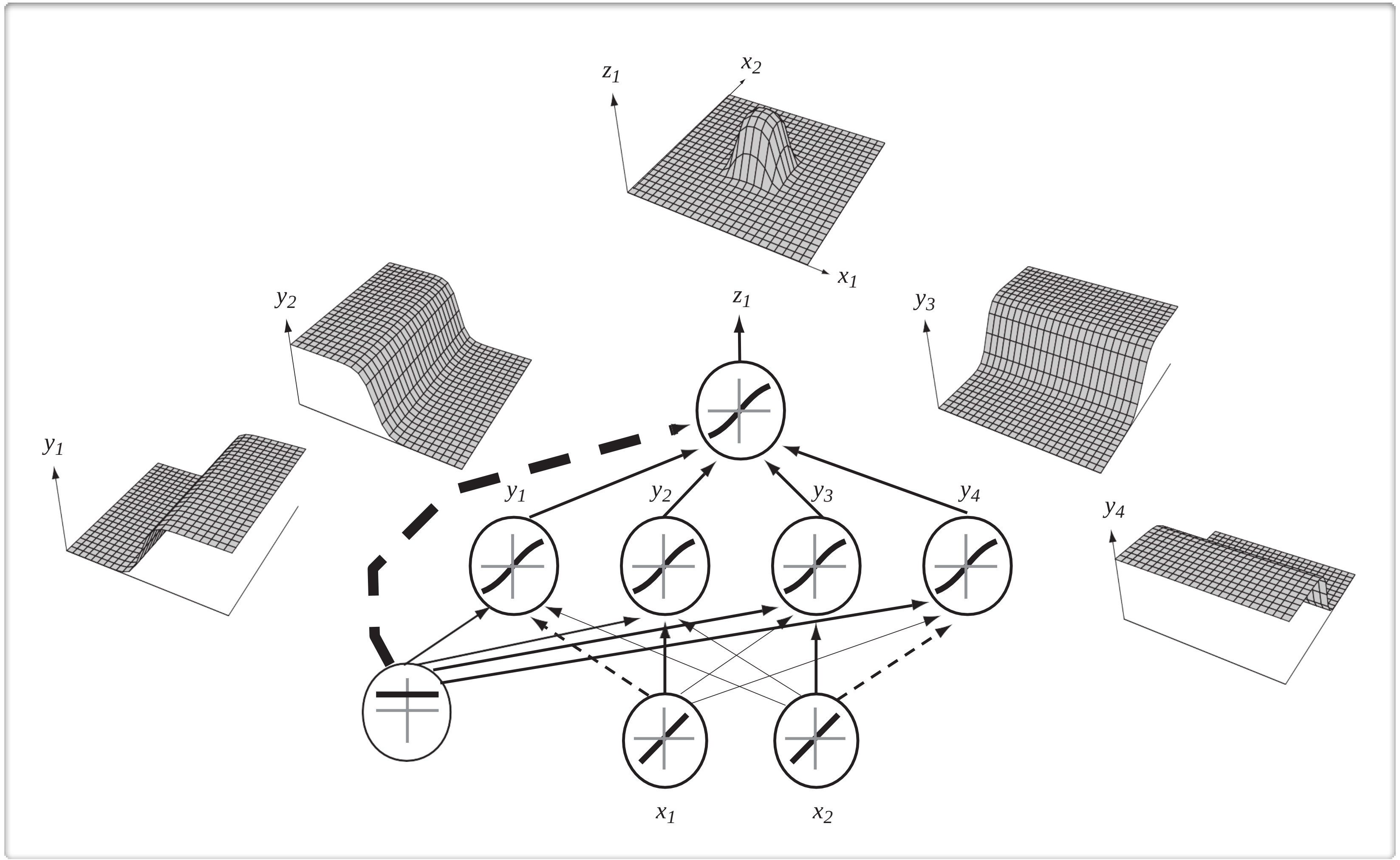
$$h^{(L+1)}(\mathbf{x}) = o(a^{(L+1)}(\mathbf{x})) = f(\mathbf{x})$$



Capacity of a Single Layer NN (1)



Capacity of a Single Layer NN (2)



Universal Approximation Theorem

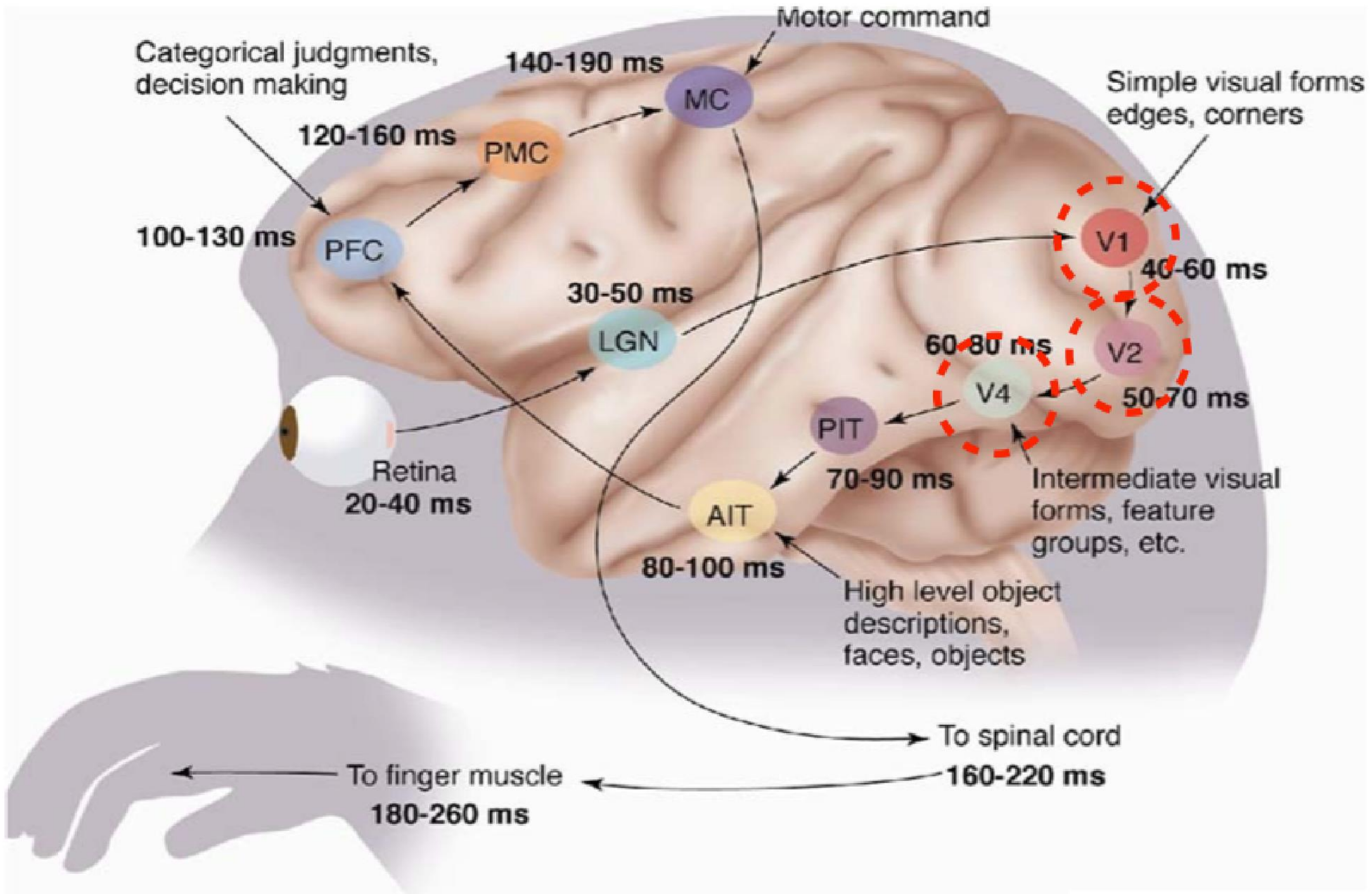
Topics: universal approximators

- “A single hidden layer NN with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units” (Hornik, 1991)
- Result applies to many other hidden layer activation functions, e.g., sigmoid, tanh, etc.

- A good result but there is no guarantee that a learning algorithm exists to derive the parameters for this arbitrarily complex function approximator

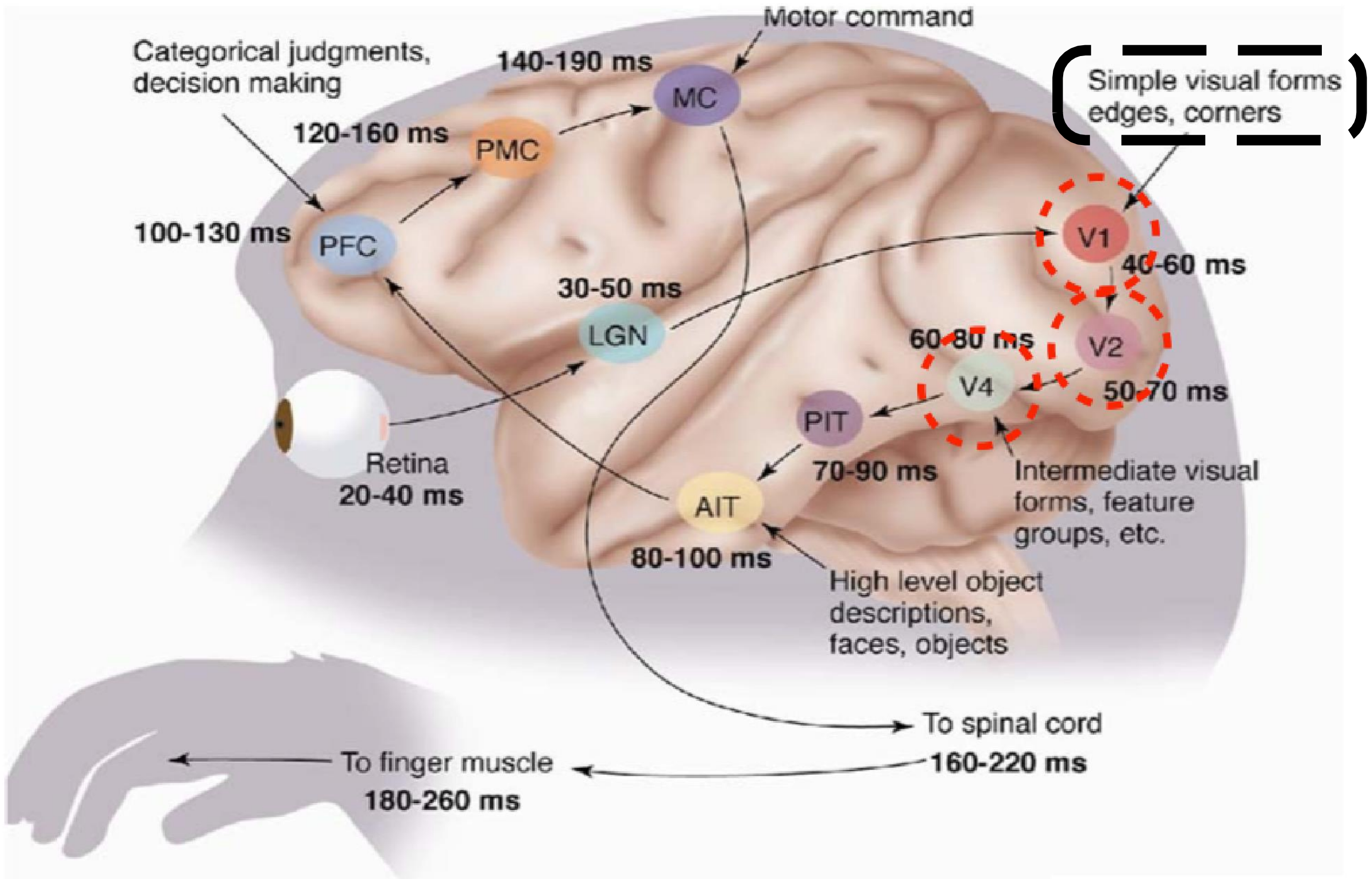
Coding Break

Parallel with Visual Cortex (1)



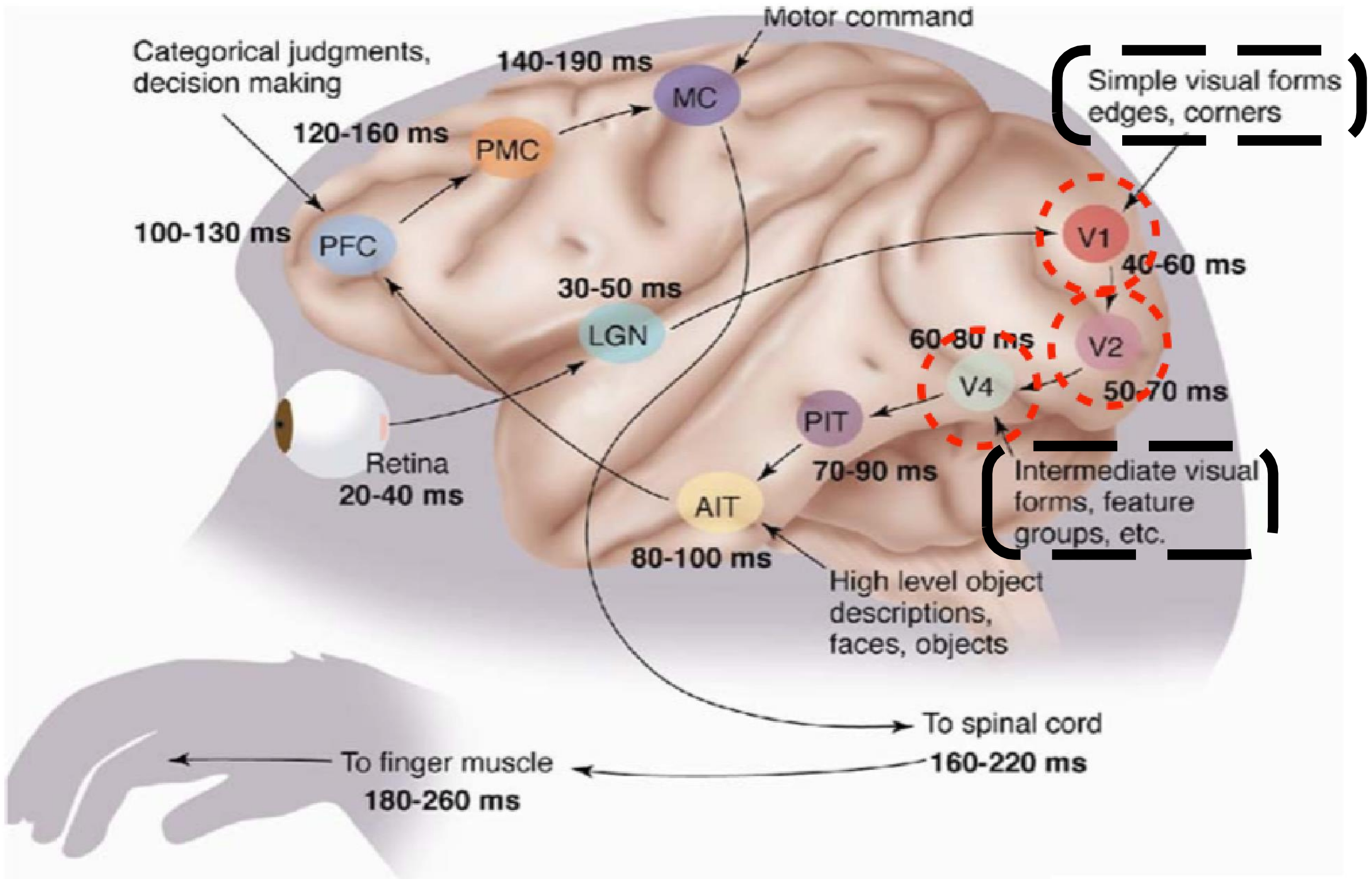
[picture from Simon Thorpe]

Parallel with Visual Cortex (2)



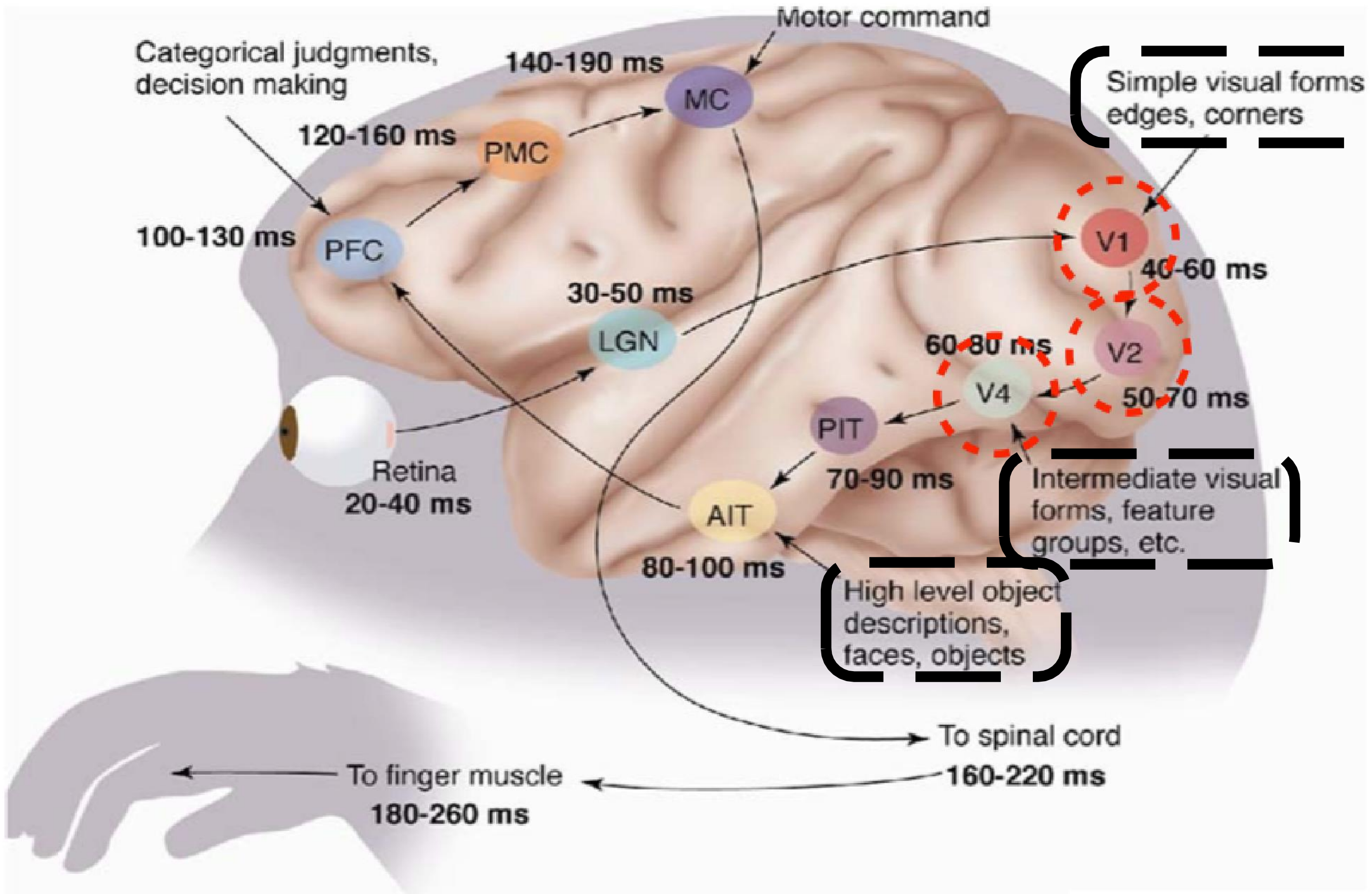
[picture from Simon Thorpe]

Parallel with Visual Cortex (3)



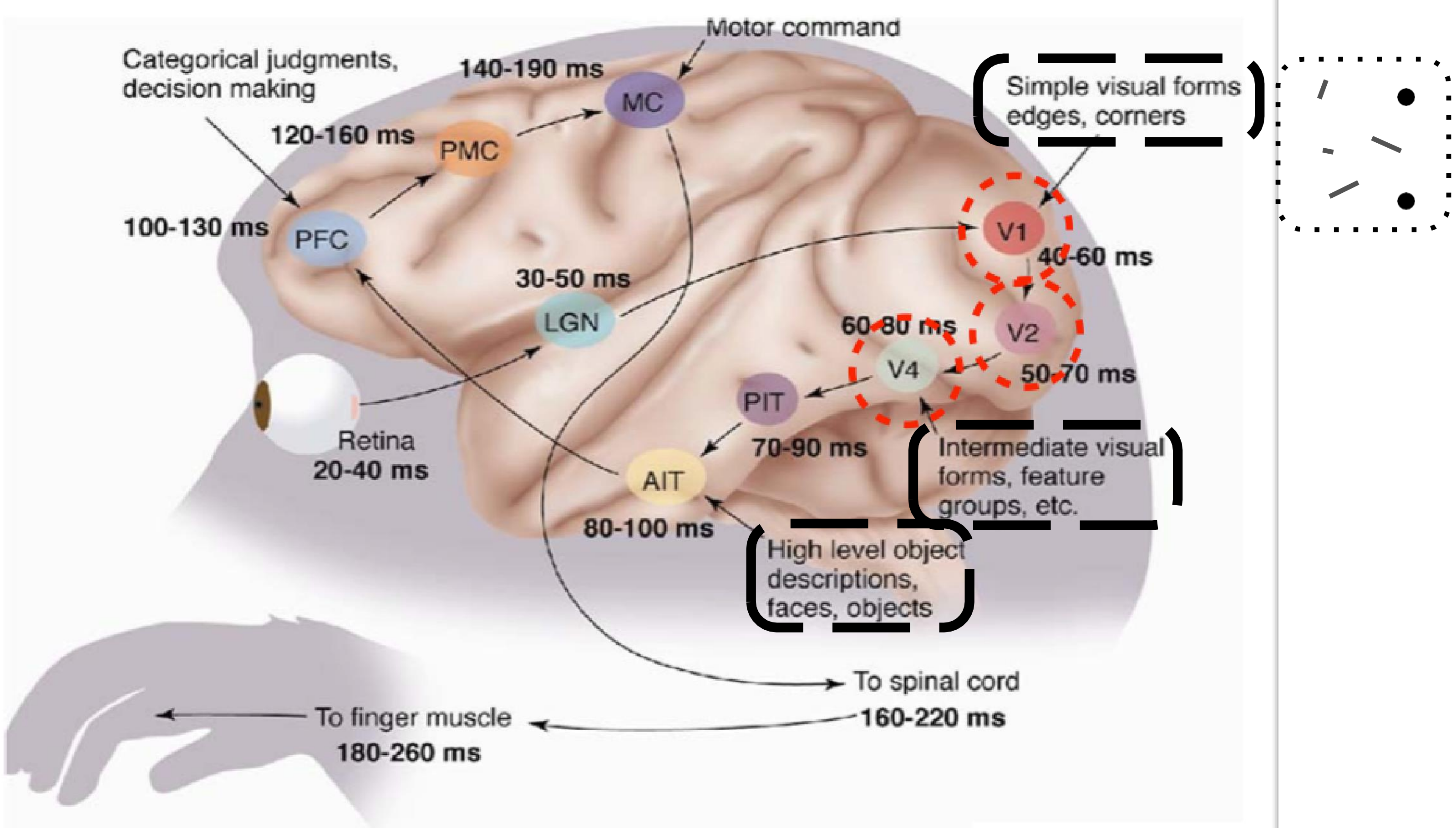
[picture from Simon Thorpe]

Parallel with Visual Cortex (4)



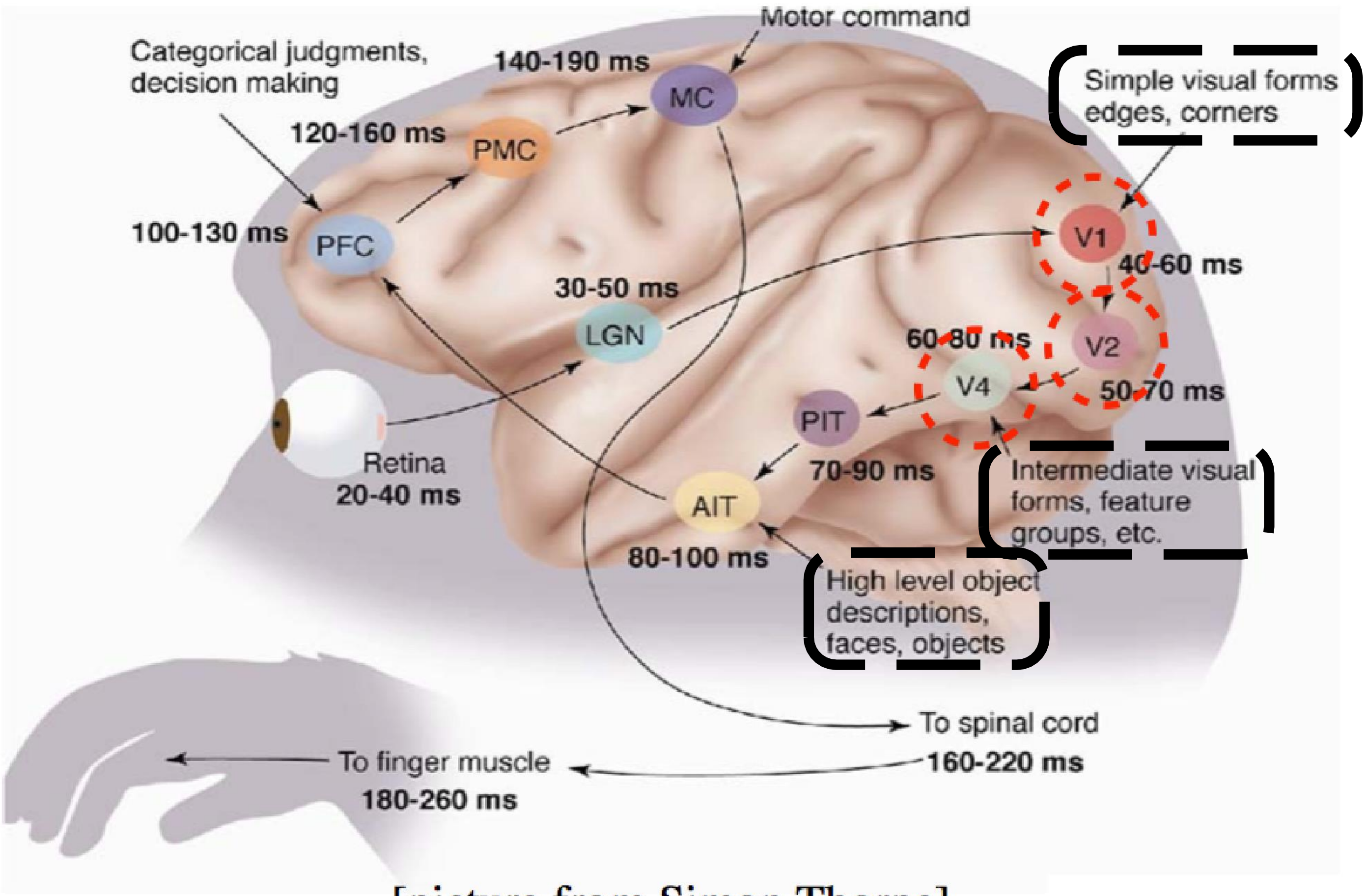
[picture from Simon Thorpe]

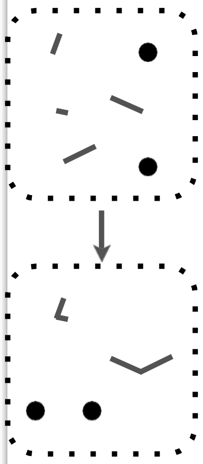
Parallel with Visual Cortex (5)



[picture from Simon Thorpe]

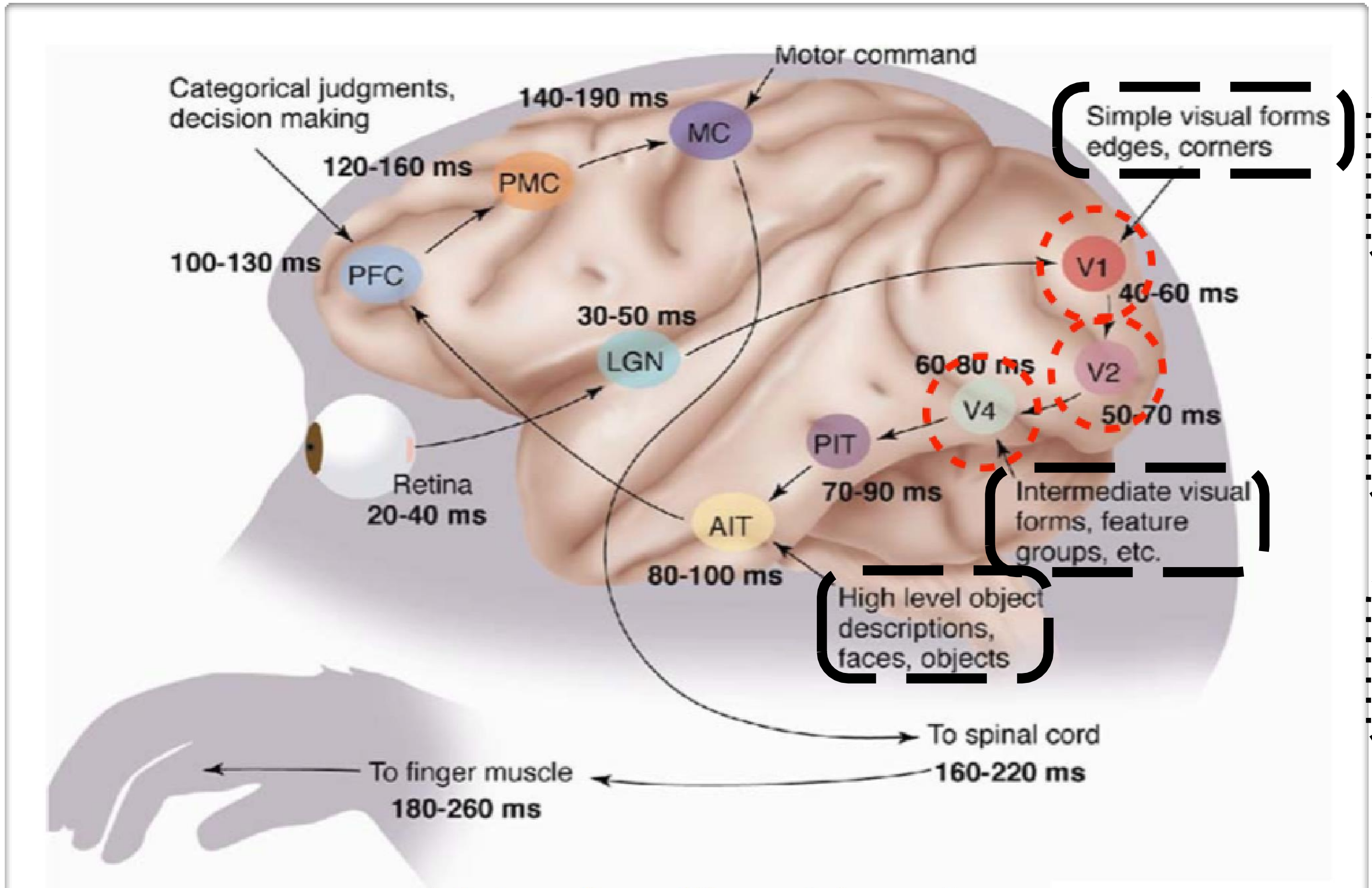
Parallel with Visual Cortex (6)



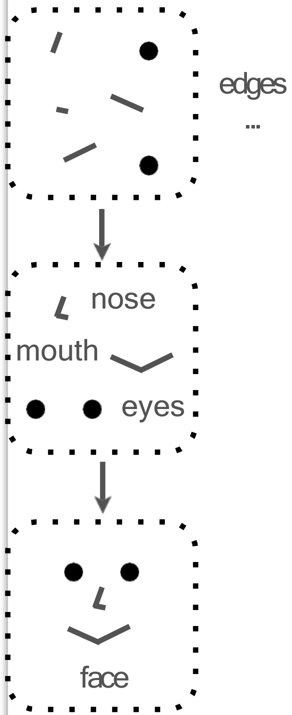


[picture from Simon Thorpe]

Parallel with Visual Cortex (7)



[picture from Simon Thorpe]



Concluding Remarks

- Introduction to **artificial neural networks**! The most popular computational system in machine learning nowadays

- Loads of material out there; but get comfortable with **PyTorch** first

- Quick example code of MLPs with PyTorch:

https://github.com/mazrk7/EECE5644_IntroMLPR_LectureCode/blob/main/notebooks/neural_networks/mlp_pytorch.ipynb

- Questions?