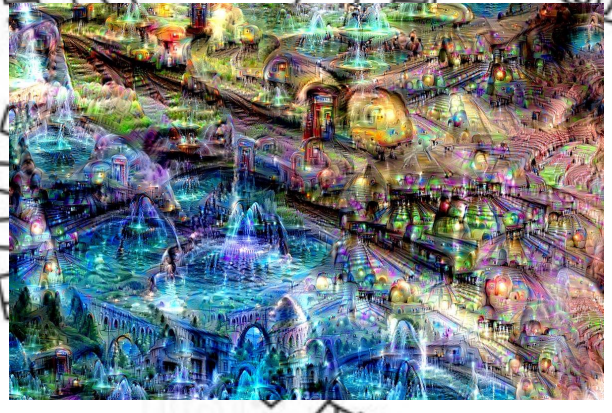
IMAGE DATA ANALYSIS

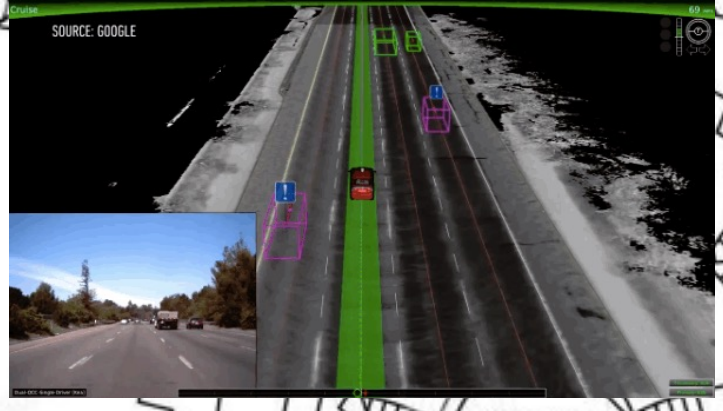
A.Y. 2021/22

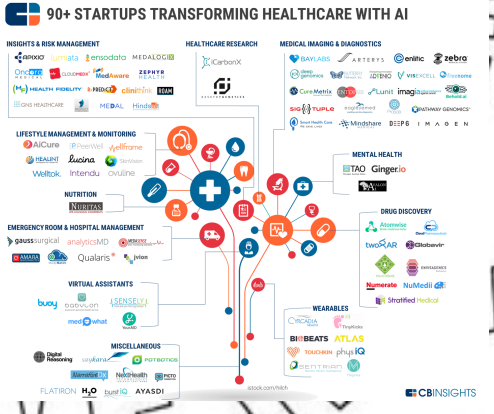
MASTER OF SCIENCE IN COMMUNICATION TECHNOLOGIES AND MULTIMEDIA

PROF. ALBERTO SIGNORONI - MATTIA SAVARDI – MATTIASAVARDI@UNIBS.IT

TOWARDS DEEPLEARNING





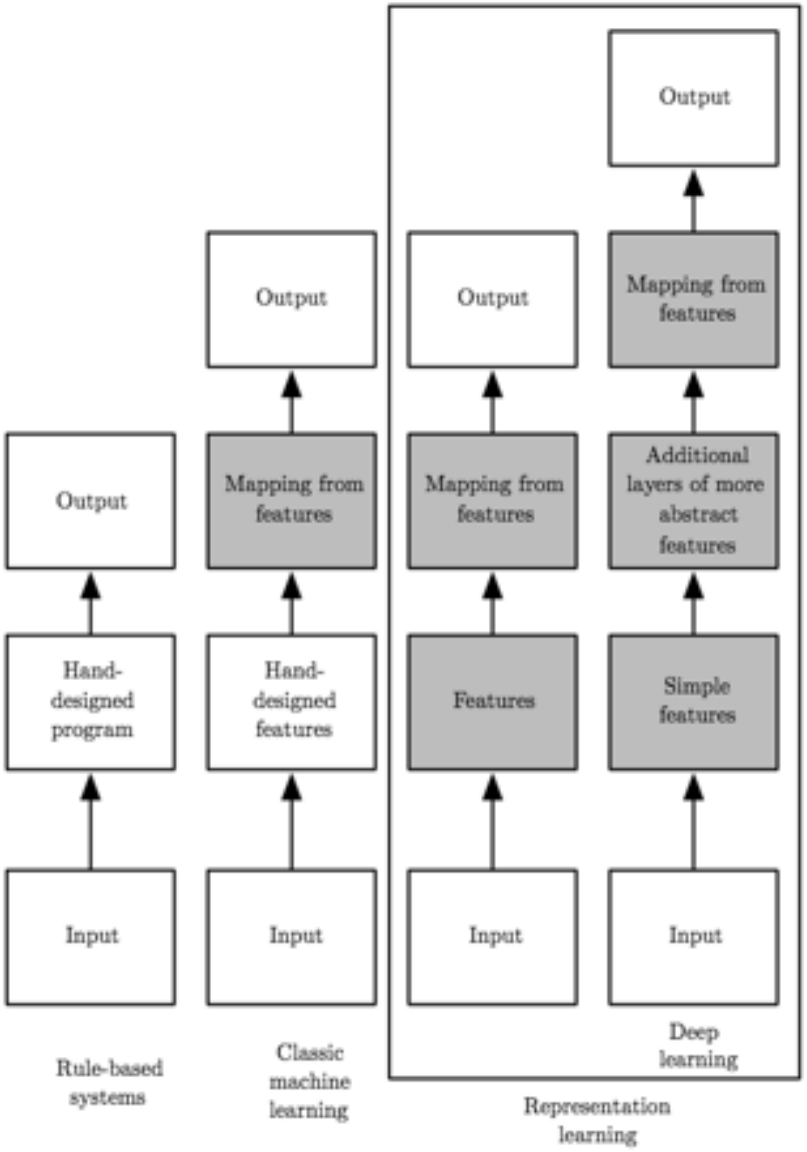


NEURAL NETWORKS INTRODUCTION

Useful resources

1. Deep Learning By — Ian Good fellow and Yoshua Bengio and Aaron Courville (http://www.deeplearningbook.org/)
2. Stanford CS231n course (http://cs231n.github.io/)
3. Deep Learning Tutorial – University of Montreal http://deeplearning.net/tutorial/deeplearning.pdf
4. http://neuralnetworksanddeeplearning.com/
5. News feed https://www.quora.com/topic/Deep-Learning
6. Coursera DeepLearning courses

Knowledge-base vs Representation learning



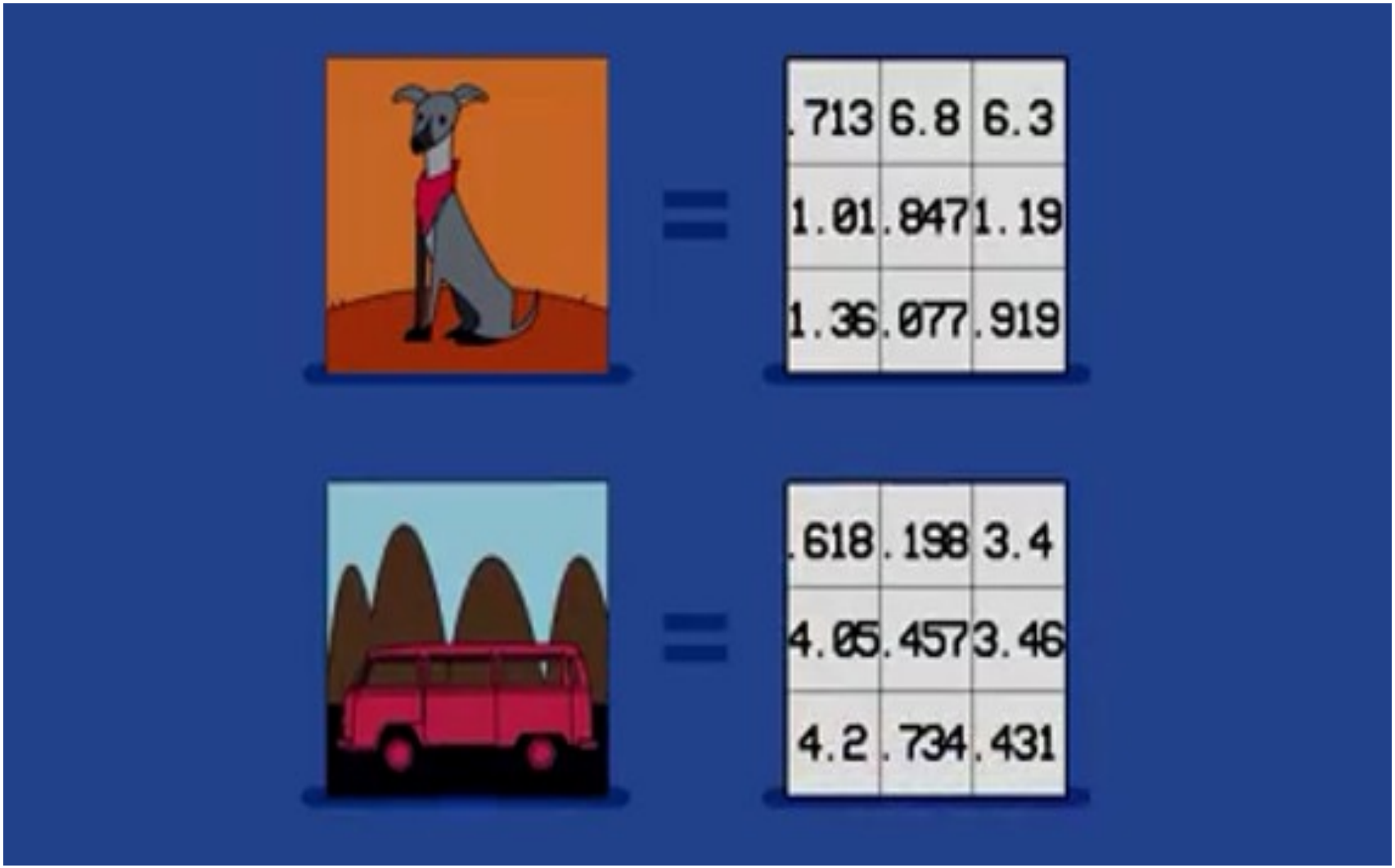
Knowledge-base vs Representation learning approaches

[deeplearning book]

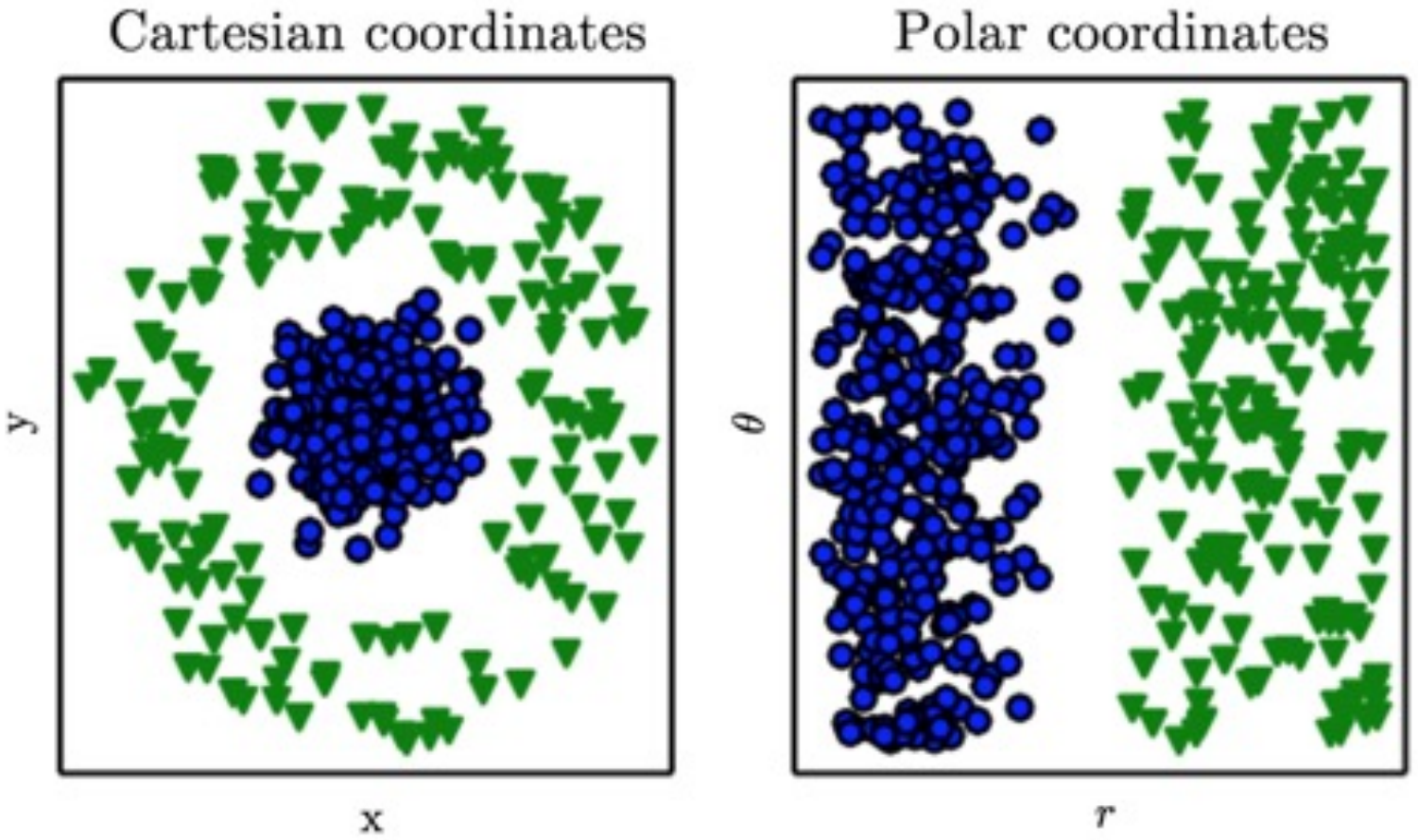
Knowledge-base vs Representation learning

[Y LeCun]

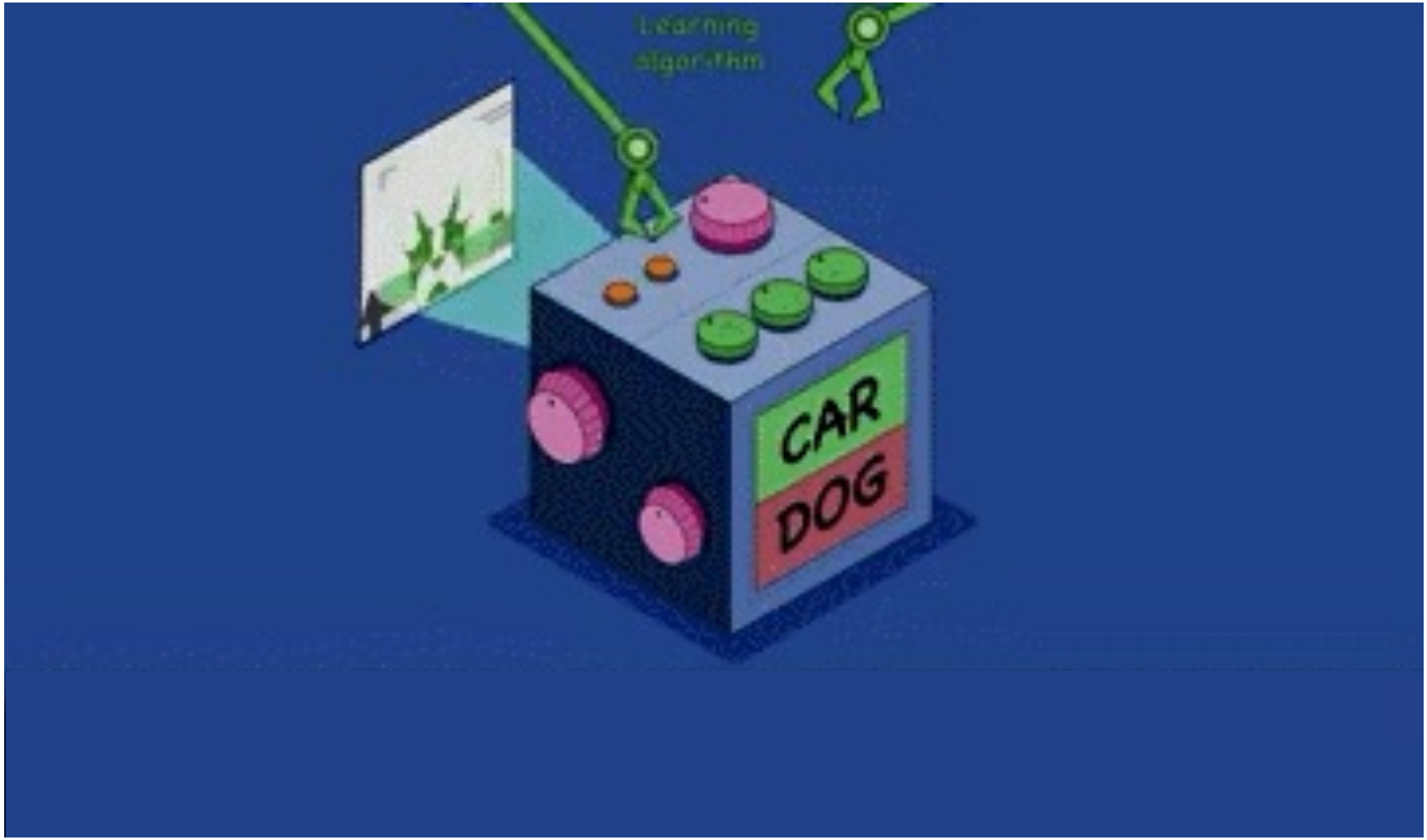
Knowledge-base vs Representation learning



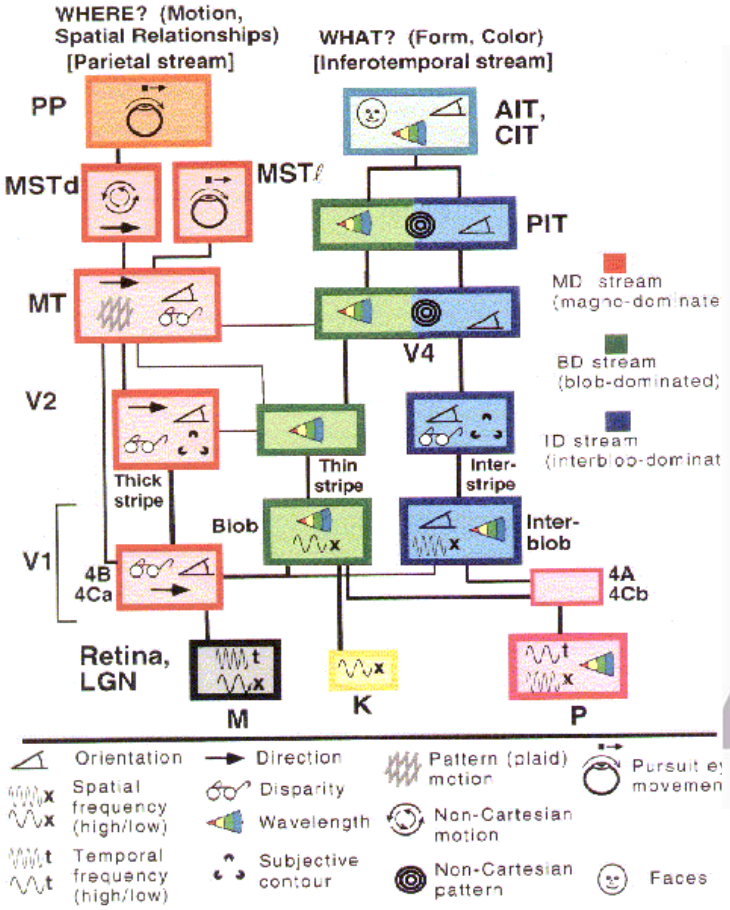
Knowledge-base vs Representation learning

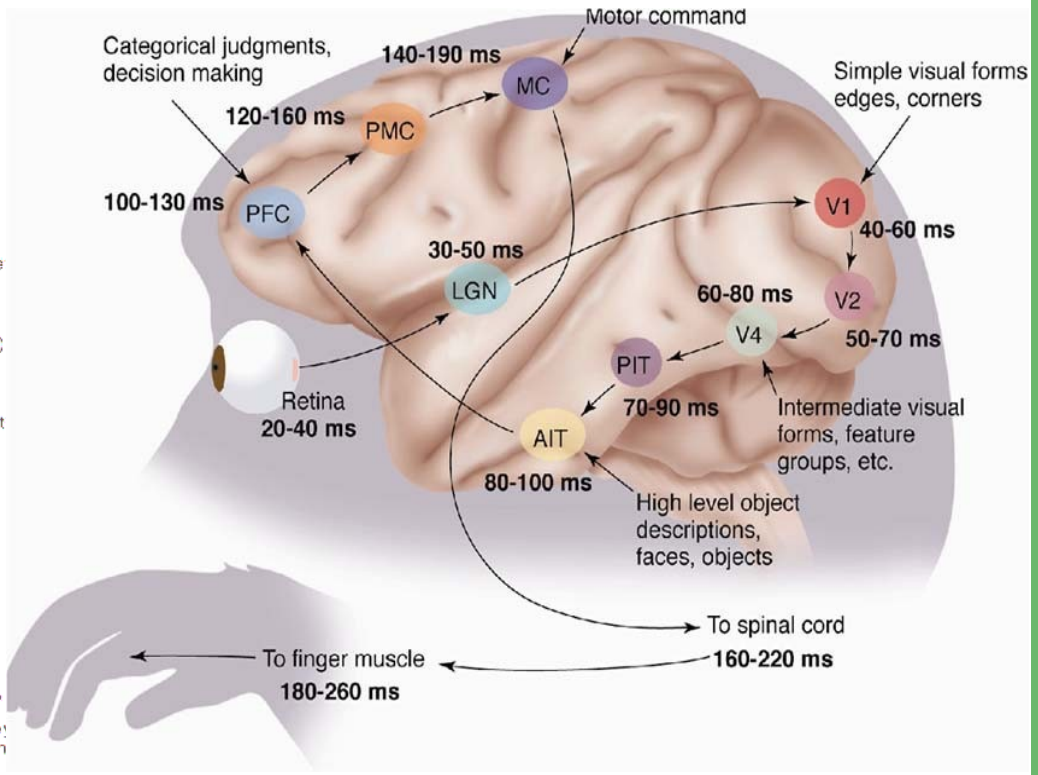


Knowledge-base vs Representation learning



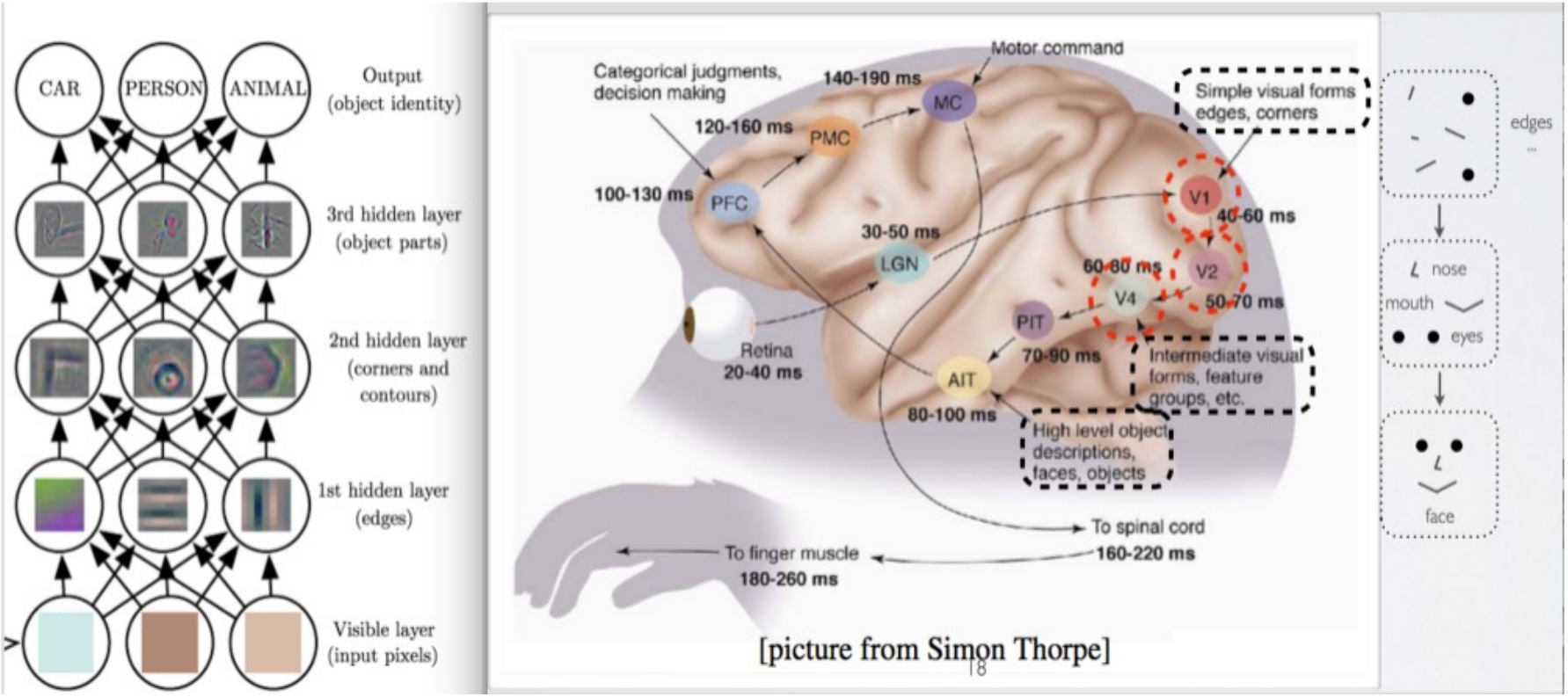
The mammalian visual cortex is hierarchical





[picture from Simon Thorpe]
[Gallant & Van Essen]

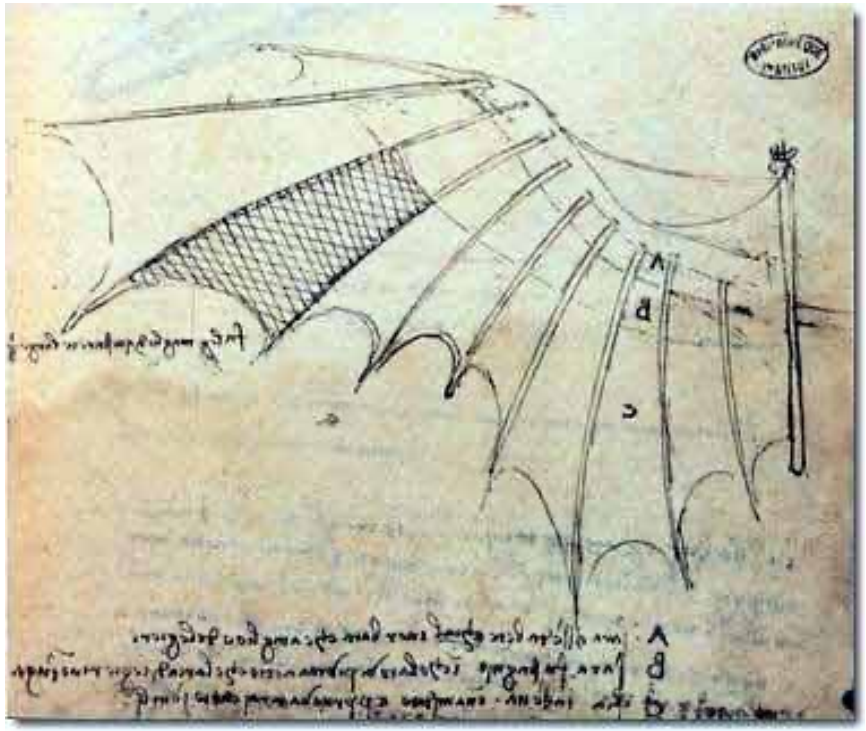
The mammalian visual cortex is hierarchical



[picture from Simon Thorpe]

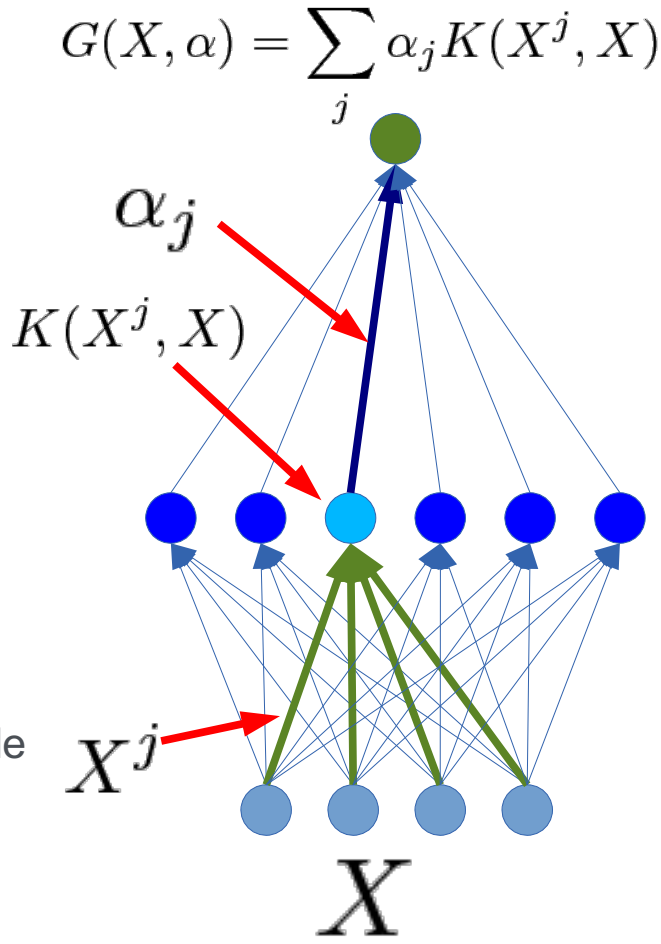
Let's be inspired by nature, but not too much

- It's nice imitate Nature,
- But we also need to understand
  - How do we know which details are important?
  - Which details are merely the result of evolution, and the constraints of biochemistry?
- For airplanes, we developed aerodynamics and compressible fluid dynamics.
  - We figured that feathers and wing flapping weren't crucial
- QUESTION: What is the equivalent of aerodynamics for understanding intelligence?



Which models are deep?

- 2-layer models are not deep (even if you train the first layer)
  - Because there is no feature hierarchy
- Neural nets with 1 hidden layer are not deep
- SVMs and Kernel methods are not deep
  - Layer1: kernels; layer2: linear
  - The first layer is “trained” in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
- Classification trees are not deep
  - No hierarchy of features. All decisions are made in the input space



Do we really need deep architectures?

- Theoretician's dilemma: "We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?"

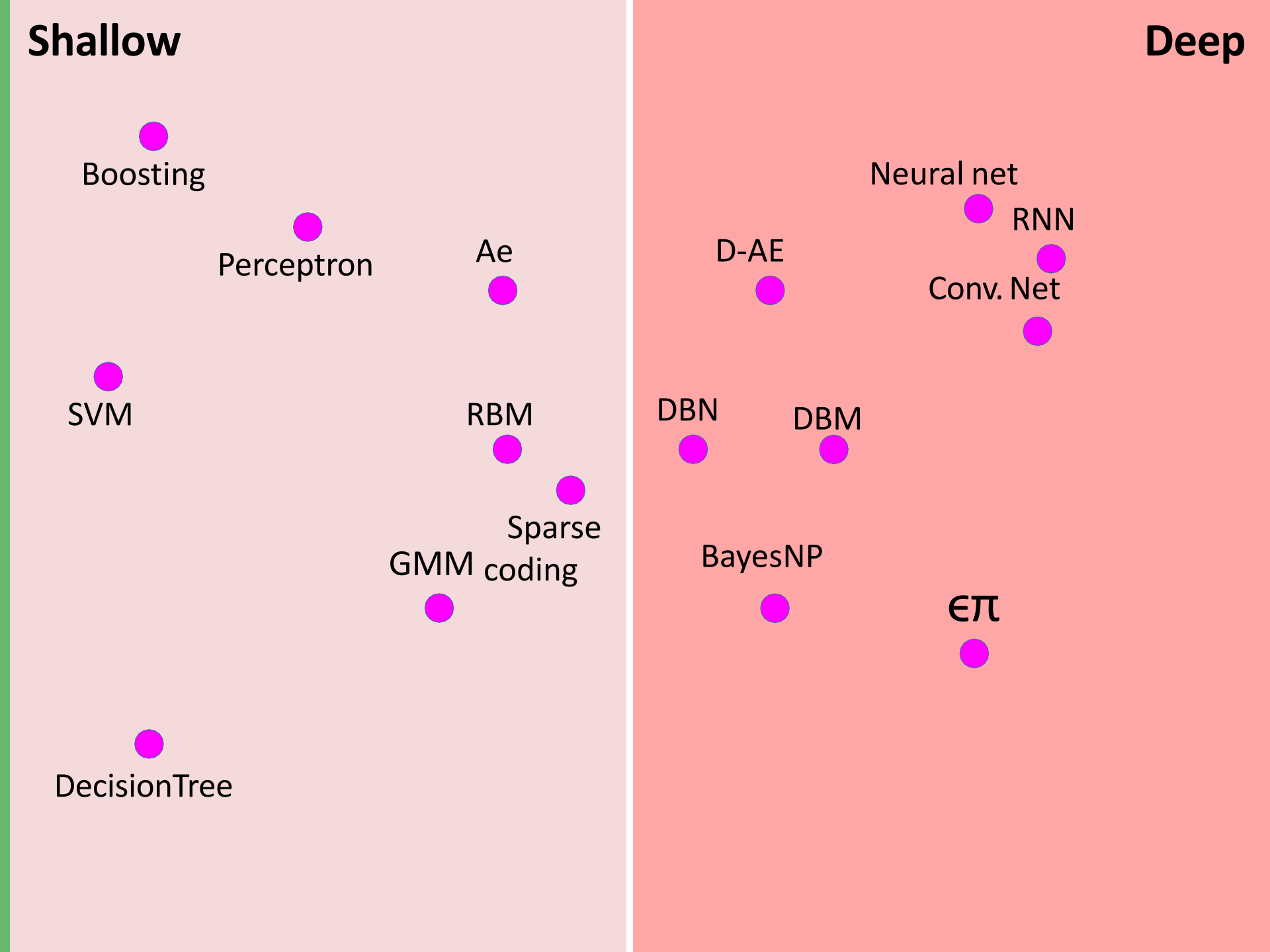
$$y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \qquad y = F(W^1.F(W^0.X))$$

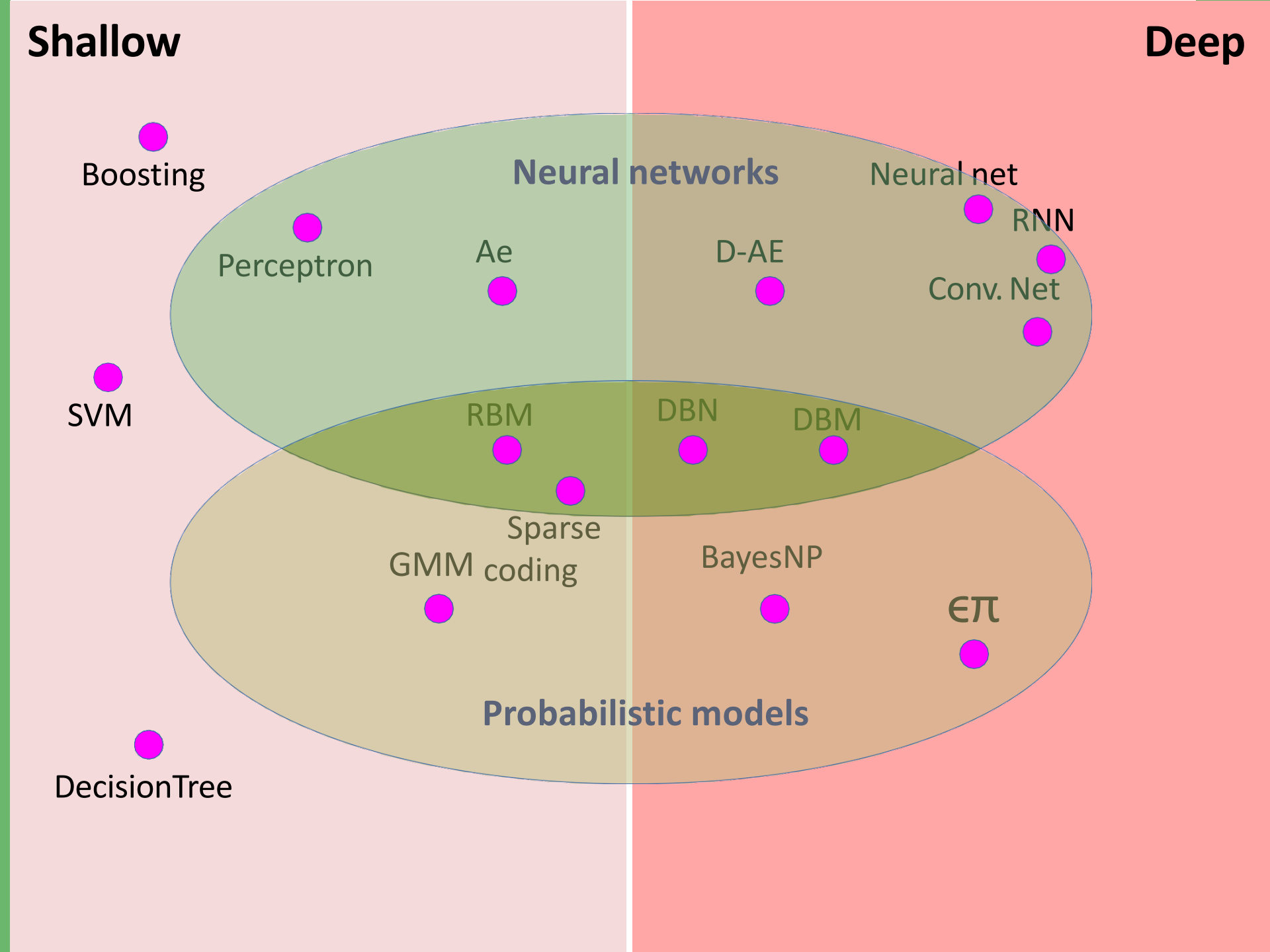
  - kernel machines (and 2-layer neural nets) are "universal".
- Deep learning machines

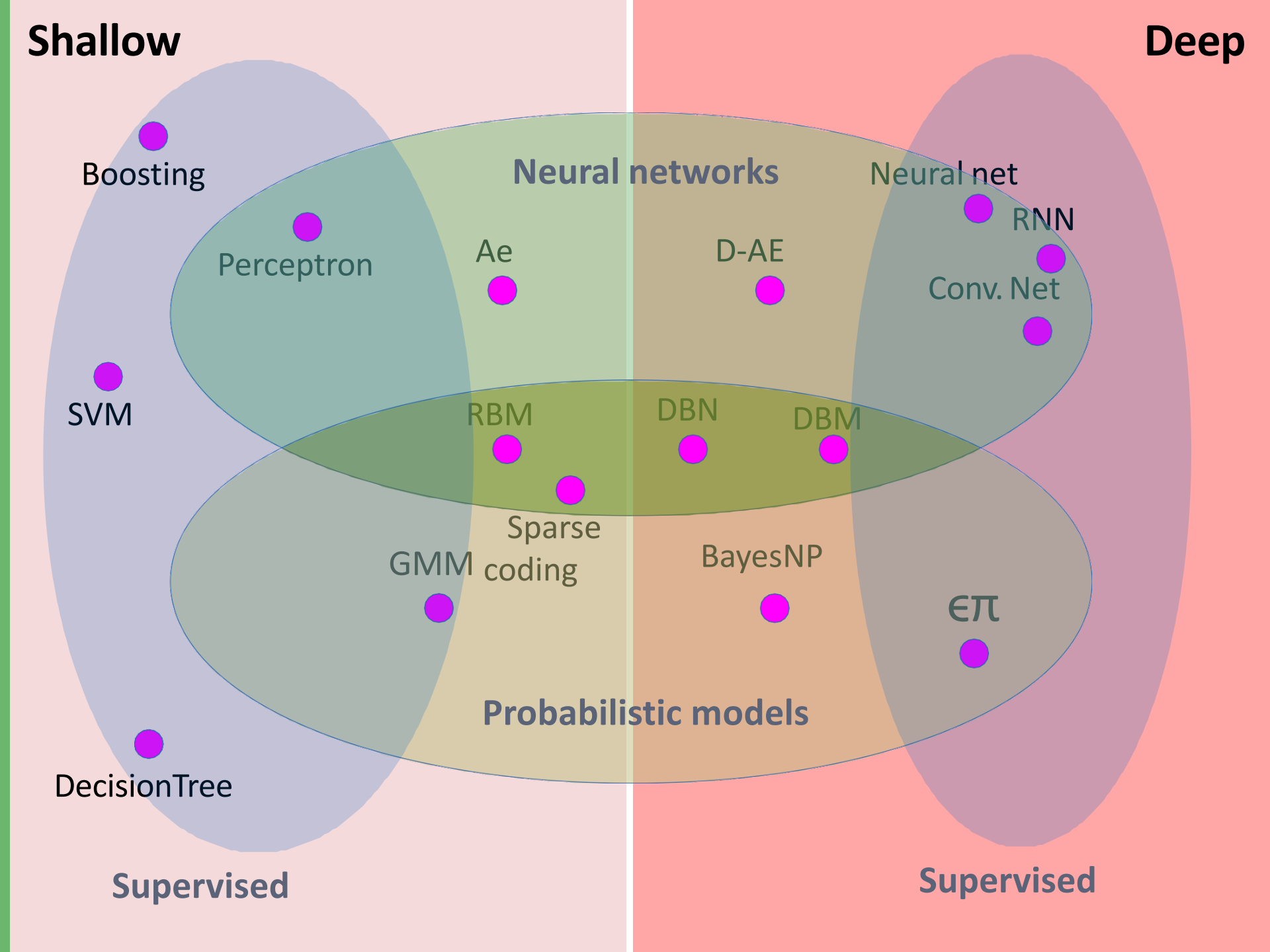
$$y = F(W^K.F(W^{K-1}.F(....F(W^0.X)...)))$$

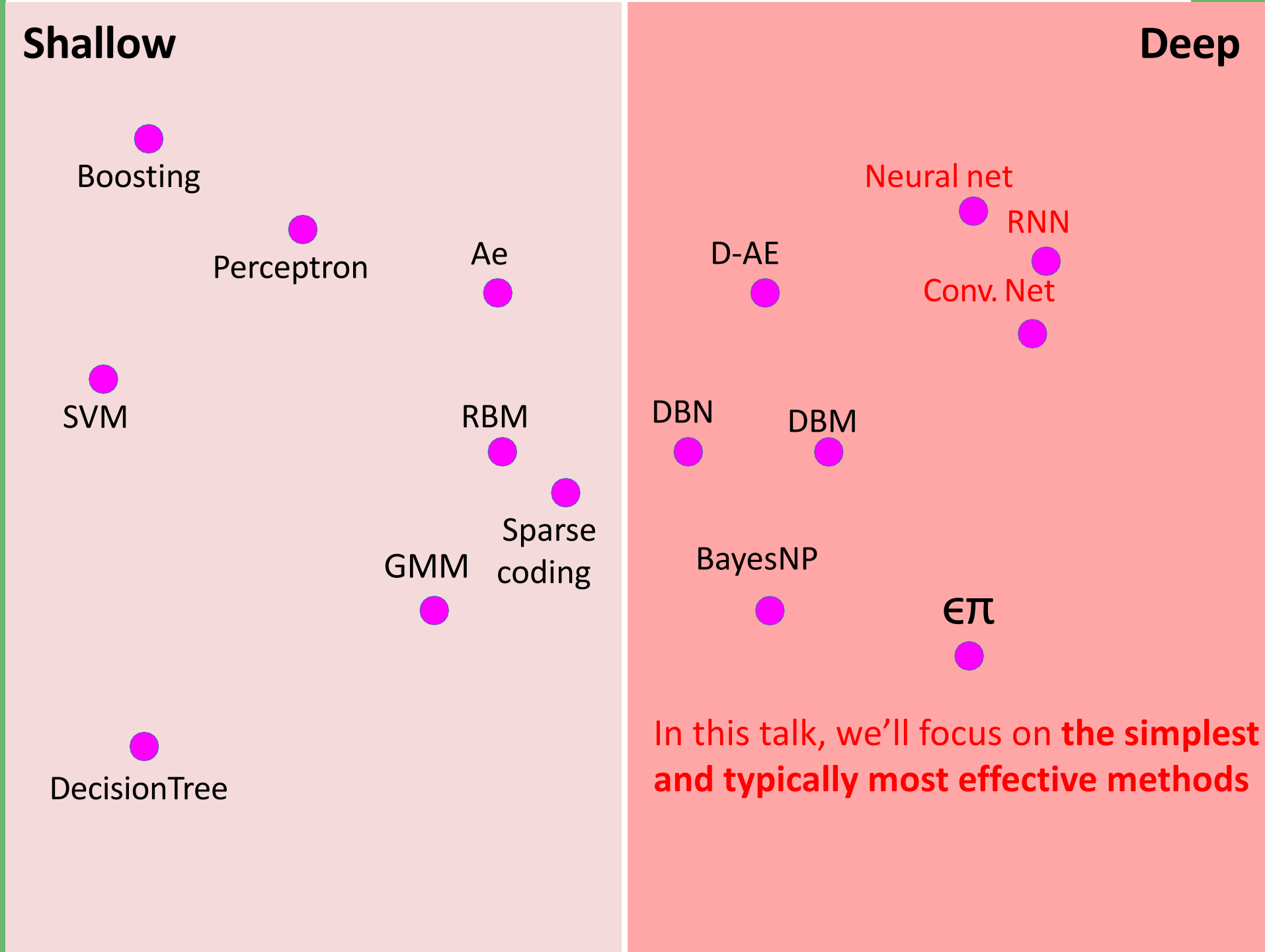
- Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition
  - They can represent more complex functions with less "hardware"
- We need an efficient parameterization of the class of functions that are useful for "AI" tasks (vision, audition, NLP...)

[Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks"]

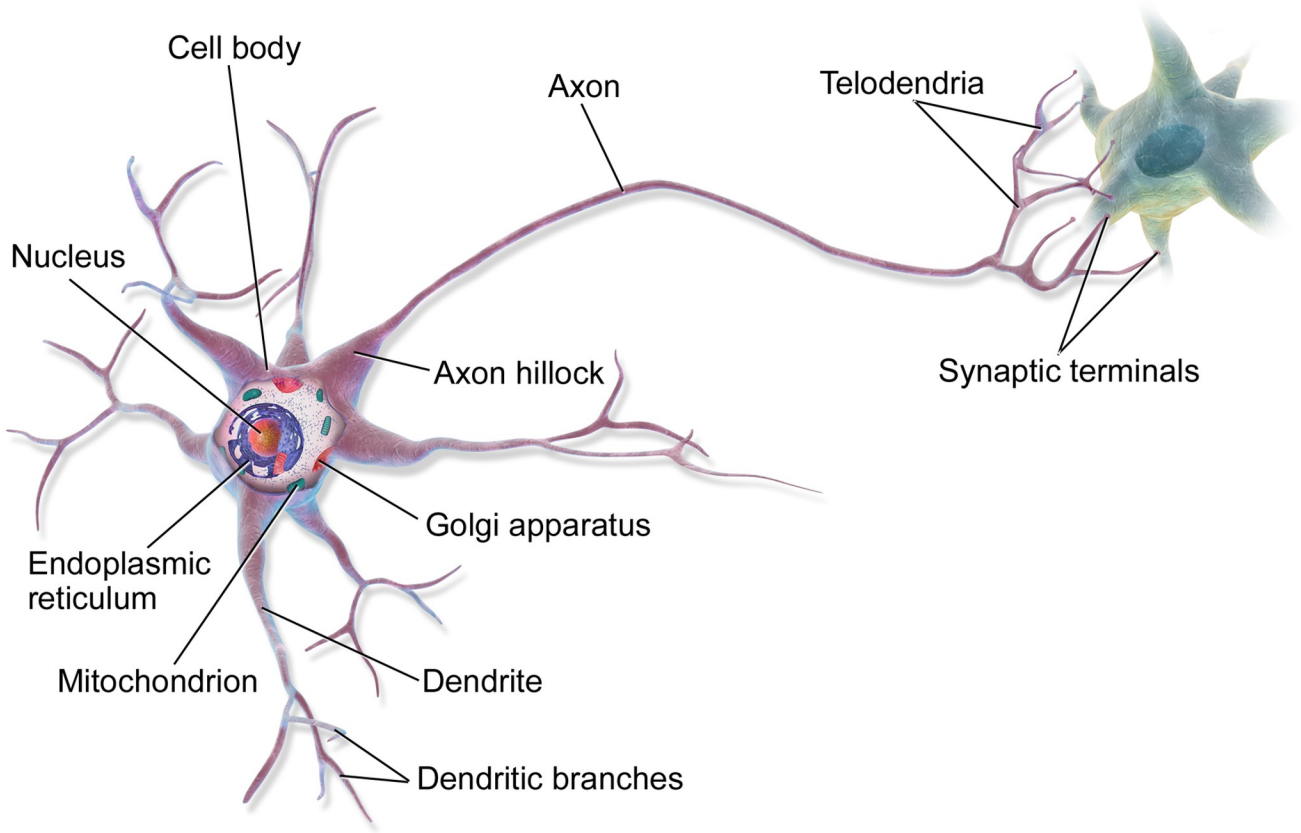

Shallow
Deep
Boosting
Perceptron
Ae
SVM
RBM
Sparse coding
GMM
DecisionTree
Neural net
RNN
D-AE
Conv. Net
DBN
DBM
BayesNP
ϵπ


Shallow
Deep
Boosting
Neural networks
Neural net
RNN
Perceptron
Ae
D-AE
Conv. Net
SVM
RBM
DBN
DBM
Sparse
GMM coding
BayesNP
∈π
Probabilistic models
DecisionTree


Shallow
Deep
Boosting
Neural networks
Neural net
RNN
Perceptron
Ae
D-AE
Conv. Net
SVM
RBM
DBN
DBM
Sparse coding
GMM
BayesNP
Probabilistic models
DecisionTree
Supervised
Supervised

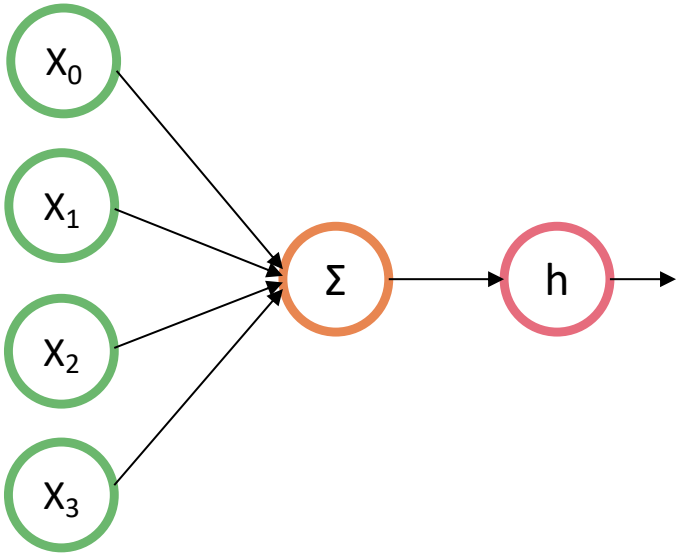

Shallow
Deep
Boosting
Perceptron
Ae
SVM
RBM
Sparse coding
GMM
DecisionTree
Neural net
RNN
D-AE
Conv. Net
DBN
DBM
BayesNP
In this talk, we'll focus on the simplest and typically most effective methods

Feed-forward neural network



Neuron model as logistic unit

We can model a neuron as a logistic unit, able to linearly separate your input data



$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \text{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Input vector weights

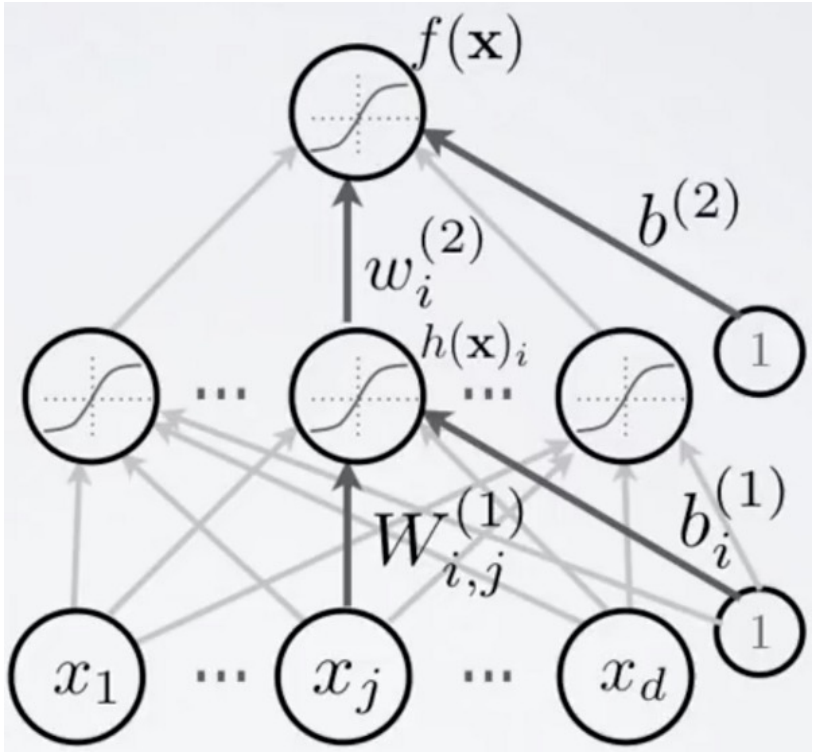
We define a cost function that tell us how well our model is fitting our input data. This cost function J is usually called Loss function.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \qquad \underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$$

with θ consisting of w and b

Neuron model as logistic unit

If we start to pack the previous simple model we can create a hierarchical model, Made up of different neurons on different levels.

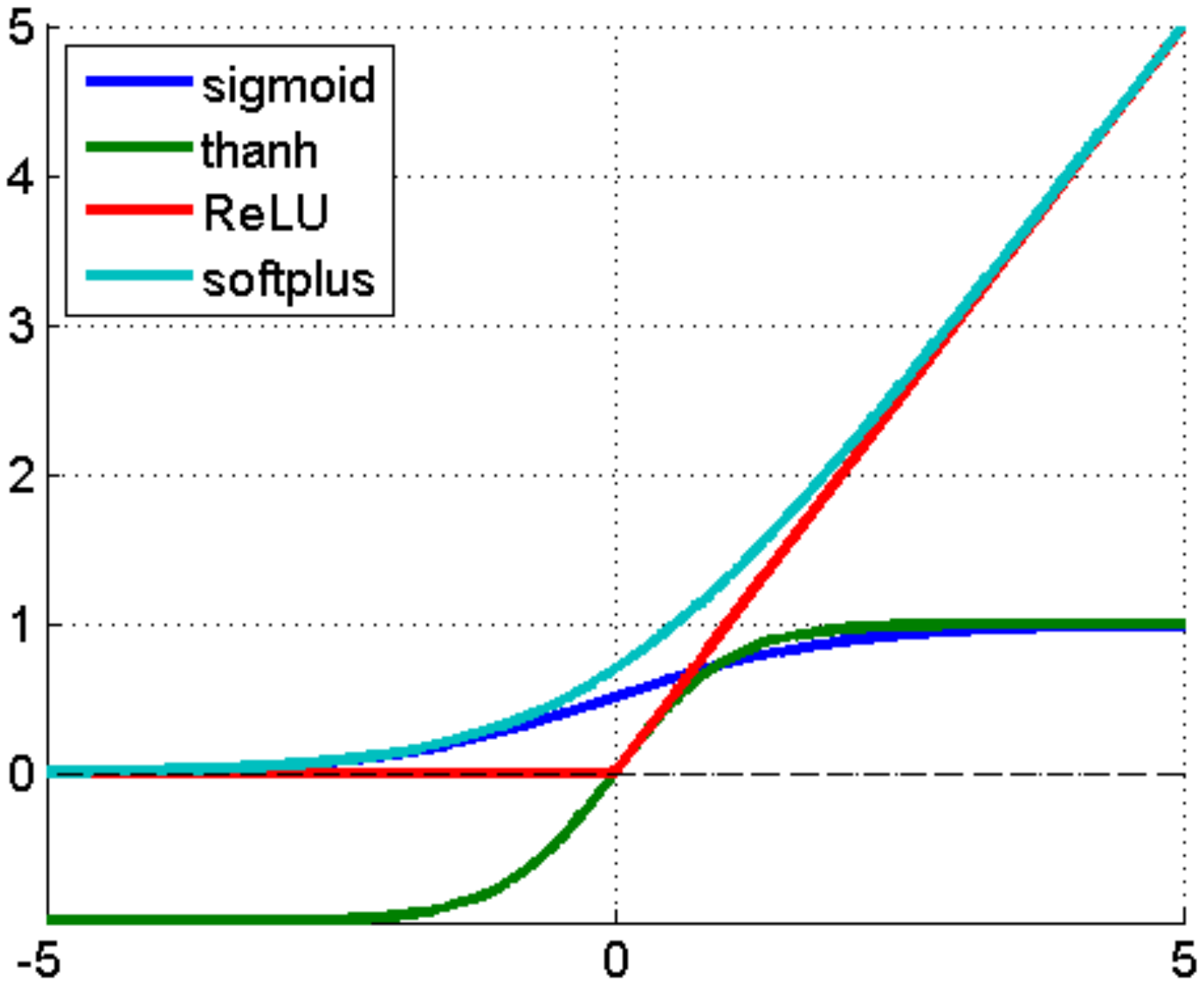


The bottom layer is called **input layer**.
The topmost or ***output*** layer contains the *output neurons*, or, as in this case, a single output neuron.
The middle layer is called a ***hidden layer***, since the neurons in this layer are neither inputs nor outputs.

Now the cost function will be in the form:

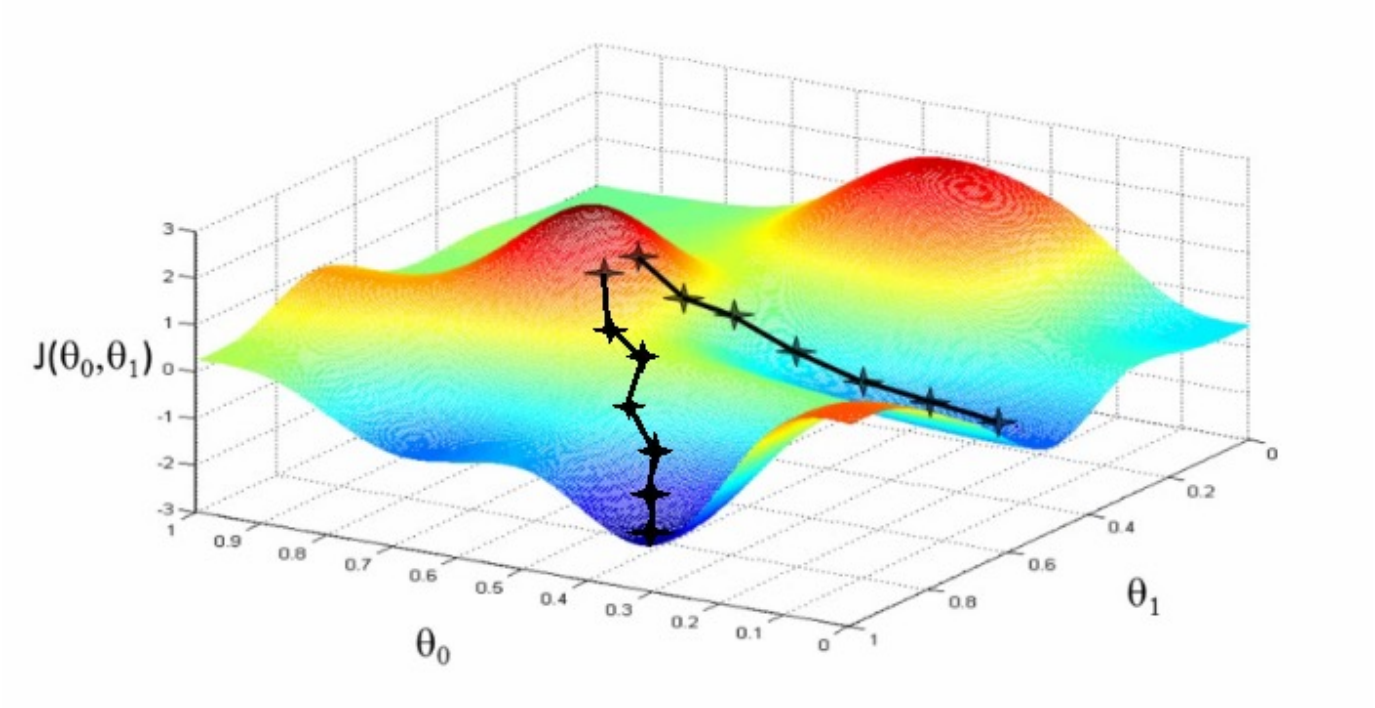
$$J(w,x) \cong \frac{1}{2n}\sum_x \left|\left|f(x) - y\right|\right|^2$$

Nonlinearities



Optimization

Defnition: an optimization problem consists of **minimizing** (or maximizing) a real **function** by systematically choosing **input values** from within an allowed set and computing the value of the function.



[A Ferrari]

Optimization

Is machine learning just optimization?

Optimization is just about finding **good minima**.

This is just half part of the story.

Classifcation is about attaining **low test classifcation** error.

Necessary to learn a function that **generalize** well to new unseen samples.

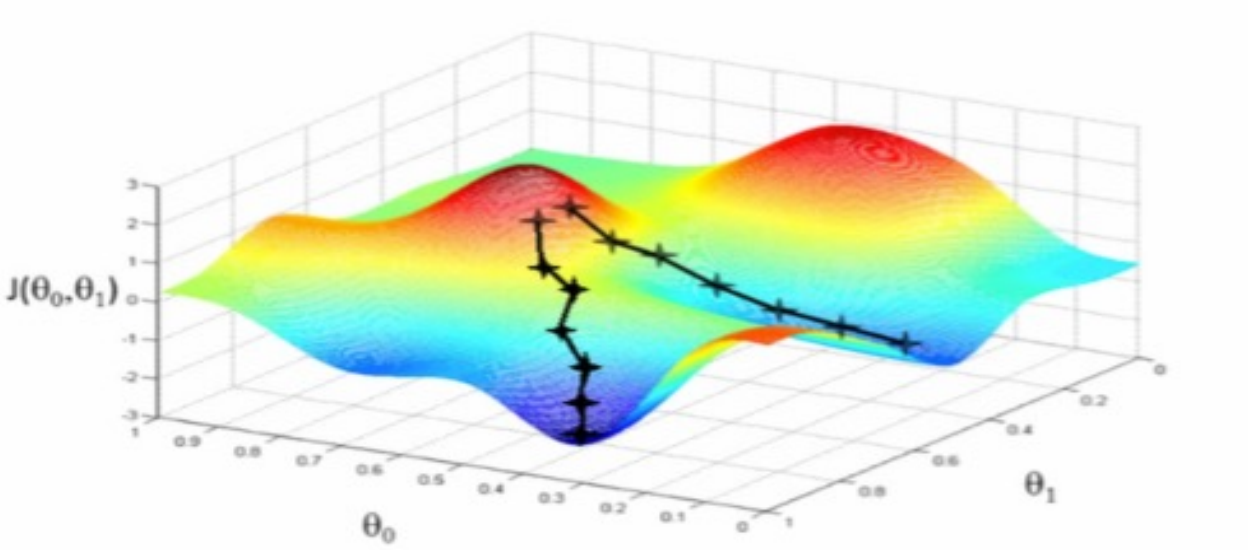
Optimization is about:

$$W' = argmin_{W,x} J(W, x)$$

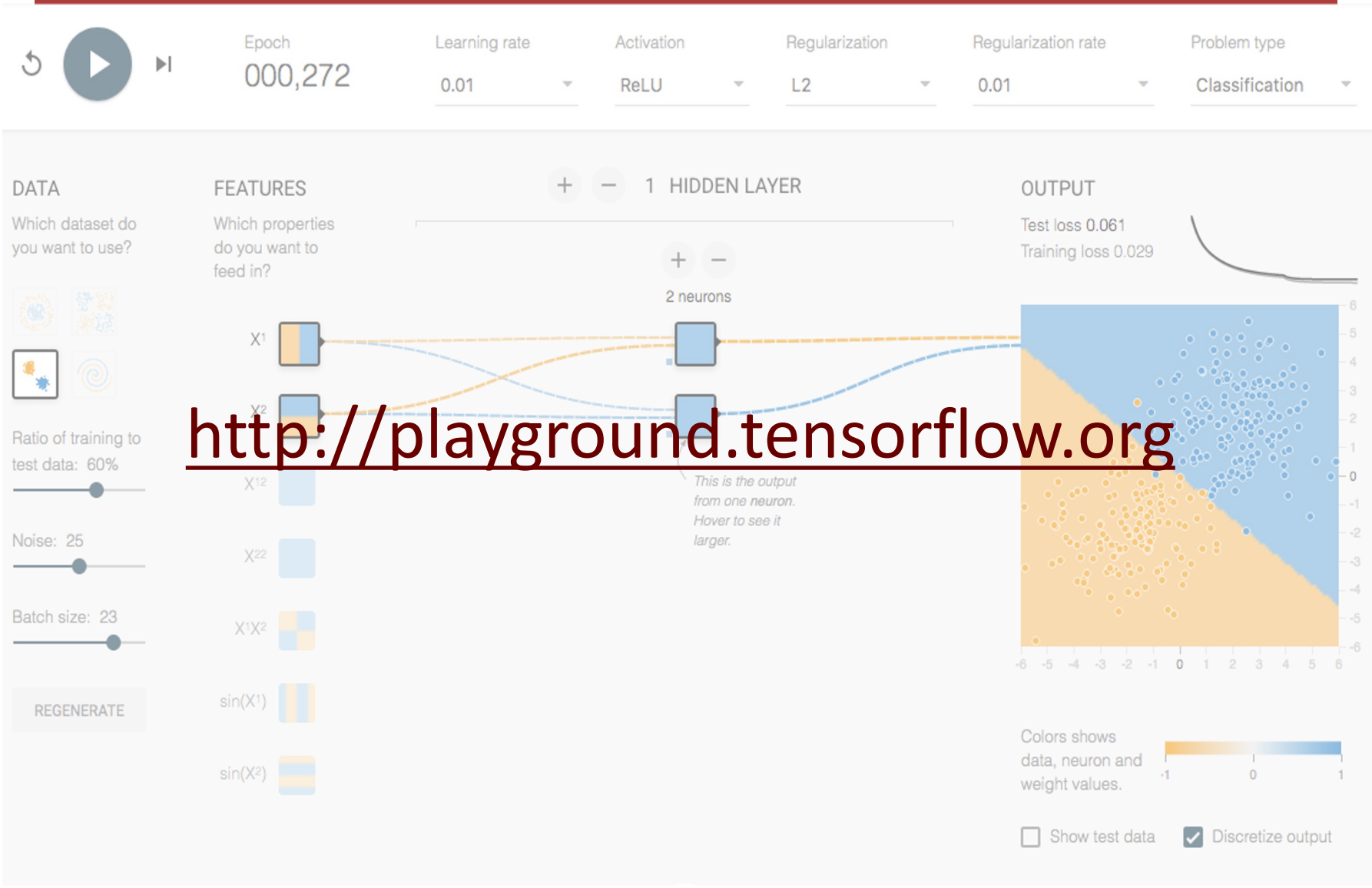
The basics: **Gradient Descent**

W is iteratively adjusted according to:

$$W_k = W_{k-1} - \varepsilon \cdot \frac{\partial E(W)}{\partial W}$$



Demo time



http://playground.tensorflow.org

The Nobel prize winning physicist Enrico Fermi was once asked his opinion of a mathematical model some colleagues had proposed as the solution to an important unsolved physics problem. The model gave excellent agreement with experiment, but Fermi was skeptical. He asked how many free parameters could be set in the model. "Four" was the answer. Fermi replied: "I remember my friend Johnny von Neumann used to say, with four parameters I can fit an elephant, and with five I can make him wiggle his trunk."
A four-parameter elephant may be found here.

OVERFITTING - UNDERFITTING

The basics: **Objective of learning**

$$E_{test} - E_{train} = k \cdot (h/P)^{\alpha}$$

k = constant

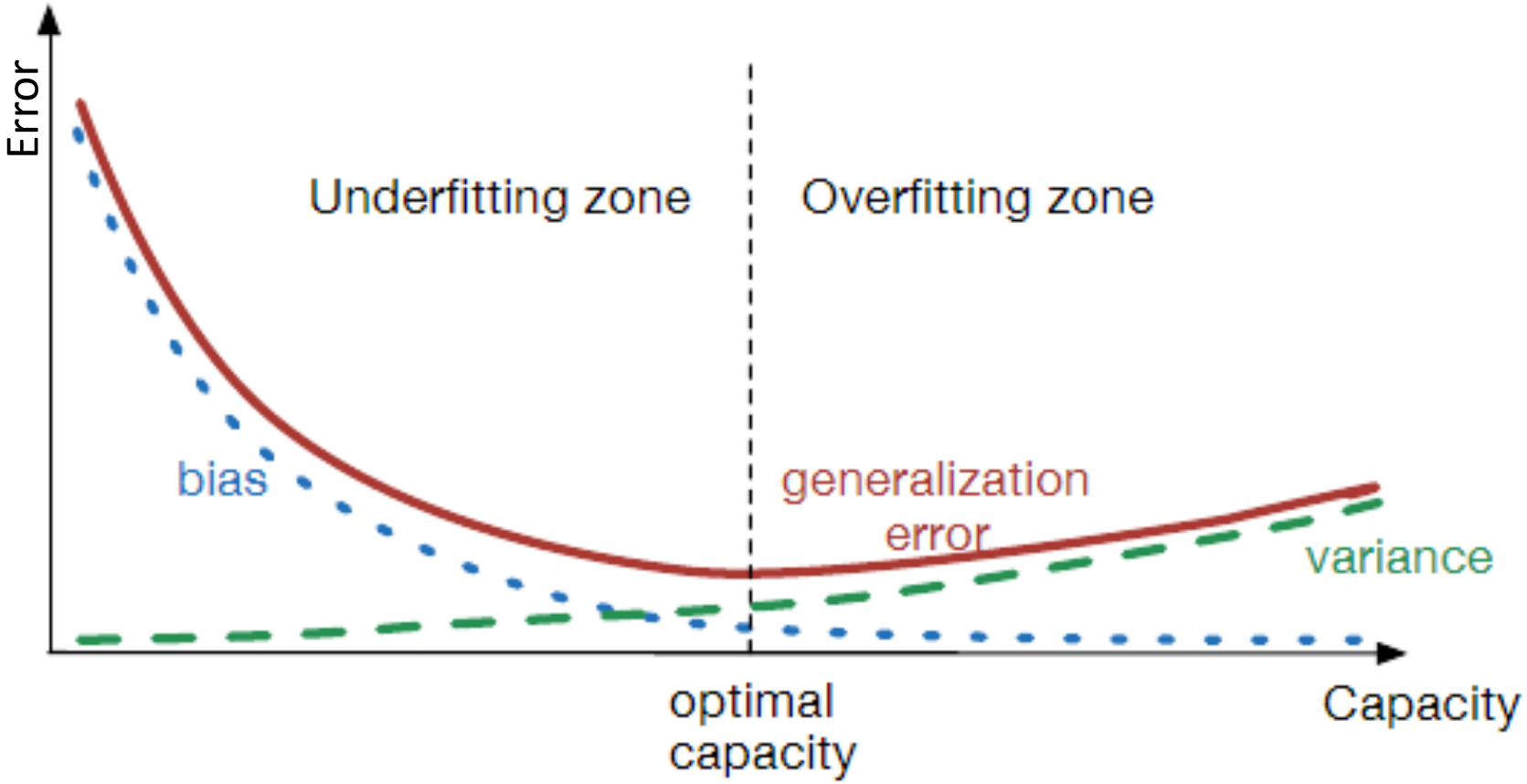
h = capacity

P = number of training samples

α = number between 0.5 and 1.0

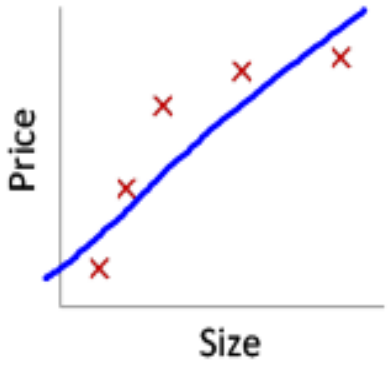
"Statistical mechanics of learning from examples" Seung *et.al.*, 1992
"Measuring the vc-dimension of a learning machine" Vapnik *et.al., 1994*
"Learning curves: asymptotic values and rate of convergence" Cortes *et.al.*, 1994

Overfitting and underfitting



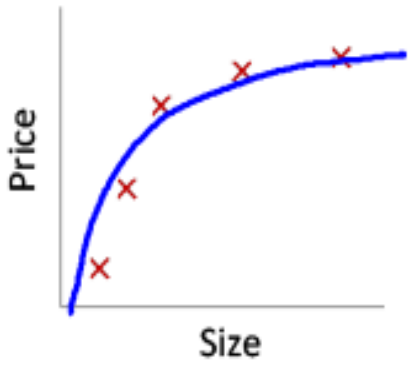
Overfitting and underfitting

Bias/variance



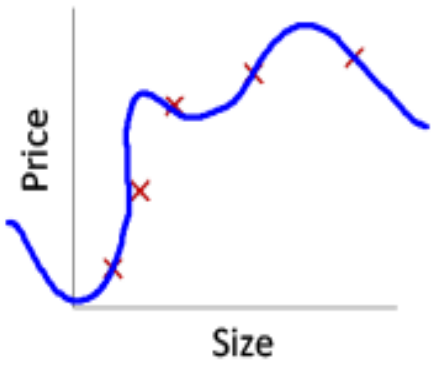
$\theta_0 + \theta_1 x$

High bias
(underfit)



$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)

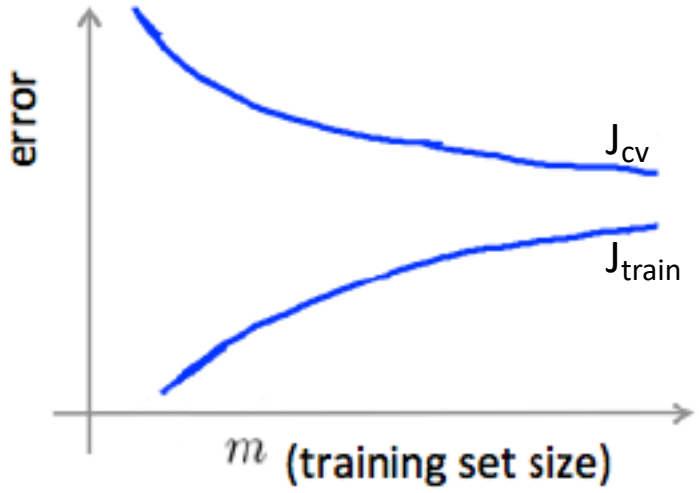
Overfitting: If we have too many features, the learned hypothesis may fit the training set very well, but fail to generalize to new examples (predict prices on new examples)

Learning curvers

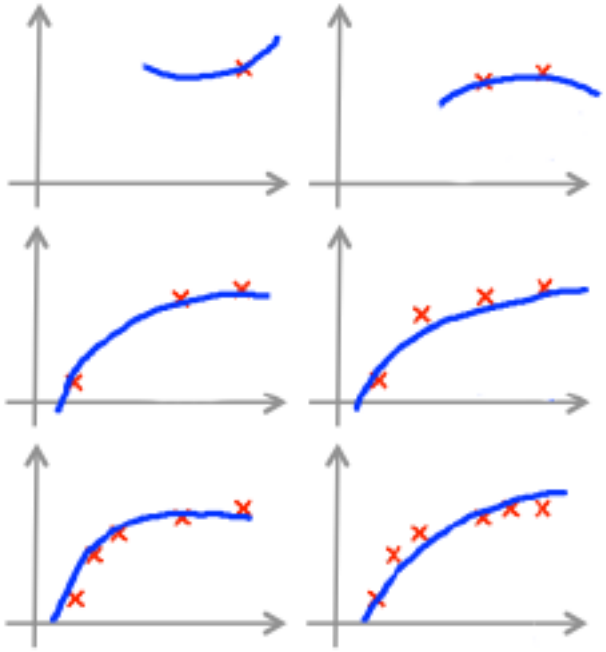
Learning curves

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



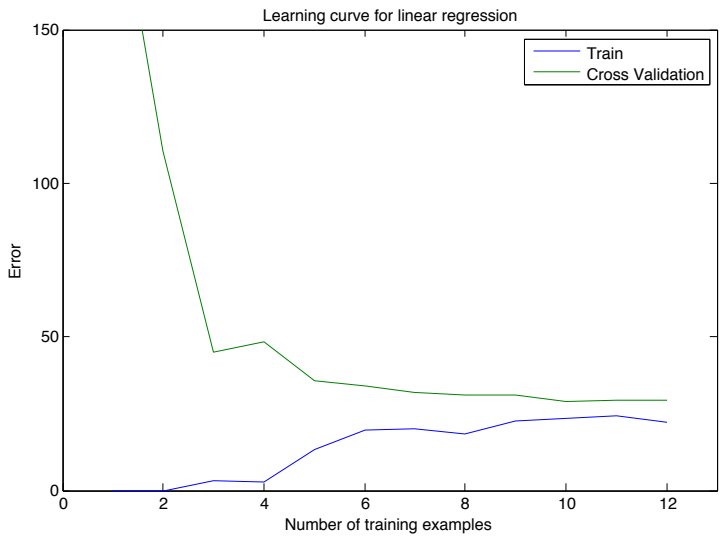
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

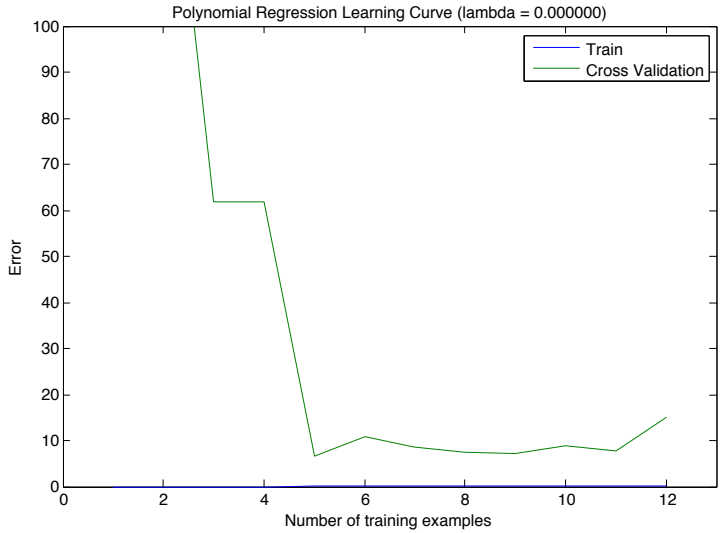


Learning curvers

High bias: If a learning algorithm is suffering from high bias, getting more training Data will not (by itself) help much.

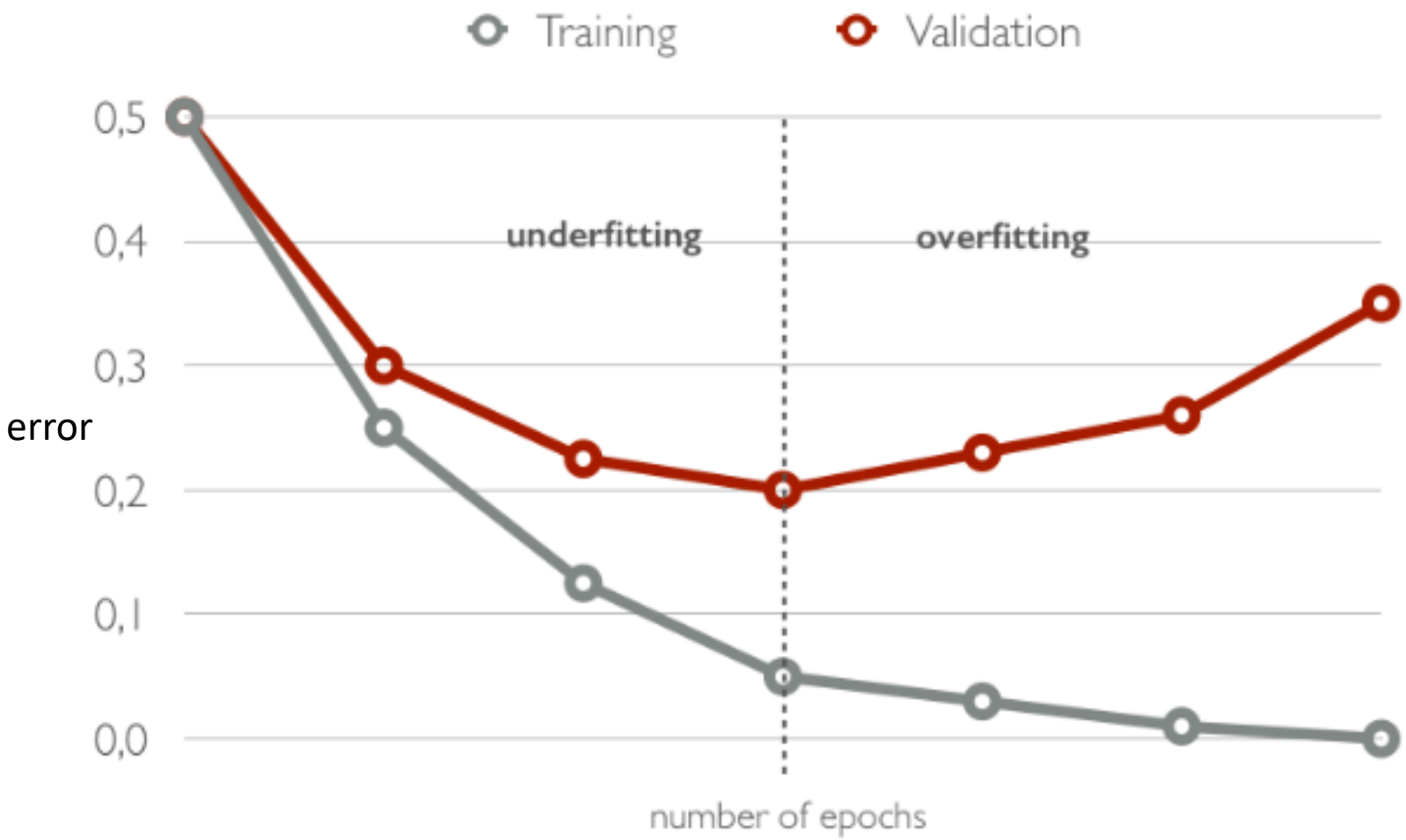
High variance: If a learning algorithm is suffering from high variace, getting more training data is likely to help.





REGULARIZATION

Early stopping



- Always split your dataset into train/validation/train set
- If you have enough resources, do a cross-validation

Regularization: weight decay

The idea of L2 regularization is to add an extra term to the cost function, a term called the *regularization term*. Here's the regularized cross-entropy:

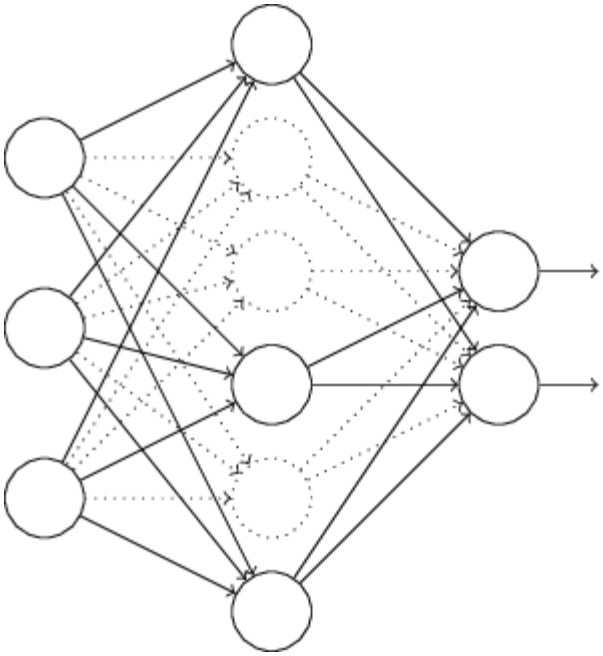
$$J(W,x) = J_0(W,x) + \frac{\lambda}{2n}\sum_w w^2$$

Where J_0 is the original, unregularized cost function.

Why does regularization help reduce overfitting?

With that said, this idea of preferring simpler explanation should make you nervous. People sometimes refer to this idea as "Occam's Razor", and will zealously apply it as though it has the status of some general scientific principle. But, of course, it's not a general scientific principle. There is no *a priori* logical reason to prefer simple explanations over more complex explanations. Indeed, sometimes the more complex explanation turns out to be correct.

Regularization: dropout



To a first approximation, dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks. Bagging involves training multiple models and evaluating multiple models on each test example. This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory. It is common to use ensembles of five to ten neural networks—Szegedy et al. (2014a) used six to win the ILSVRC— but more than this rapidly becomes unwieldy.

Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks

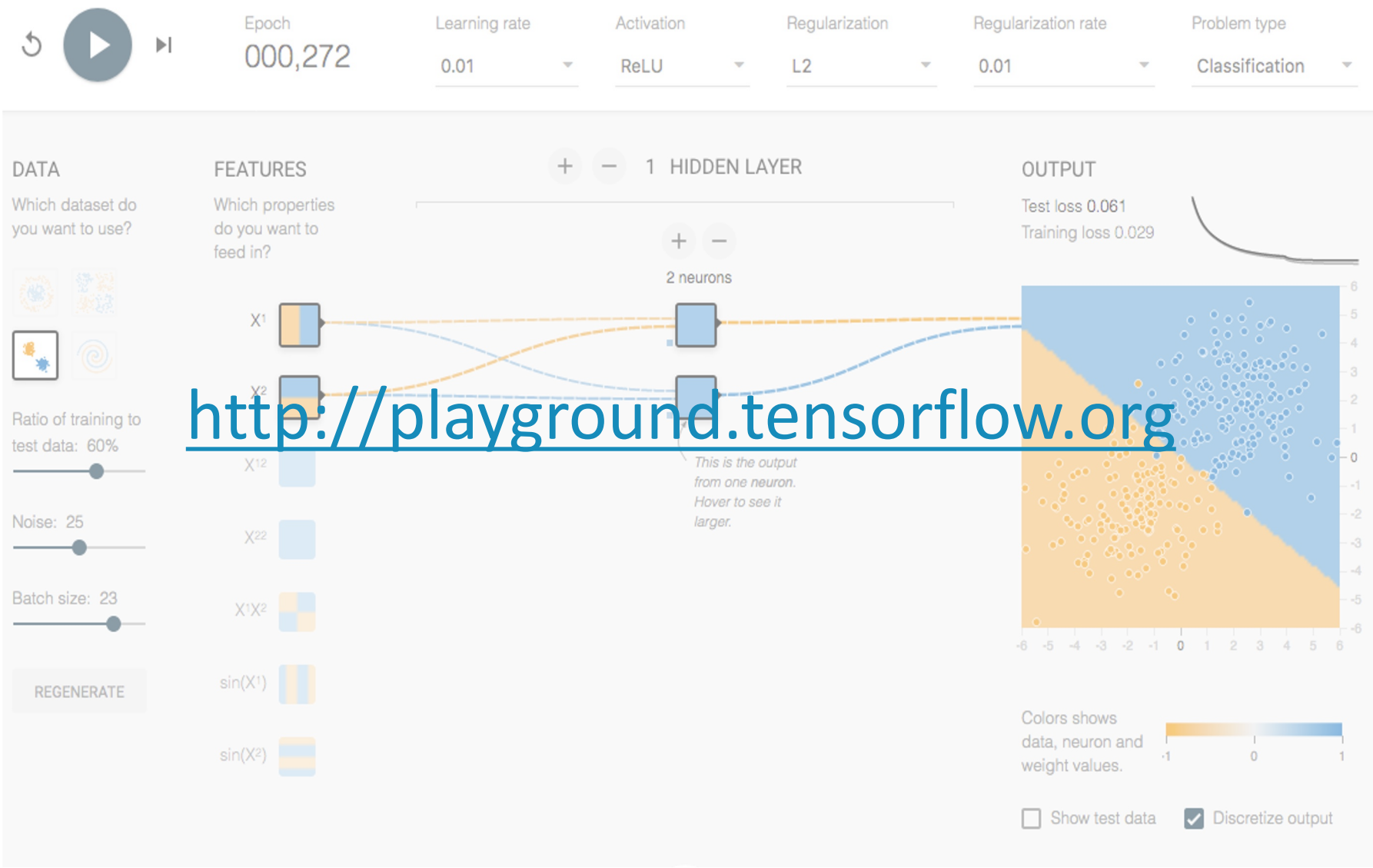
Hints

Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features $(x_1^2, x_2^2, x_1 x_2, \text{etc})$
- Try decreasing λ
- Try increasing λ

[Andrew NG]

Demo time


Epoch
000,272
Learning rate
0.01
Activation
ReLU
Regularization
L2
Regularization rate
0.01
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 60%
Noise: 25
Batch size: 23
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
2 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.061
Training loss 0.029
Colors shows data, neuron and weight values.
Show test data
Discretize output
http://playground.tensorflow.org

LAB SESSION

Neural networks and Regularization