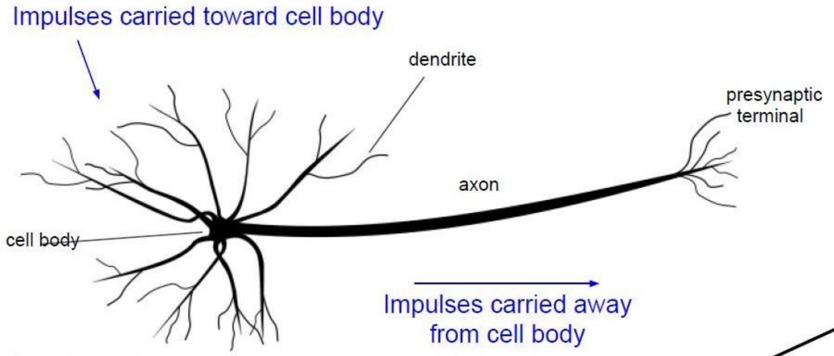
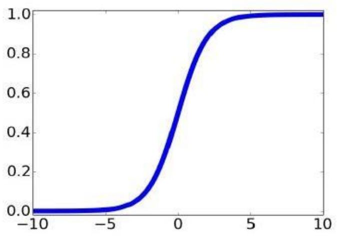
Lecture 5: Fundamentals of Deep Learning
(DNN, CNN)

Deep Neural Networks

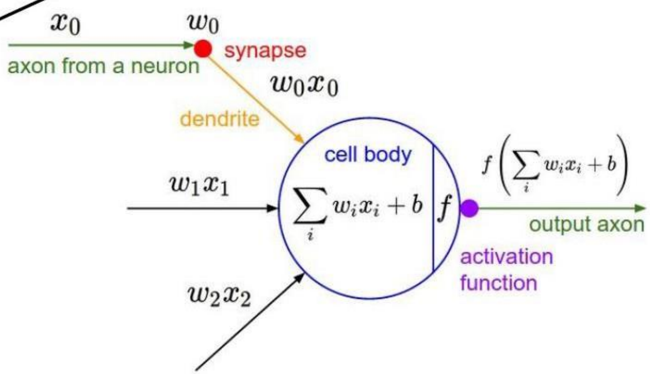
Biological Neuron & Perceptron





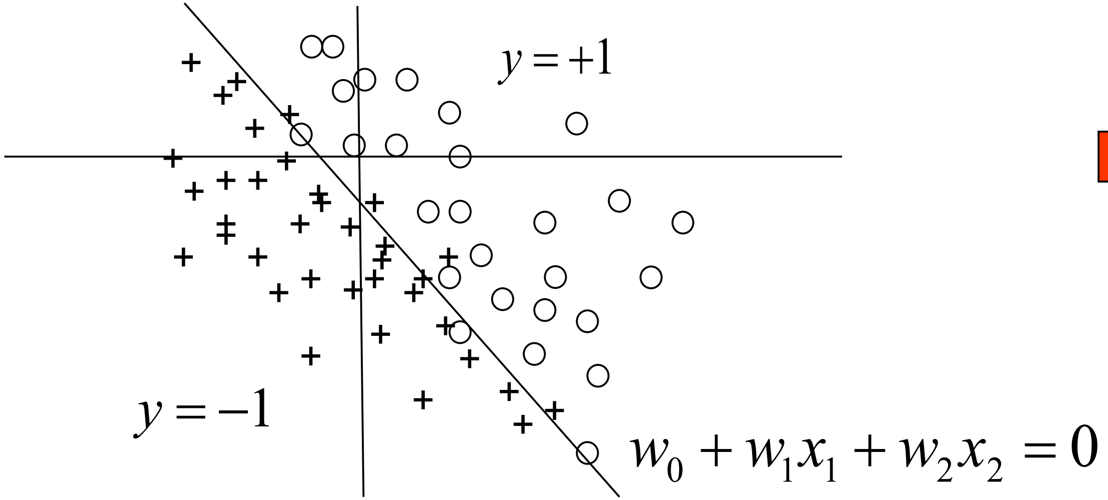
sigmoid activation function

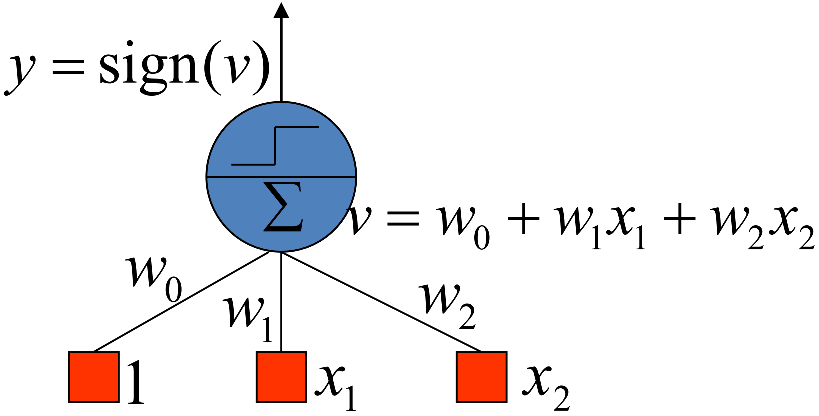
$$\frac{1}{1+e^{-x}}$$



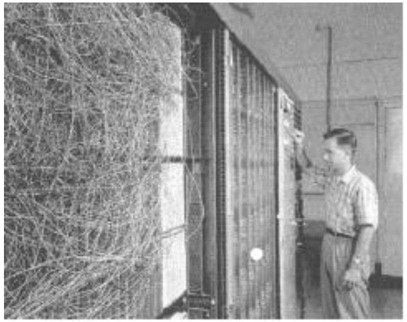
Perceptrons (1958, Rosenblatt)

Linear separation
Inputs: vector of real values
Outputs :1 or -1





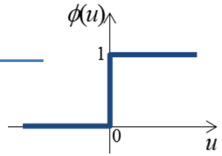
Perceptrons (1958, Rosenblatt)



A neuron (activation function)

MaCulloch-Pittz Neuron (Step function)

$$y_j = \phi_{step}\Big(\underbrace{\sum_{i=1}^{n} w_{ij}\, x_i}_{u} + \underbrace{w_{0j}}_{\text{bias}}\Big)$$

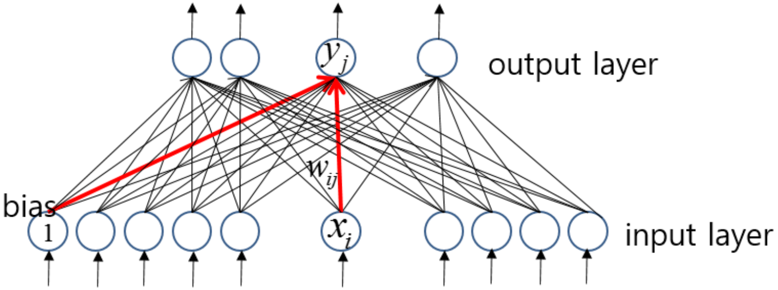


Structure

Single-layer, Feed forward,
Fully connected

Learning

Supervised learning with
binary input/output



$$w_{ij}^{(\tau+1)} = w_{ij}^{(\tau)} + \underbrace{\eta}_{\text{Learning rate}}(\underbrace{t_j}_{\text{Target}} - \underbrace{y_j}_{\text{Output}})x_i$$

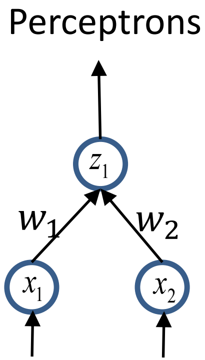
Limit of Perceptrons

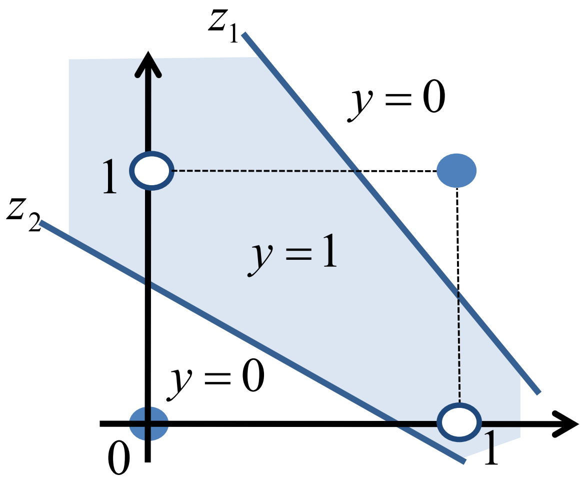
Shape of output function is linear

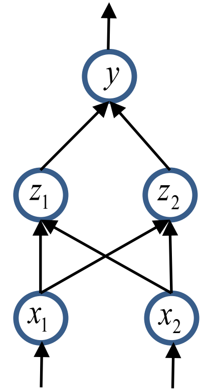
→ Nonlinear decision boundary cannot be represented

XOR function

A typical problem with nonlinear decision boundary







Multilayer Perceptrons (MLP)

A <u>neuron</u> (activation function)

Weighted sum of input signal

output is nonlinear mapping of input:

(logistic) sigmoid, hyper tangent

<u>Structure</u>

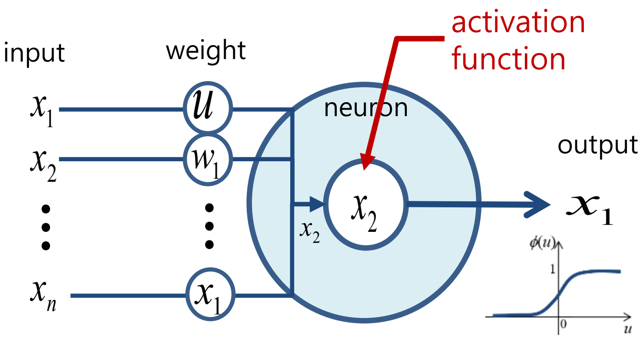
Multi-layer, Feed forward,

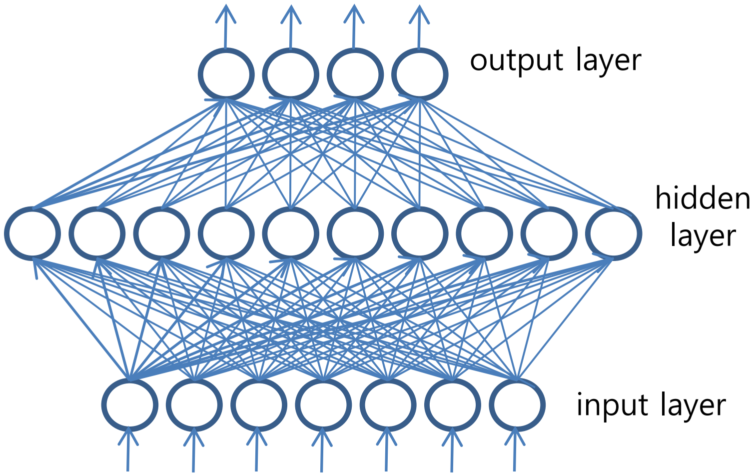
Fully connected

<u>Learning</u>

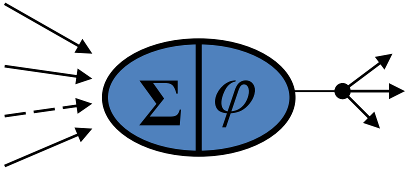
Supervised learning with

error backpropagation algorithm

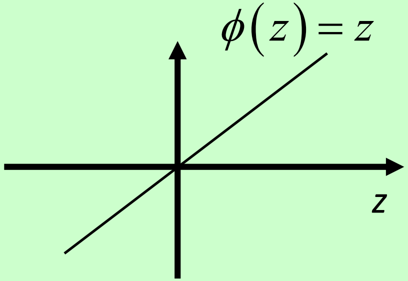




Activation Functions

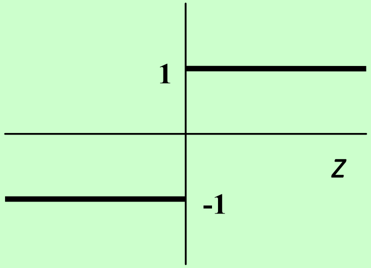


Linear activation



Threshold activation

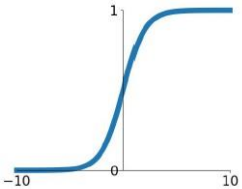
$$\phi(z) = \mathrm{sign}(z) = \begin{cases} 1, & if \quad z \geq 0, \\ -1, & if \quad z < 0. \end{cases}$$



Activation functions

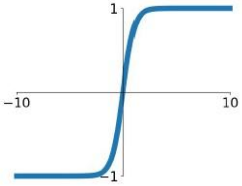
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



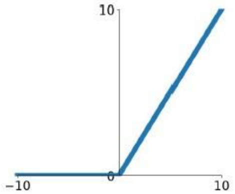
tanh

$\tanh(x)$



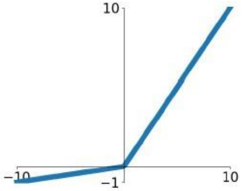
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

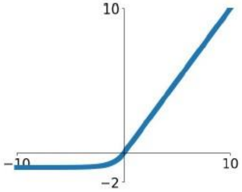


Maxout

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

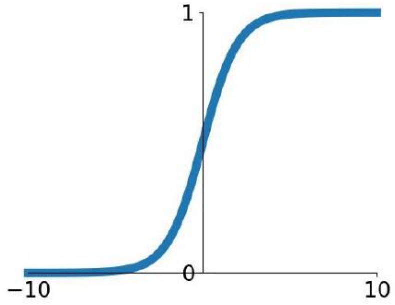
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation functions

Activation Functions



Sigmoid

$$\sigma(x) = 1/(1+e^{-x})$$

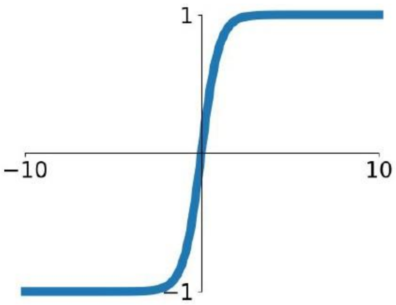
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

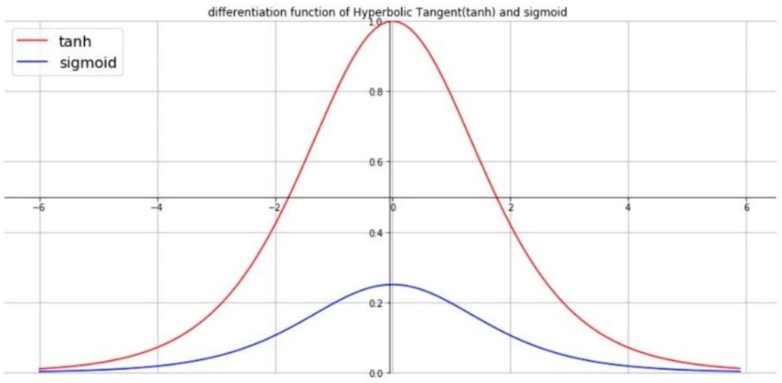
Activation functions

- **Tanh**



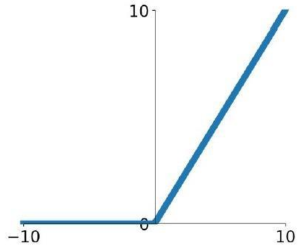
tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

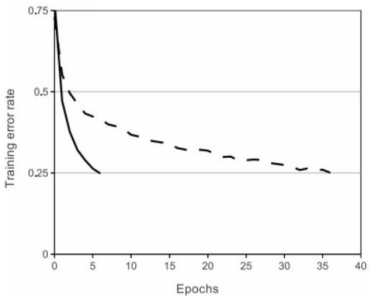


Activation functions

- **Rectified Linear Unit**



ReLU
(Rectified Linear Unit)



- Computes **f(x) = max(0,x)**

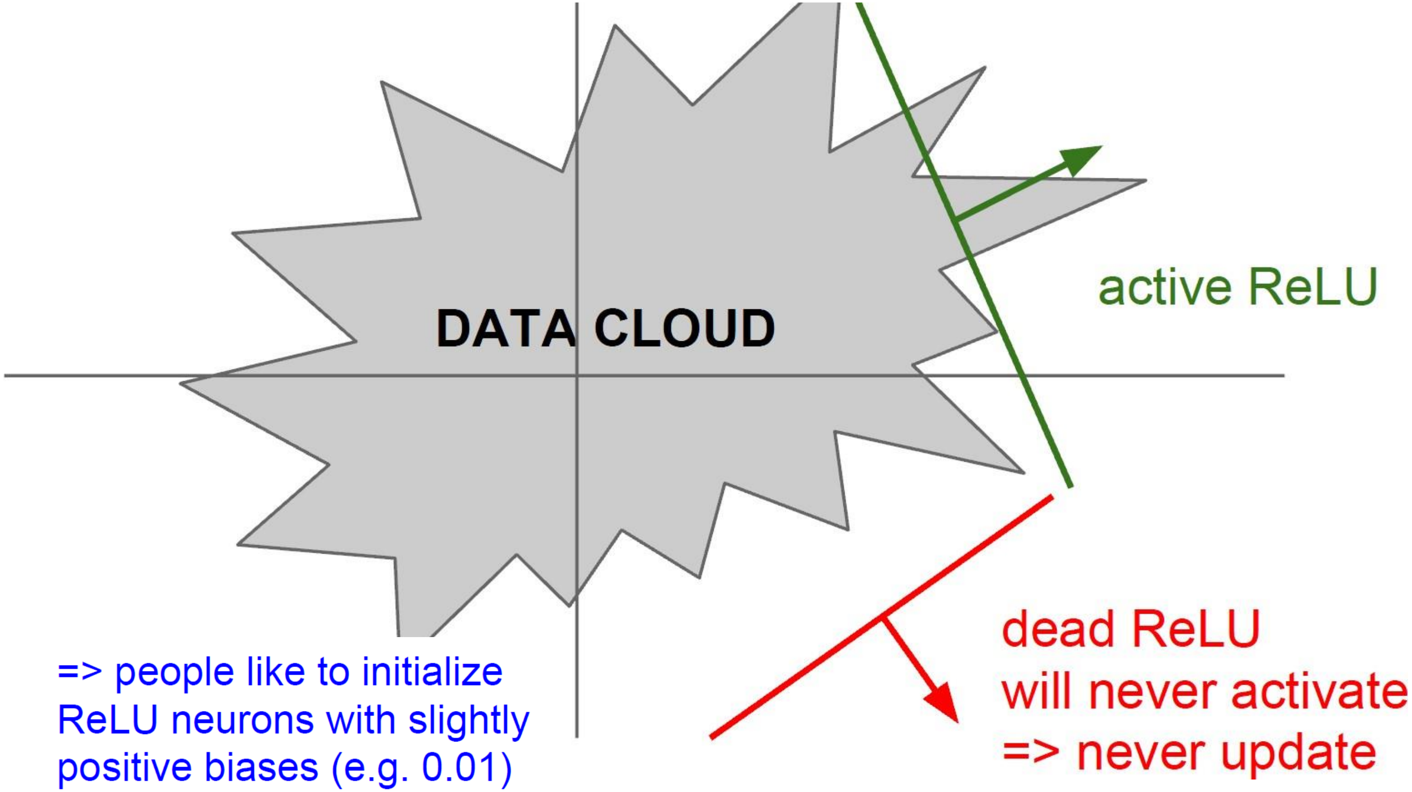
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

- Not zero-centered output
- An annoyance:

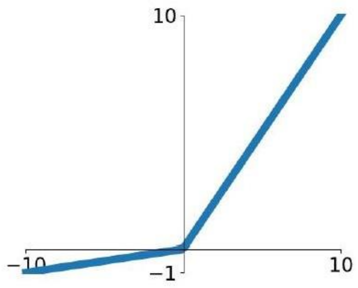
hint: what is the gradient when x < 0?

Activation functions

- **Rectified Linear Unit**



Activation functions



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

some small value

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not "die".**

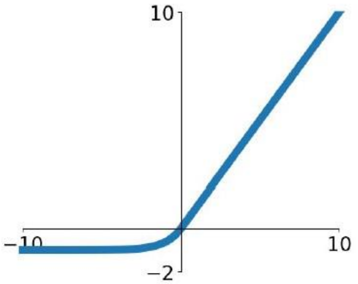
Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

backprop into α
(parameter)

Activation functions

Exponential Linear Units (ELU)



- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$

- Computation requires exp()

Activation functions

- **MaxOut**

  - Does not have the basic form of dot product -> nonlinearity
  - Generalizes ReLU and Leaky ReLU
  - Linear Regime! Does not saturate! Does not die!

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

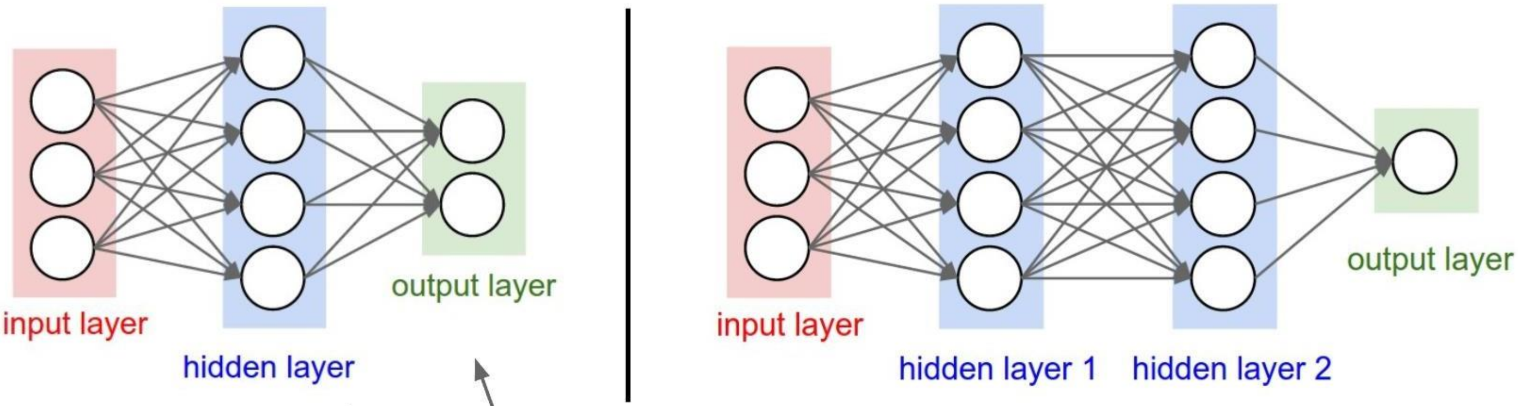
Problem: doubles the number of parameters :(

Activation functions

TLDR: In practice:

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout
- Try out tanh but don't expect much
- Never use sigmoid

Neural Networks: Architectures

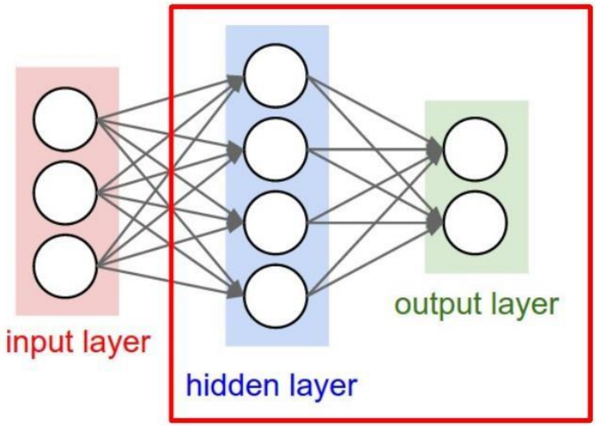


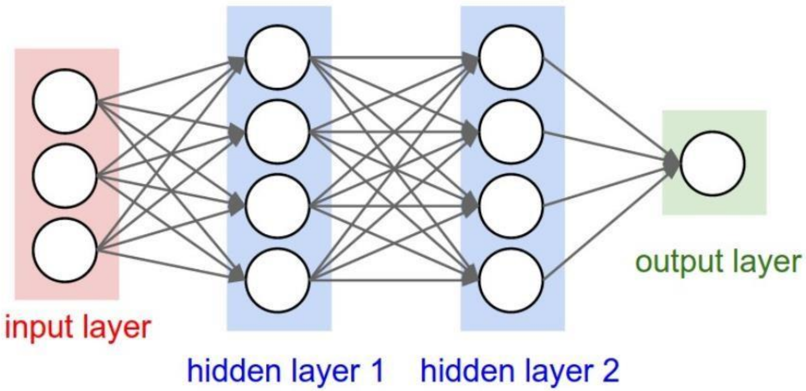
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"Fully-connected" layers

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Neural Networks: Architectures

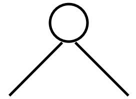
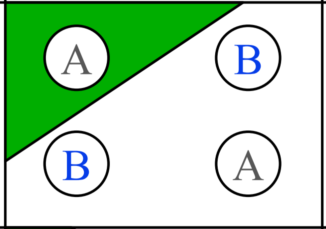
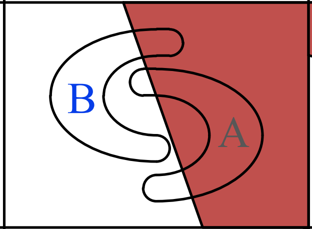
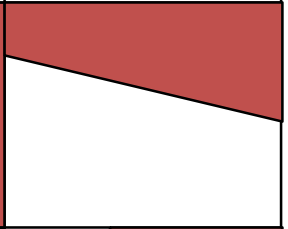
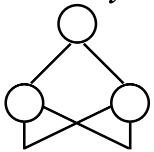
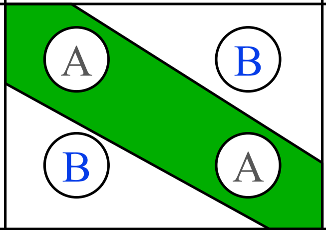
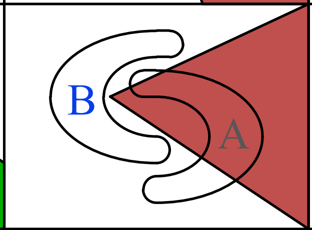
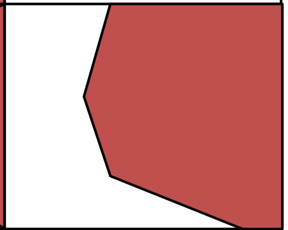
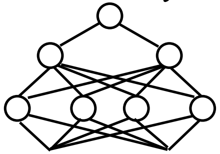
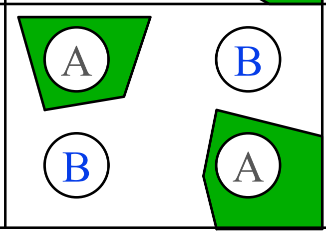
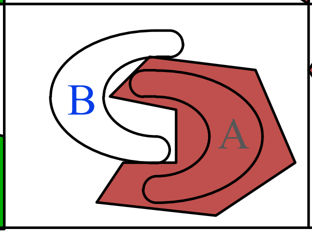
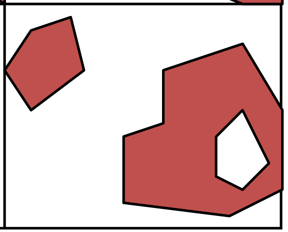




Number of Neurons: 4+2 = 6
Number of Weights: [4x3 + 2x4] = 20
Number of Parameters: 20 + 6 = 26 (biases!)

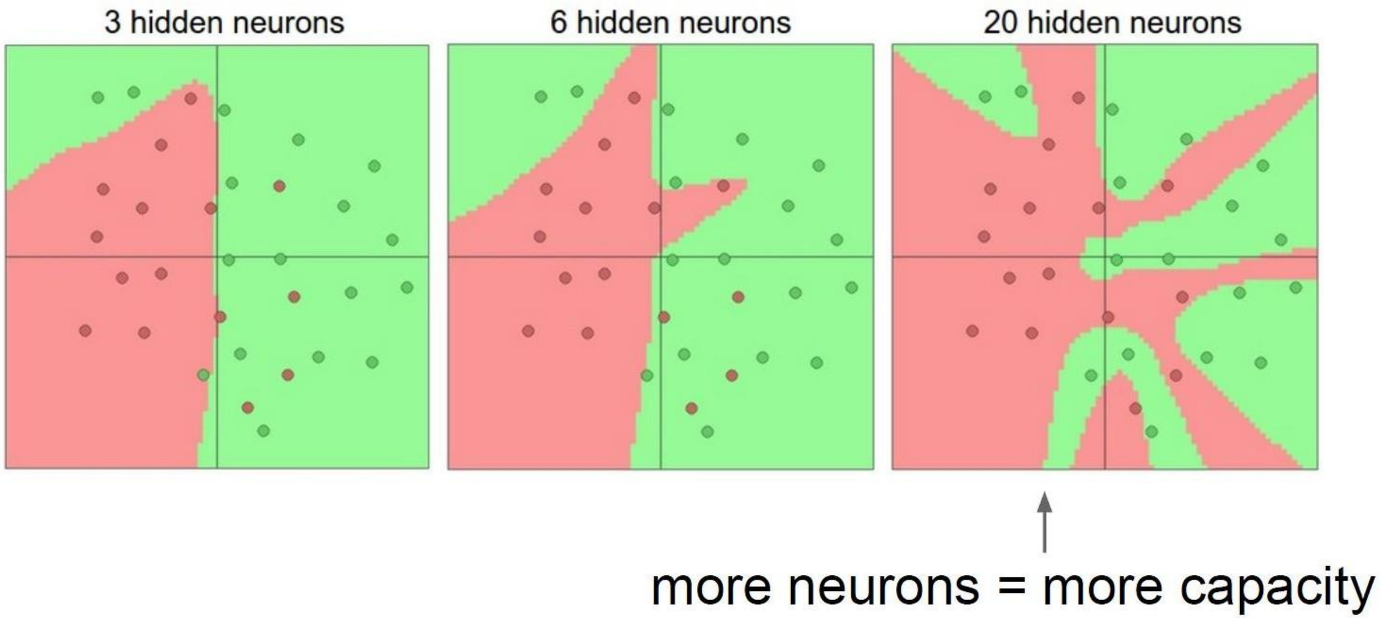
Number of Neurons: 4 + 4 + 1 = 9
Number of Weights: [4x3+4x4+1x4]=32
Number of Parameters: 32+9 = 41

Different non linearly separable problems

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyperplane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Abitrary (Complexity Limited by No. of Nodes)			

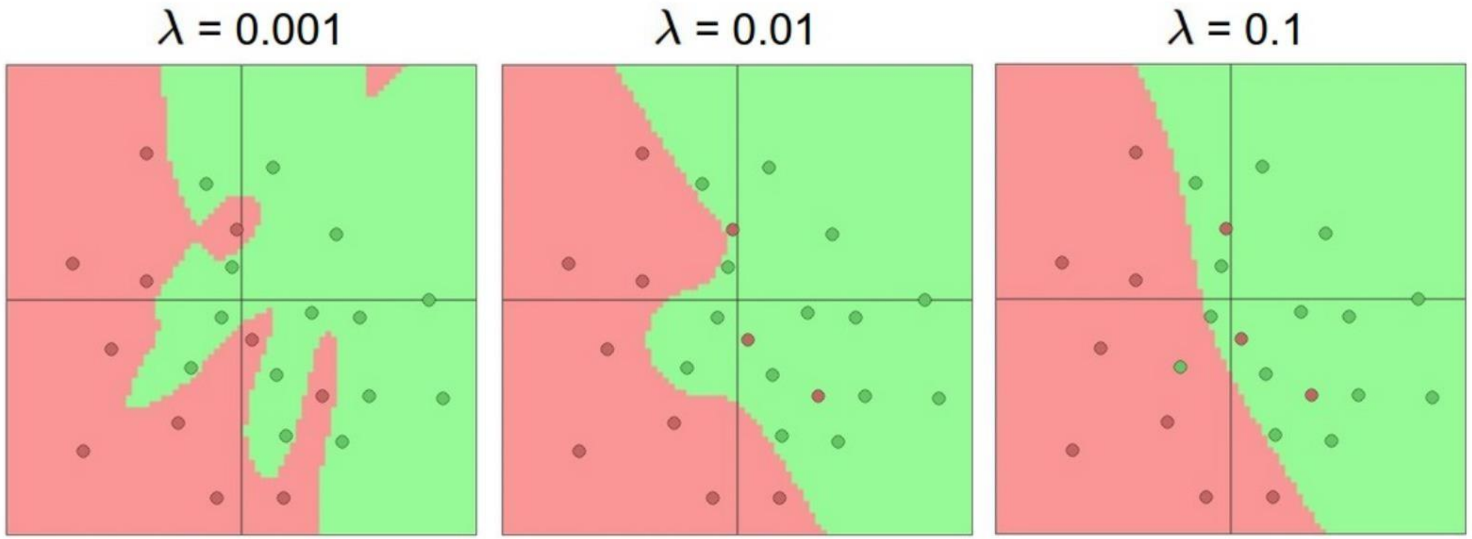
Neural Networks – An Introduction Dr. Andrew Hunter

Setting the number of layers and their sizes



Weight Decay Regularizer

Do not use size of neural network as a regularizer. Use stronger regularization instead:



(you can play with this demo over at ConvNetJS: http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

Learning of MLP

Learning of a Neural Network

To find optimal weight parameter

Learning of MLP

Supervised Learning

Uses set of training data pair $X = \{(x_i, t_i) | i = 1 \ldots N\}$

Objective

Minimize difference between output y_i computed from input x_i and target t_i

Error function, Loss function

$$E(X, \boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} e(\boldsymbol{x_i}, \boldsymbol{\theta})$$

$$= \frac{1}{2N} \sum_{i=1}^{N} \sum_{k=1}^{M} (t_k^i - y_k^i)^2 = \frac{1}{2N} \sum_{i=1}^{N} \sum_{k=1}^{M} (t_k^i - f_k(\boldsymbol{x_i}, \boldsymbol{\theta}))^2$$

Optimal weight: θ to minimize error function $E(X, \theta)$

Goal of learning

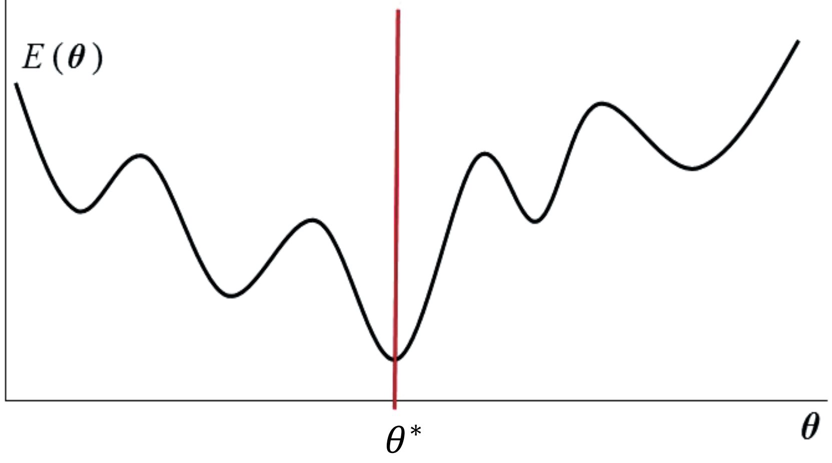
Find an optimal weight $\theta^* = argmin_\theta \{E(X;\theta)\}$

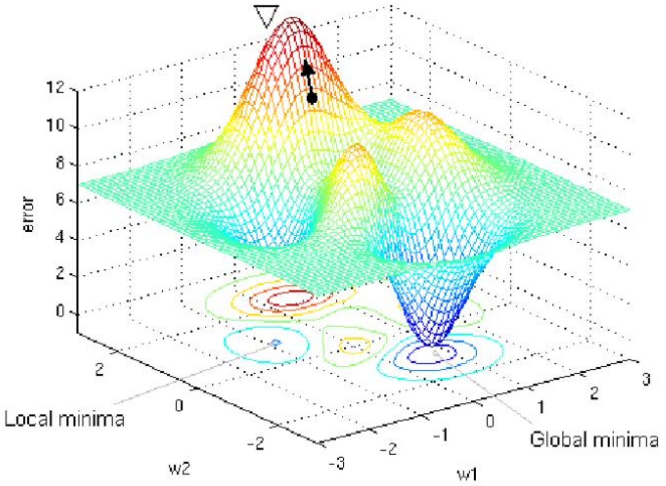
Gradient Descent Method

Method for finding θ^* of highly nonlinear function $E(\theta)$

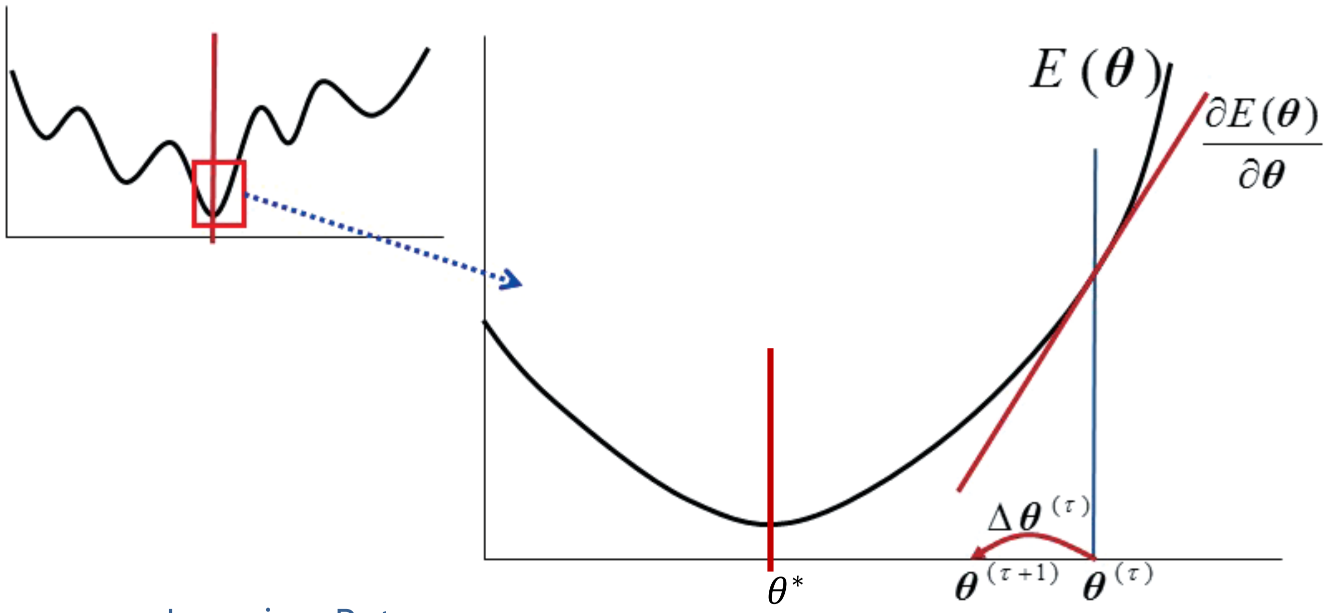
Move in direction to minimize error function (loss function) $E(\theta)$

Error function and optimal weight





Gradient Decent Learning Method

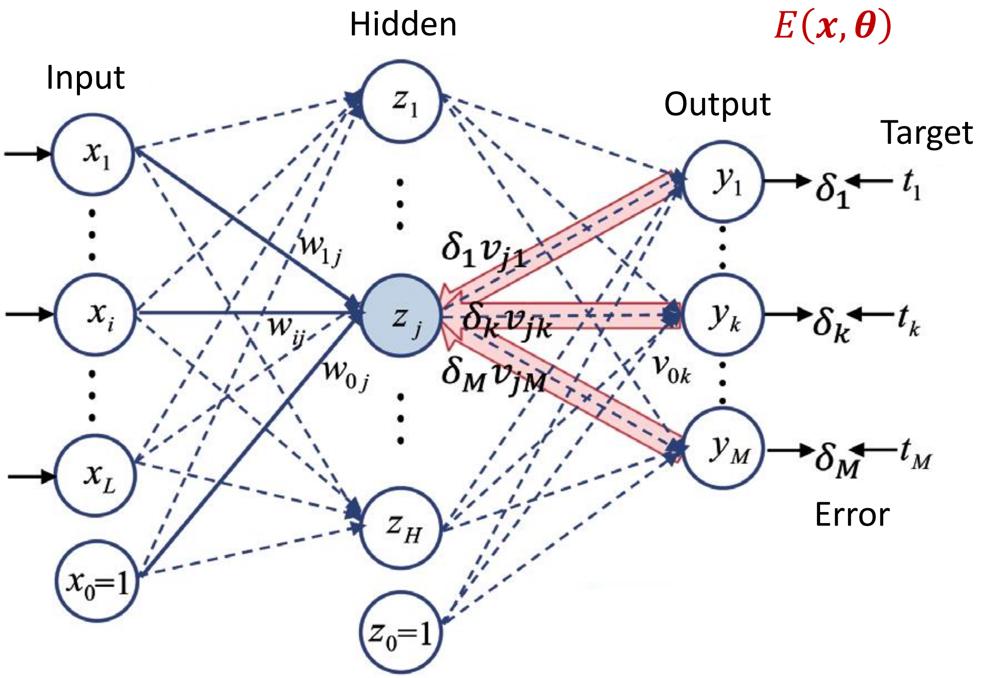


Learning Rate

$$\theta^{(\tau+1)} = \theta^{(\tau)} + \Delta\theta^{(\tau)} = \theta^{(\tau)} - \eta \boxed{\frac{\partial E(\theta^{(\tau)})}{\partial \theta}}$$

Computed by
Error Backpropagation

Layer-wise error propagation from output to input



$$\Delta v_{jk} = -\eta \frac{\partial E}{\partial v_{jk}} = -\eta \delta_k z_j$$

$$\delta_k = -\phi_o'(u_k^o)(t_k - y_k)$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \delta_j x_i$$

$$\delta_j = \phi_h'(u_j^h) \textstyle\sum_{k=1}^{M} \delta_k v_{jk}$$

$$v_{jk}^{(\tau+1)} = v_{jk}^{(\tau+1)} + \Delta v_{jk}$$

$$w_{jk}^{(\tau+1)} = w_{jk}^{(\tau+1)} + \Delta w_{jk}$$

An Application: DIGIT recognition

Mixed National Institute of Standards and Technology

Large handwritten digit classification database

Re-mix of NIST digit databases

60k training images from American Census Bureau employees

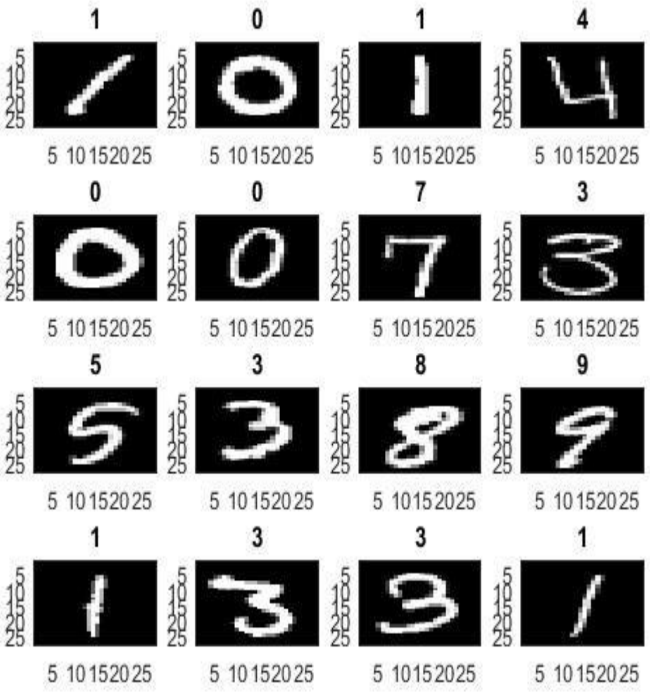
10k testing images from American high school students

Format

Input: 32 x 32 grayscale images (dimension 1024) or 28 x 28 grayscale images (dimension 768)

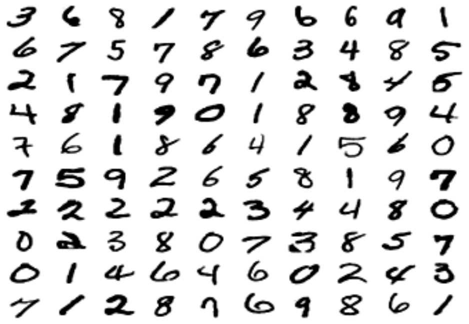
Output: 10 labels (0-9)

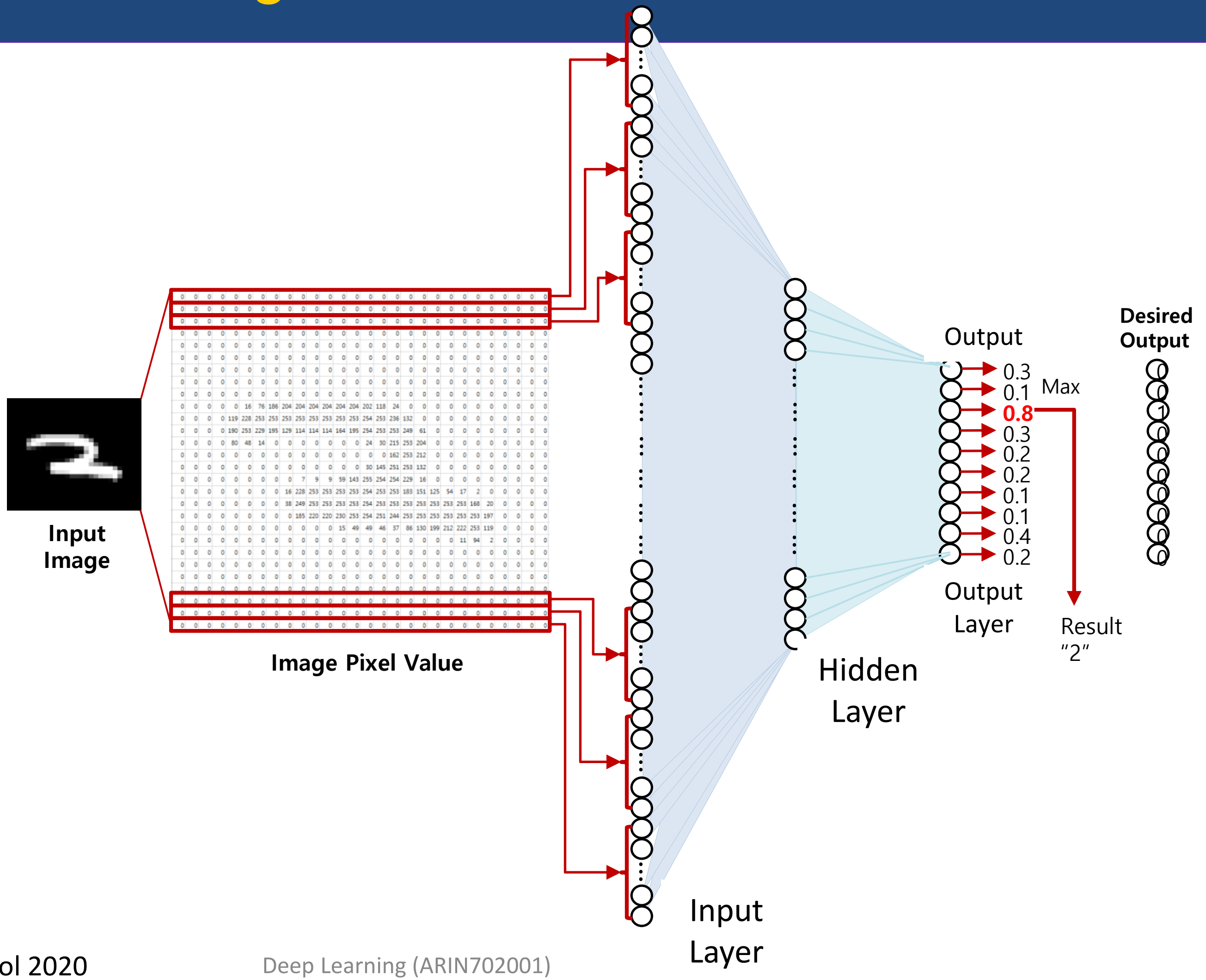
Centered on center of mass



An Application: DIGIT recognition

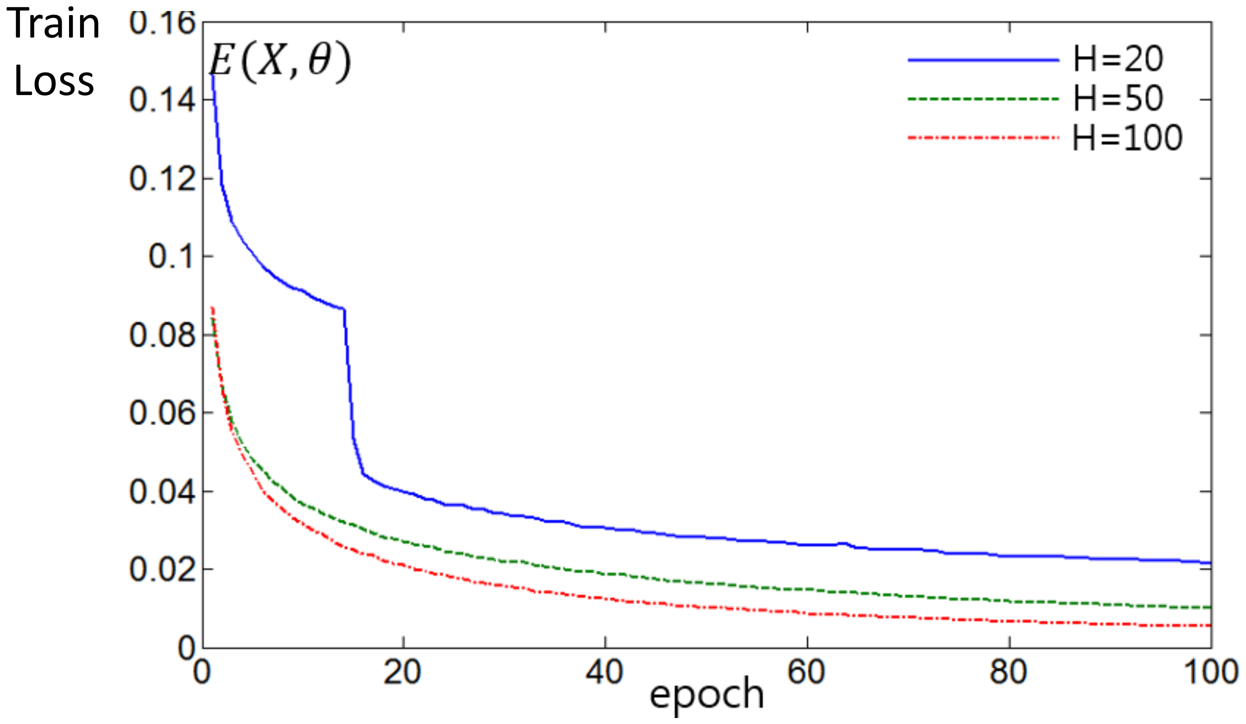
MNIST data: 60000





Learning Curves

Learning curves according to the number of hidden nodes



H.Y. Park et al., KNU Deep Learning School 2020

Conclusion (Remind)

- **I am sure that you can answer following questions if you fully understood this material.**

1. What is Activation function?
2. Why does ReLU make dead neuron?

Thank You!