

CSE 417T

# Introduction to Machine Learning

Lecture 21

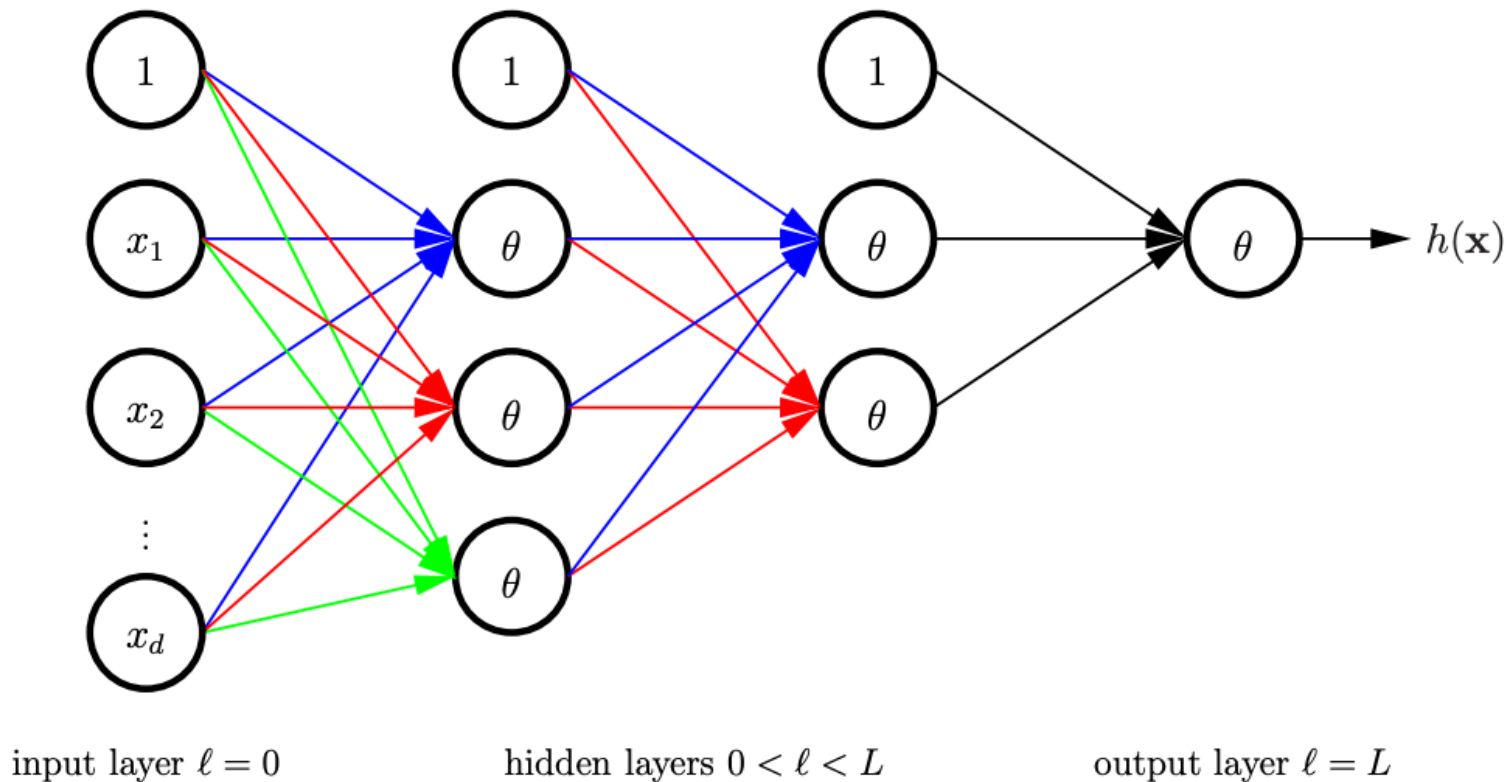
Instructor: Chien-Ju (CJ) Ho

# Logistics

- Homework 5 is due Apr 19 (Tuesday)
- Exam 2 will be on April 28 (Thursday)
  - Will focus on the topics in the second half of the semester
  - Format / logistics will be similar with what we have in Exam 1
    - Timed exam (75 min) during lecture time in the classroom
    - Closed-book exam with 2 letter-size cheat sheets allowed (4 pages in total)
      - No format limitations (it can be typed, written, or a combination)
  - April 26 (Tuesday) will be a review lecture

Recap

# Neural Networks



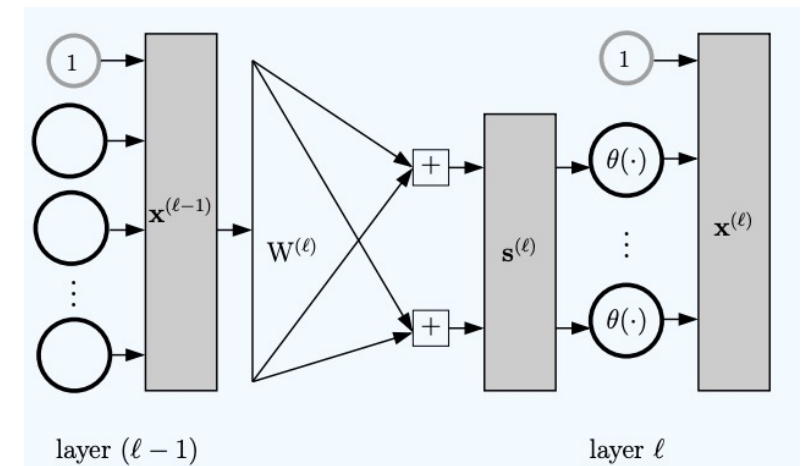
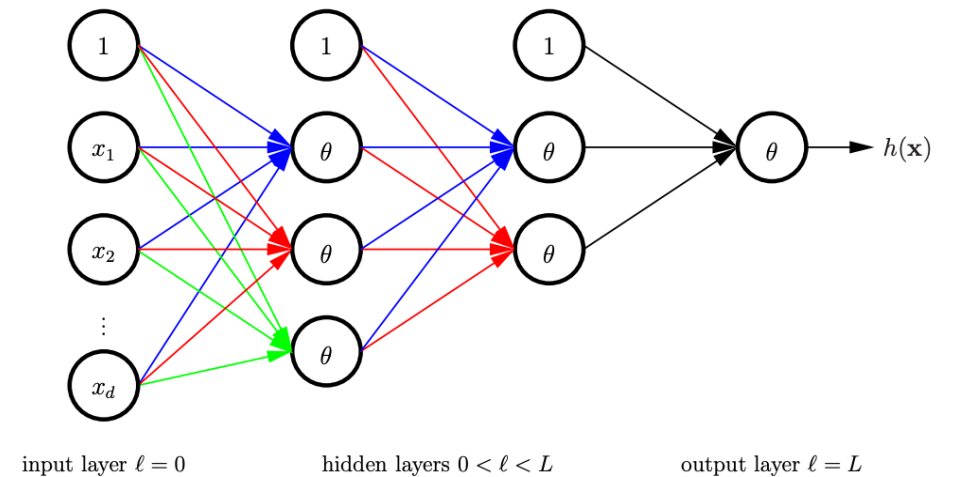
$\theta$ : **activation function**  
(Specify the “activation” of the neuron)



We mostly focus on **feed-forward** network structure

# Notations of Neural Networks (NN)

- Notations:
  - $\ell = 0$  to  $L$ : layer
  - $d^{(\ell)}$ : dimension of layer  $\ell$
  - $\vec{x}^{(\ell)}$ : the nodes in layer  $\ell$
  - $w_{i,j}^{(\ell)}$ : weights; characterize hypothesis in NN
  - $s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{i,j}^{(\ell)} x_i^{(\ell-1)}$ : linear signals
  - $\theta$ : activation function
    - $x_j^{(\ell)} = \theta(s_j^{(\ell)})$



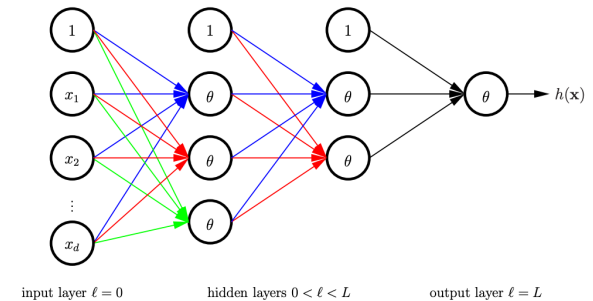
# Forward Propagation (evaluate $h(\vec{x})$ )

- A NN hypothesis  $h$  is characterized by  $\{w_{i,j}^{(\ell)}\}$
- How to evaluate  $h(\vec{x})$ ?

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{W^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{W^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\theta} \mathbf{x}^{(2)} \dots \xrightarrow{W^{(L)}} \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = h(\mathbf{x}).$$

Forward propagation to compute  $h(\mathbf{x})$ :

```
1:  $\mathbf{x}^{(0)} \leftarrow \mathbf{x}$                                 [Initialization]
2: for  $\ell = 1$  to  $L$  do                                [Forward Propagation]
3:    $\mathbf{s}^{(\ell)} \leftarrow (W^{(\ell)})^T \mathbf{x}^{(\ell-1)}$ 
4:    $\mathbf{x}^{(\ell)} \leftarrow \begin{bmatrix} 1 \\ \theta(\mathbf{s}^{(\ell)}) \end{bmatrix}$ 
5: end for
6:  $h(\mathbf{x}) = \mathbf{x}^{(L)}$                                 [Output]
```



Given weights  $w_{i,j}^{(\ell)}$  and  $\vec{x}^{(0)} = \vec{x}$ , we can calculate all  $\vec{x}^{(\ell)}$  and  $\vec{s}^{(\ell)}$  through forward propagation.

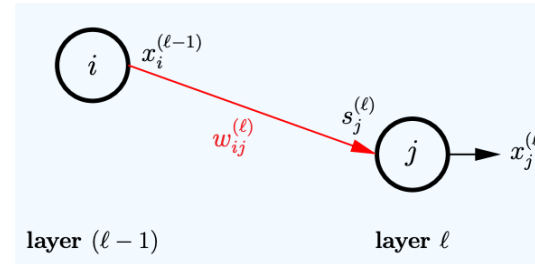
# How to Learn NN From Data?

- Given  $D$ , how to learn the weights  $W = \{w_{i,j}^{(\ell)}\}$ ?
- Intuition: Minimize  $E_{in}(W) = \frac{1}{N} \sum_{n=1}^N e_n(W)$
- How?
  - Gradient descent:  $W(t+1) \leftarrow W(t) - \eta \nabla_W E_{in}(W)$
  - Stochastic gradient descent  $W(t+1) \leftarrow W(t) - \eta \nabla_W e_n(W)$
- Key step: we need to be able to evaluate the gradient...
  - Not trivial given the network structure
  - **Backpropagation** is an algorithmic procedure to calculate the gradient

# Compute the Gradient $\nabla_W e_n(W)$

- Applying chain rule

$$\frac{\partial e_n(W)}{\partial w_{i,j}^{(\ell)}} = \frac{\partial e_n(W)}{\partial s_j^{(\ell)}} \frac{\partial s_j^{(\ell)}}{\partial w_{i,j}^{(\ell)}} = \delta_j^{(\ell)} x_i^{(\ell-1)}$$



- Calculating  $\delta_j^{(\ell)}$  (Using dynamic programming idea)

- Boundary conditions

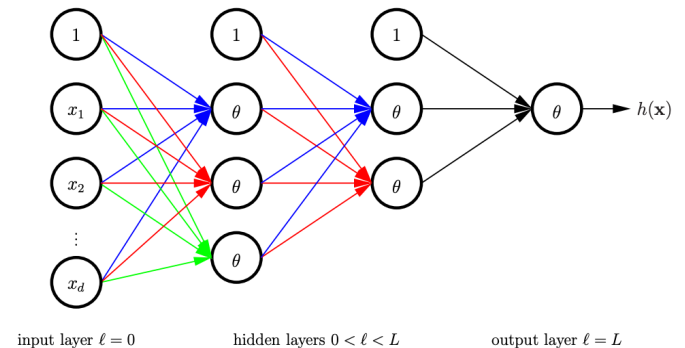
- The output layer (assume regression)

- $\delta_1^{(L)} = 2 \left( s_1^{(L)} - y_n \right)$  (generalizable to other differentiable error)

- Backward recursive formulation

- $$\delta_j^{(\ell)} = \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n(W)}{\partial s_k^{(\ell+1)}} \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}} = \sum_{k=1}^{d^{(\ell+1)}} \delta_k^{(\ell+1)} w_{j,k}^{(\ell+1)} \theta' \left( s_j^{(\ell)} \right)$$

- Backward propagation





# Backpropagation Algorithm

- Recall that  $\frac{\partial e_n(W)}{\partial w_{i,j}^{(\ell)}} = \delta_j^{(\ell)} x_i^{(\ell-1)}$
- Backpropagation Algorithm
  - Initialize  $w_{i,j}^{(\ell)}$  randomly
  - For  $t = 1$  to  $T$ 
    - Randomly pick a point from  $D$  (for stochastic gradient descent)
    - Forward propagation: Calculate all  $x_i^{(\ell)}$  and  $s_i^{(\ell)}$
    - Backward propagation: Calculate all  $\delta_j^{(\ell)}$
    - Update the weights  $w_{i,j}^{(\ell)} \leftarrow w_{i,j}^{(\ell)} - \eta \delta_j^{(\ell)} x_i^{(\ell-1)}$
- Return the weights

# Discussion

- Backpropagation is gradient descent with efficient gradient computation
- Note that the  $E_{in}$  is **not convex** in weights
- Gradient descent doesn't guarantee to converge to global optimal
- Common approaches:
  - Run it many times
  - Each with a different initialization (the choice of initialization matters)

# Today's Lecture

The notes are not intended to be comprehensive. They should be accompanied by lectures and/or textbook.  
Let me know if you spot errors.

# Neural Network is Expressive

- Universal approximation theorem:
  - A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of  $\mathbb{R}^n$ , under mild assumptions on the activation function.
  - A single-hidden-layer NN can approximate ANY continuous target function!
- We also seem to only discuss how to minimize  $E_{in}$

What about overfitting?

# Regularization in Neural Networks

# Weight-Based Regularization

- Weight decay

$$E_{aug}(W) = E_{in}(W) + \frac{\lambda}{N} \sum_{i,j,\ell} \left( w_{i,j}^{(\ell)} \right)^2$$

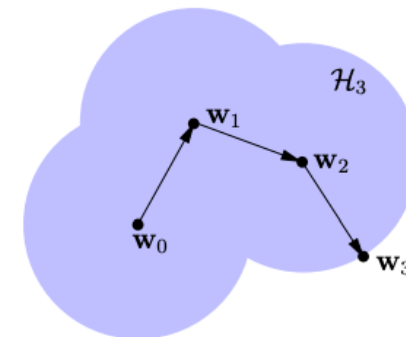
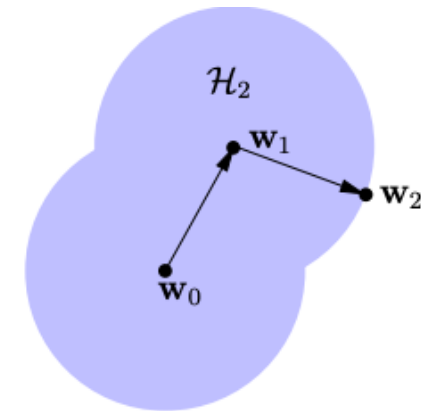
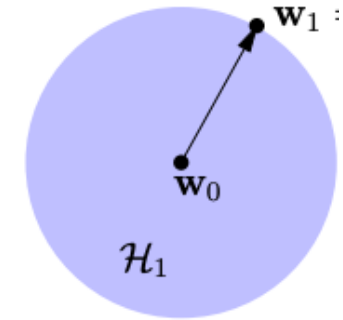
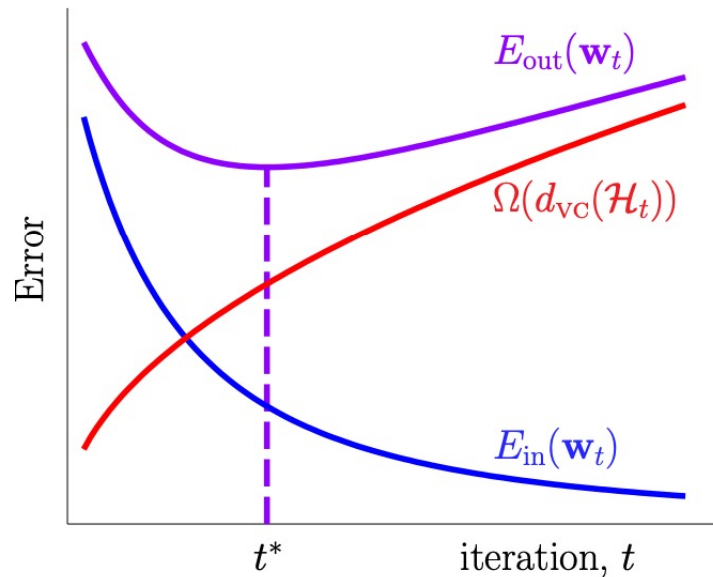
- Weight elimination

$$E_{aug}(W) = E_{in}(W) + \frac{\lambda}{N} \sum_{i,j,\ell} \frac{\left( w_{i,j}^{(\ell)} \right)^2}{1 + \left( w_{i,j}^{(\ell)} \right)^2}$$

- When  $w_{i,j}^{(\ell)}$  is small, approximates weight decay
- When  $w_{i,j}^{(\ell)}$  is large, approximates adding a constant (no impacts to gradient)
- “Decaying” more on smaller weights (i.e., eliminating small weights)

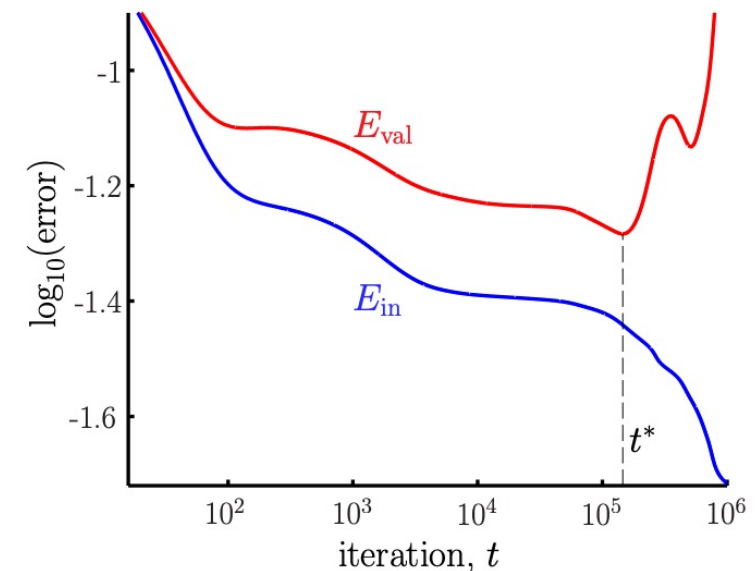
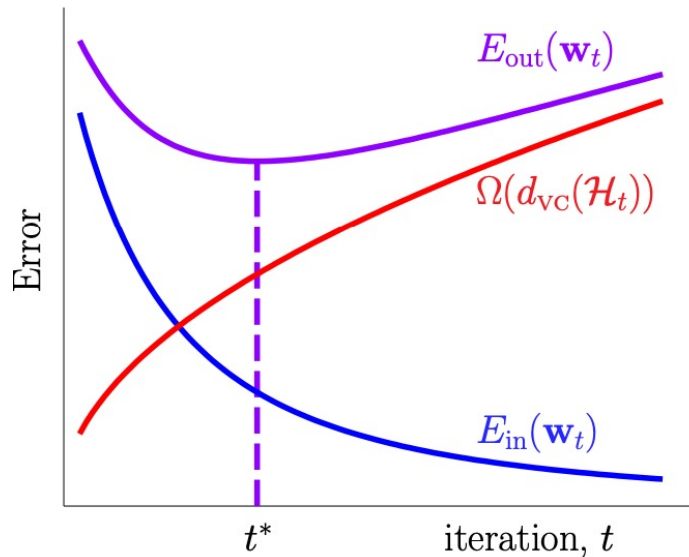
# Early Stopping

- Consider gradient descent (GD)
  - $H_1$ : the set of hypothesis GD can reach at  $t = 1$
  - $H_2$ : the set of hypothesis GD can reach at  $t = 2$
  - ...
  - $H_1 \subseteq H_2 \subseteq H_3 \subseteq \dots$



# Early Stopping

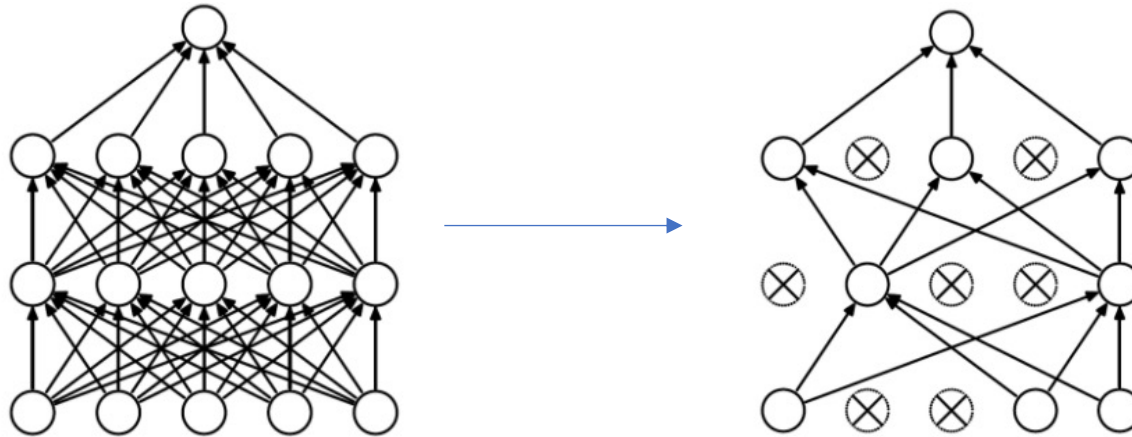
- Stopping gradient descent early is a regularization method
  - **Constrain** the hypothesis set
- How to find the optimal stopping point  $t^*$ ?
  - Using validation is a common approach





# Dropout

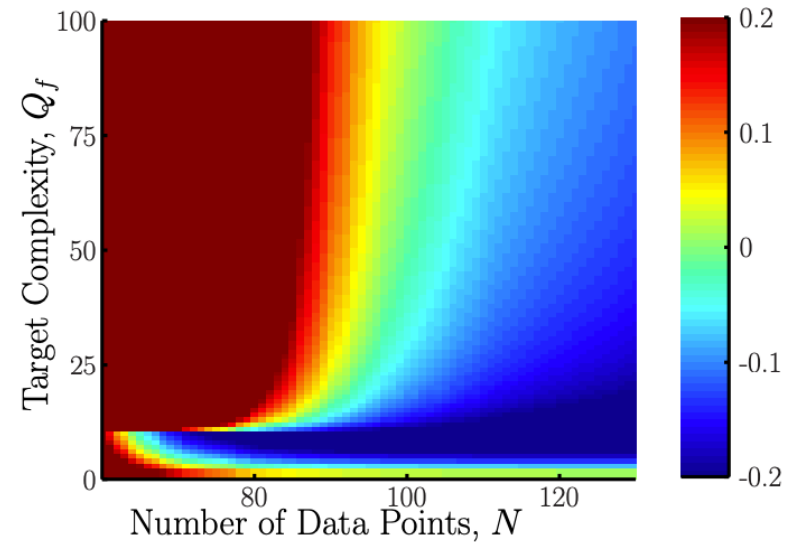
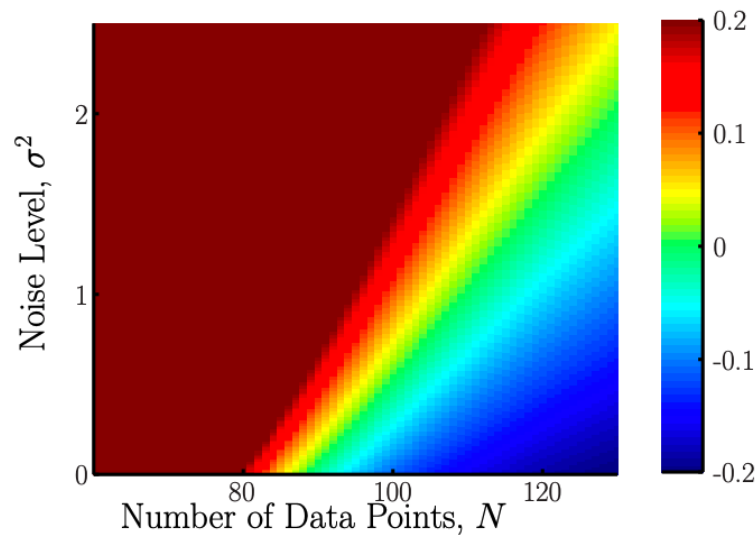
- Neural networks is very expressive (low bias, potentially high variance)
- Dropout
  - Randomly **drop**  $p$  portion of the weights during training



- Learn many models with dropout
- **Average** them during prediction (reduce weights by a ratio of  $p$ )

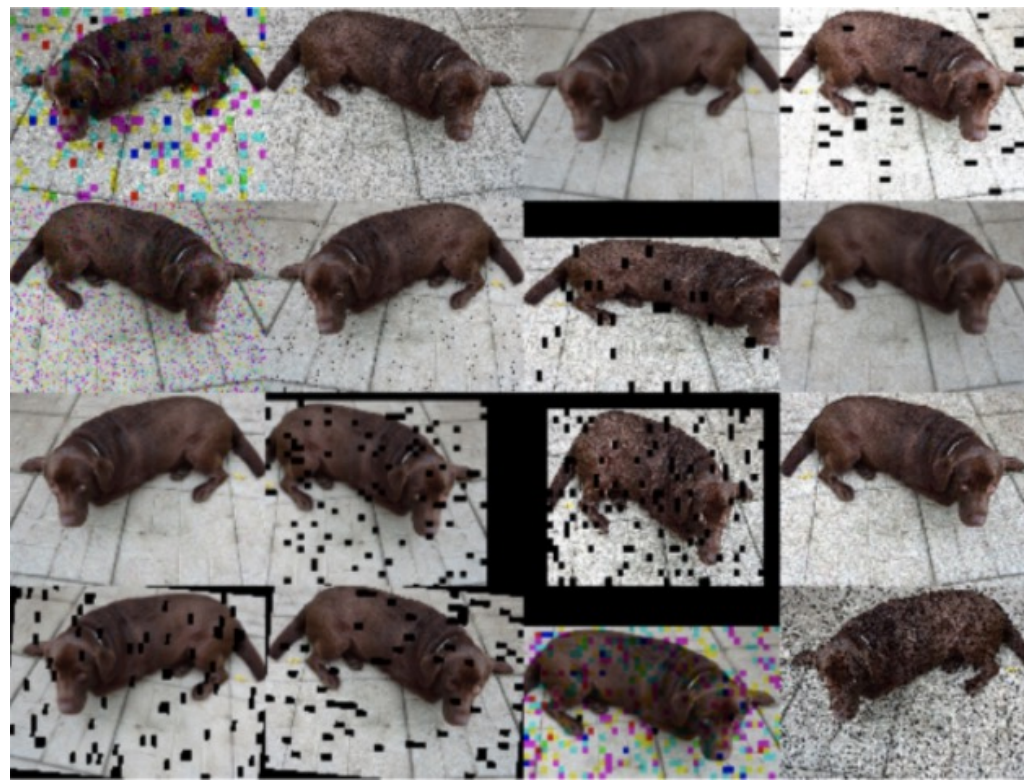
# A Nontraditional Method to Avoid Overfitting

- What's the cause of overfitting?



- Fitting the **noise** instead of the target
- Regularization: Constrain  $H$  so it's not that powerful to fit noise
- How about **adding noises** to data?

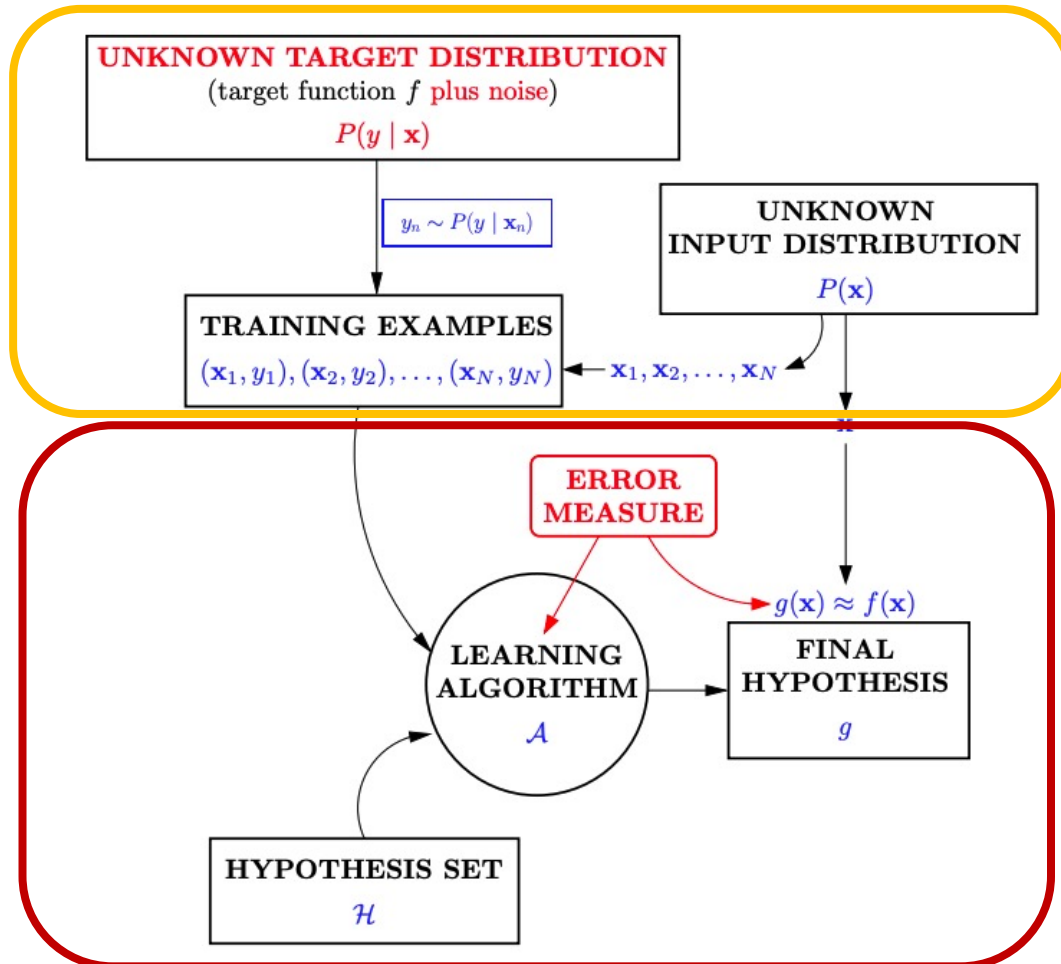
# Adding Noises as Regularization



A bit discussion on optimization

# Revisit the Learning Setup

Learning problem given to us



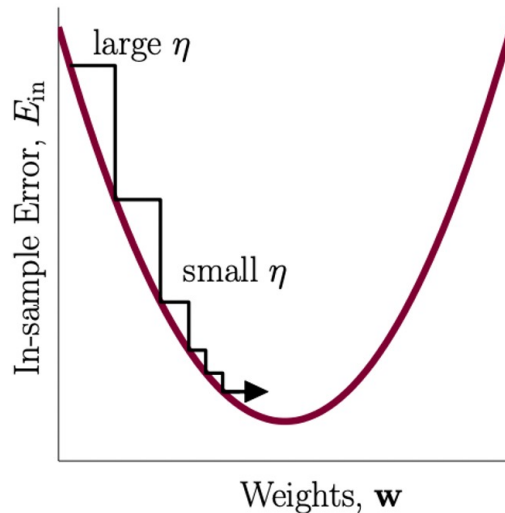
Our focus of this course

Discussion of different learning models:

Given dataset  $D$  and a hypothesis set  $H$ , find a hypothesis  $h \in H$  that minimizes some error  $E(h)$

# Learning as an Optimization Problem

- Minimize error  $E(h)$
- We have talked about gradient descent



- An iterative method of the form:  
$$\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$$
- $\vec{v}_t$ : a unit vector, determining the direction of the update
  - $\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{in}(\vec{w}(t))}{\|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|}$
  - The opposite direction of  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$
  - Coming from first-order approximation
- $\eta_t$ : a scalar, determining how much to update
  - $\eta_t = \eta \|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|$
  - set  $\eta_t$  smaller when  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$  is smaller (could be closer to the minimum)
- Gradient descent:  $\vec{w}(t+1) \leftarrow \vec{w}(t) - \eta \nabla_{\vec{w}} E_{in}(\vec{w}(t))$

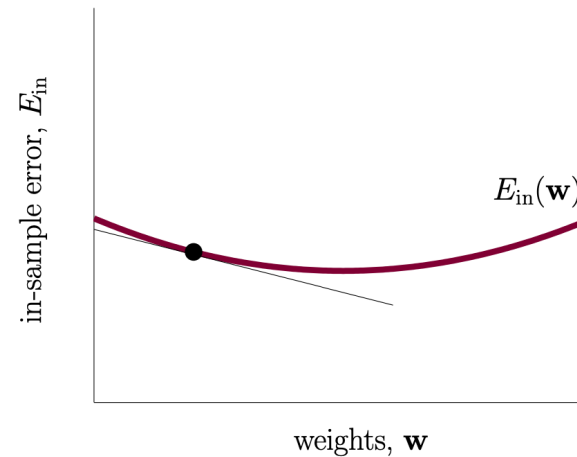
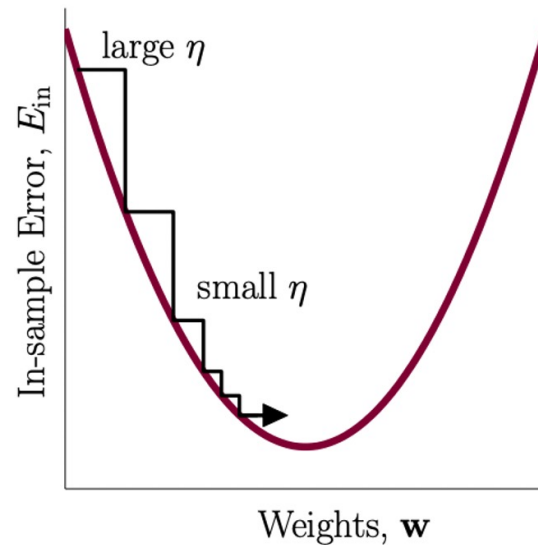
# Variations

- Can we choose a different learning rate?

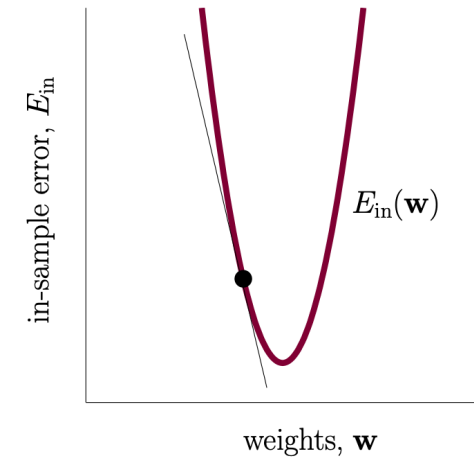
- An iterative method of the form:

$$\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$$

- $\vec{v}_t$ : a unit vector, determining the direction of the update
  - $\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{in}(\vec{w}(t))}{\|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|}$ 
    - The opposite direction of  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$
    - Coming from first-order approximation
- $\eta_t$ : a scalar, determining how much to update
  - $\eta_t = \eta \|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|$
  - set  $\eta_t$  smaller when  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$  is smaller (which might be closer to be the minimum)



Shallow: use large  $\eta$ .



Deep: use small  $\eta$ .

# Variations

- Can we choose a different learning rate?

- Variable learning rate gradient descent

- Intuition:

- Start with some learning rate  $\eta$
    - Do gradient descent
    - If the update leads to a “smaller” error
      - Slightly increase  $\eta$  in the next step
    - If the update leads to a “larger error”
      - Don’t update
      - Decrease  $\eta$  and re-do it again

- An iterative method of the form:

$$\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$$

- $\vec{v}_t$ : a unit vector, determining the direction of the update
  - $\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{in}(\vec{w}(t))}{\|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|}$ 
    - The opposite direction of  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$
    - Coming from first-order approximation
- $\eta_t$ : a scalar, determining how much to update
  - $\eta_t = \eta \|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|$
  - set  $\eta_t$  smaller when  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$  is smaller (which might be closer to be the minimum)

## Variable Learning Rate Gradient Descent:

- 1: Initialize  $\mathbf{w}(0)$ , and  $\eta_0$  at  $t = 0$ . Set  $\alpha > 1$  and  $\beta < 1$ .
- 2: **while** stopping criterion has not been met **do**
- 3:   Let  $\mathbf{g}(t) = \nabla E_{in}(\mathbf{w}(t))$ , and set  $\mathbf{v}(t) = -\mathbf{g}(t)$ .
- 4:   **if**  $E_{in}(\mathbf{w}(t) + \eta_t \mathbf{v}(t)) < E_{in}(\mathbf{w}(t))$  **then**
- 5:     accept:  $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta_t \mathbf{v}(t)$ ;  $\eta_{t+1} = \alpha \eta_t$ .
- 6:   **else**
- 7:     reject:  $\mathbf{w}(t+1) = \mathbf{w}(t)$ ;  $\eta_{t+1} = \beta \eta_t$ .
- 8:   Iterate to the next step,  $t \leftarrow t + 1$ .



# Variations

- Can we choose a different learning rate?
- Steepest descent
  - Choosing  $\eta$  that minimizes the error

## Steepest Descent (Gradient Descent + Line Search):

- 1: Initialize  $\mathbf{w}(0)$  and set  $t = 0$ ;
- 2: **while** stopping criterion has not been met **do**
- 3:   Let  $\mathbf{g}(t) = \nabla E_{\text{in}}(\mathbf{w}(t))$ , and set  $\mathbf{v}(t) = -\mathbf{g}(t)$ .
- 4:   Let  $\eta^* = \operatorname{argmin}_{\eta} E_{\text{in}}(\mathbf{w}(t) + \eta \mathbf{v}(t))$ .
- 5:    $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta^* \mathbf{v}(t)$ .
- 6:   Iterate to the next step,  $t \leftarrow t + 1$ .

- An iterative method of the form:

$$\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$$

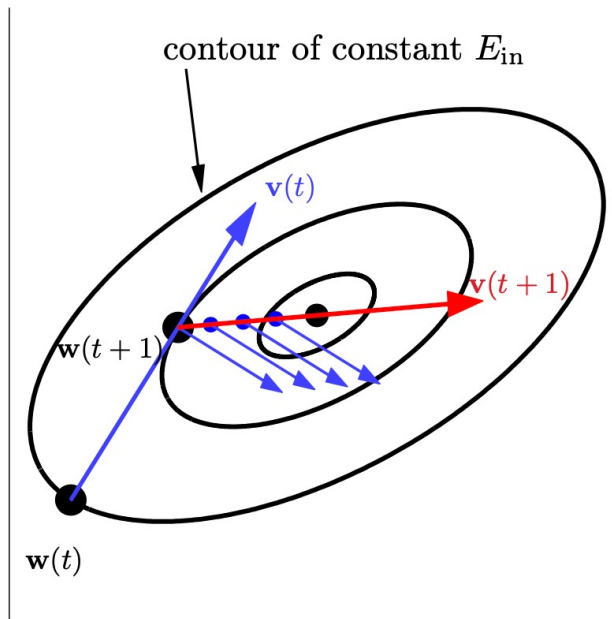
- $\vec{v}_t$ : a unit vector, determining the direction of the update
  - $\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{\text{in}}(\vec{w}(t))}{\|\nabla_{\vec{w}} E_{\text{in}}(\vec{w}(t))\|}$ 
    - The opposite direction of  $\nabla_{\vec{w}} E_{\text{in}}(\vec{w}(t))$
    - Coming from first-order approximation
- $\eta_t$ : a scalar, determining how much to update
  - $\eta_t = \eta \|\nabla_{\vec{w}} E_{\text{in}}(\vec{w}(t))\|$
  - set  $\eta_t$  smaller when  $\nabla_{\vec{w}} E_{\text{in}}(\vec{w}(t))$  is smaller (which might be closer to be the minimum)

- Line search
  - Doubling trick:
    - Find the interval containing the minimum
  - Bisection algorithm
    - Apply binary search within the interval

# Variations

- Can we choose a different update direction?

- Conjugate gradient



- Intuition:
  - Apply steepest descent in each step
  - Choose the update “direction” that is orthogonal to the previous direction
  - Why?
    - Steepest descent has reached the “minimum” in the update direction
    - It might help to not “redo” the work in the same direction; therefore, choose an orthogonal direction

- An iterative method of the form:

$$\vec{\mathbf{w}}(t+1) \leftarrow \vec{\mathbf{w}}(t) + \eta_t \vec{\mathbf{v}}_t$$

- $\vec{\mathbf{v}}_t$ : a unit vector, determining the direction of the update
  - $\vec{\mathbf{v}}_t = \frac{-\nabla_{\vec{\mathbf{w}}} E_{in}(\vec{\mathbf{w}}(t))}{\|\nabla_{\vec{\mathbf{w}}} E_{in}(\vec{\mathbf{w}}(t))\|}$ 
    - The opposite direction of  $\nabla_{\vec{\mathbf{w}}} E_{in}(\vec{\mathbf{w}}(t))$
    - Coming from first-order approximation
- $\eta_t$ : a scalar, determining how much to update
  - $\eta_t = \eta \|\nabla_{\vec{\mathbf{w}}} E_{in}(\vec{\mathbf{w}}(t))\|$
  - set  $\eta_t$  smaller when  $\nabla_{\vec{\mathbf{w}}} E_{in}(\vec{\mathbf{w}}(t))$  is smaller (which might be closer to be the minimum)

# Variations

- Beyond first-order optimization

- Why do we choose (negative) gradient as the update direction?

- Intuition: Choose  $\vec{v}_t$  that moves towards the “steepest” direction

- Approaching the minimum faster

- Taylor’s approximation:

- $E_{in}(\vec{w}(t) + \eta_t \vec{v}_t) = E_{in}(\vec{w}(t)) + \eta_t \nabla_{\vec{w}} E_{in}(\vec{w}(t))^T \vec{v}_t + O(\eta_t^2)$

- $E_{in}(\vec{w}(t + 1)) - E_{in}(\vec{w}(t)) \approx \eta_t \nabla_{\vec{w}} E_{in}(\vec{w}(t))^T \vec{v}_t$

We have skipped higher-order terms

- We might leverage higher-order information for more efficient learning

- An iterative method of the form:

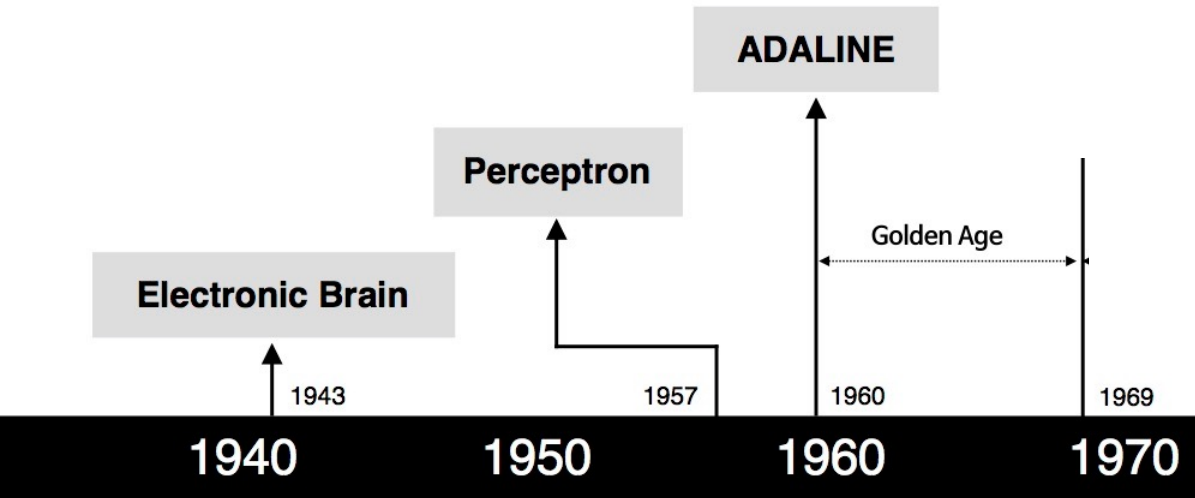
$$\vec{w}(t + 1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$$

- $\vec{v}_t$ : a unit vector, determining the direction of the update
  - $\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{in}(\vec{w}(t))}{\|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|}$ 
    - The opposite direction of  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$
    - Coming from first-order approximation
- $\eta_t$ : a scalar, determining how much to update
  - $\eta_t = \eta \|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|$
  - set  $\eta_t$  smaller when  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$  is smaller (which might be closer to be the minimum)

Short Break and Q&A

# Deep Learning

# Brief/Informal History



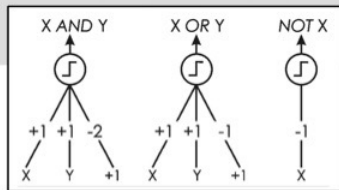
S. McCulloch - W. Pitts



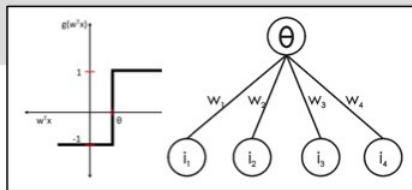
F. Rosenblatt



B. Widrow - M. Hoff



- Adjustable Weights
- Weights are not Learned



- Learnable Weights and Threshold

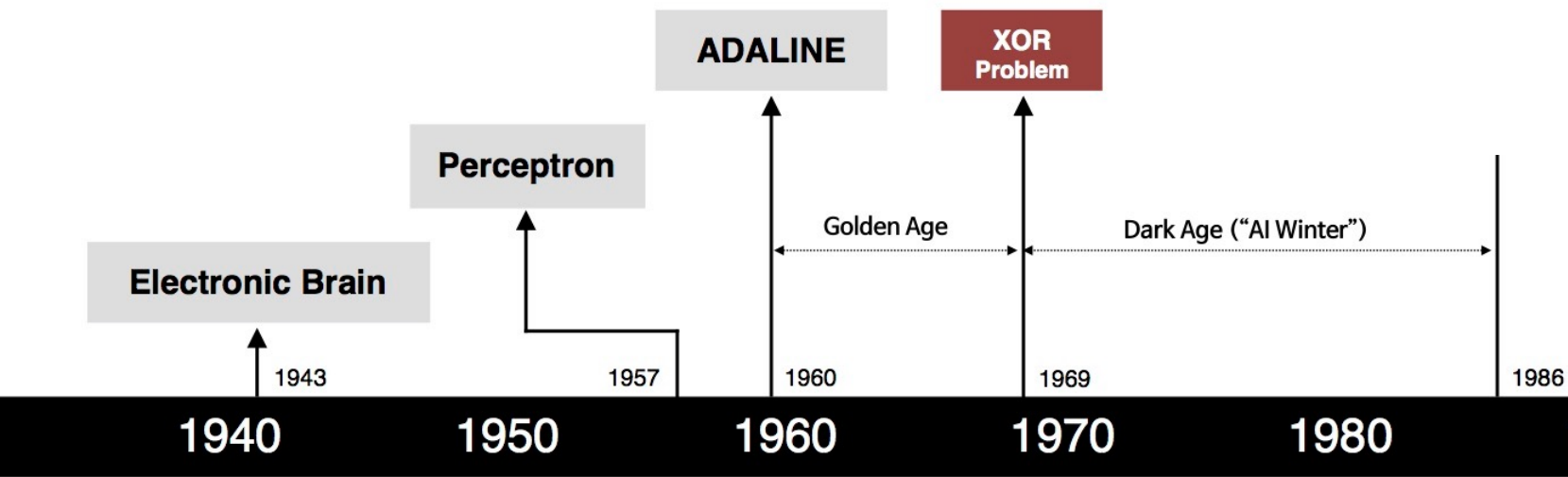
# NEW NAVY DEVICE LEARNS BY DOING


Psychologist Shows Embryo  
of Computer Designed to  
Read and Grow Wiser

WASHINGTON, July 7 (UPI)  
—The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

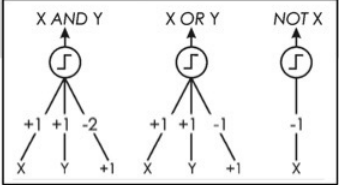


Minsky: “However, I started to worry about what such a machine could not do. For example, it could tell ‘E’s from ‘F’s, and ‘5’s from ‘6’s—things like that. But when there were disturbing stimuli near these figures that weren’t correlated with them the recognition was destroyed.”






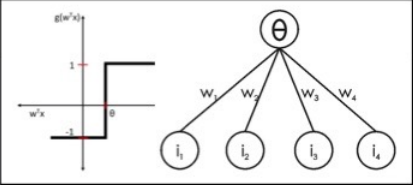
S. McCulloch – W. Pitts




- Adjustable Weights
- Weights are not Learned




F. Rosenblatt



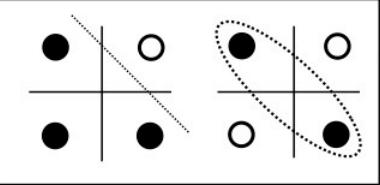
- Learnable Weights and Threshold



B. Widrow – M. Hoff

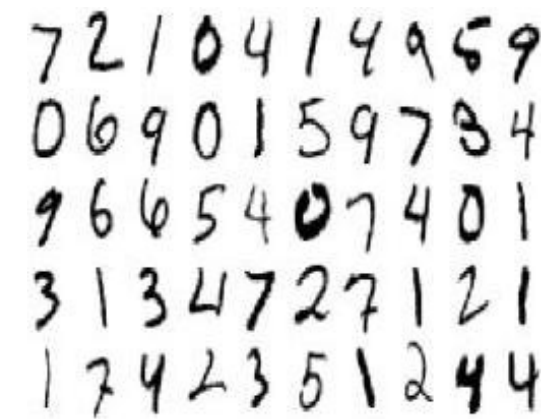
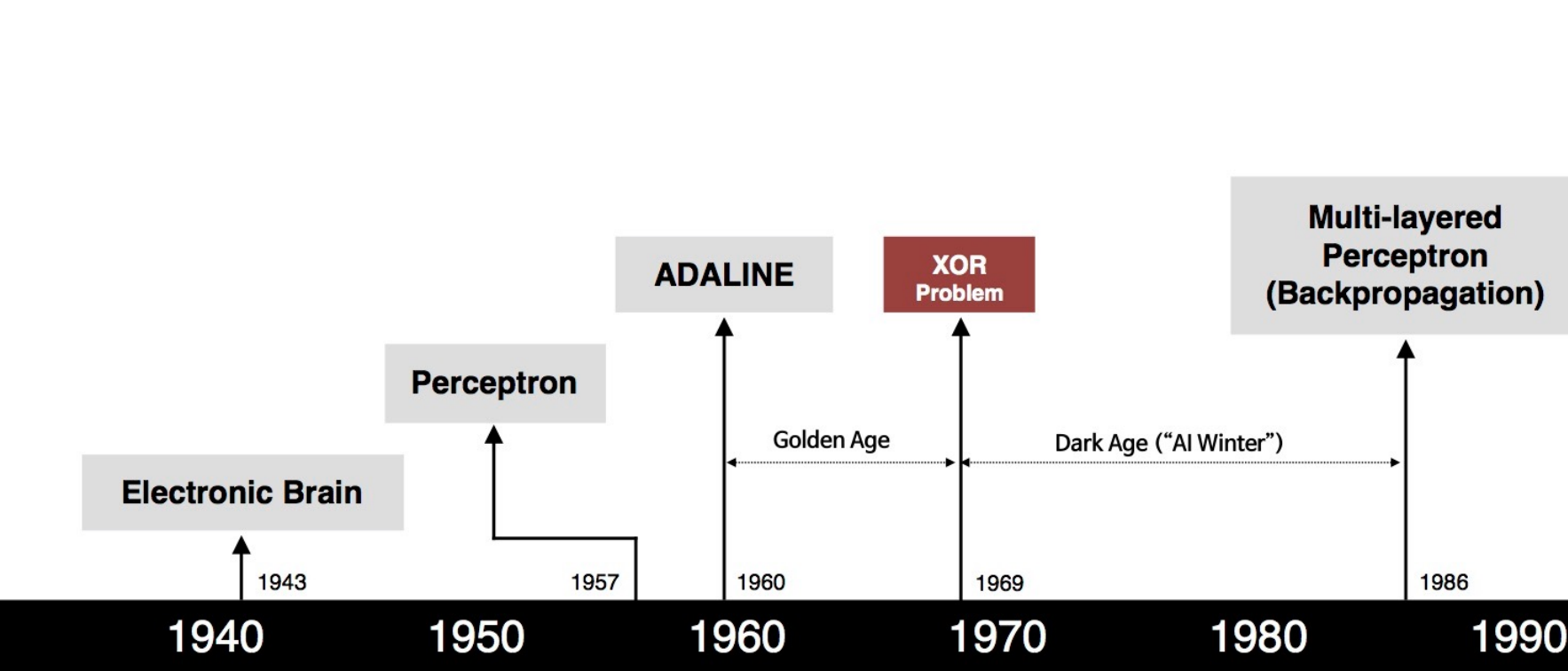


M. Minsky – S. Papert

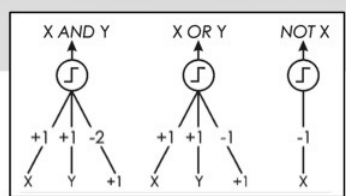


- XOR Problem





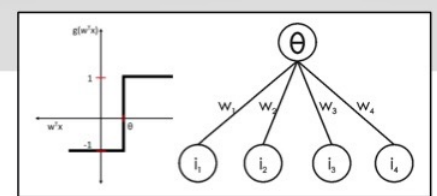
S. McCulloch - W. Pitts



- Adjustable Weights
- Weights are not Learned



F. Rosenblatt



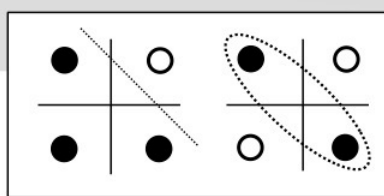
- Learnable Weights and Threshold



B. Widrow - M. Hoff



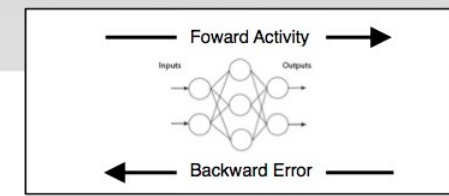
M. Minsky - S. Papert



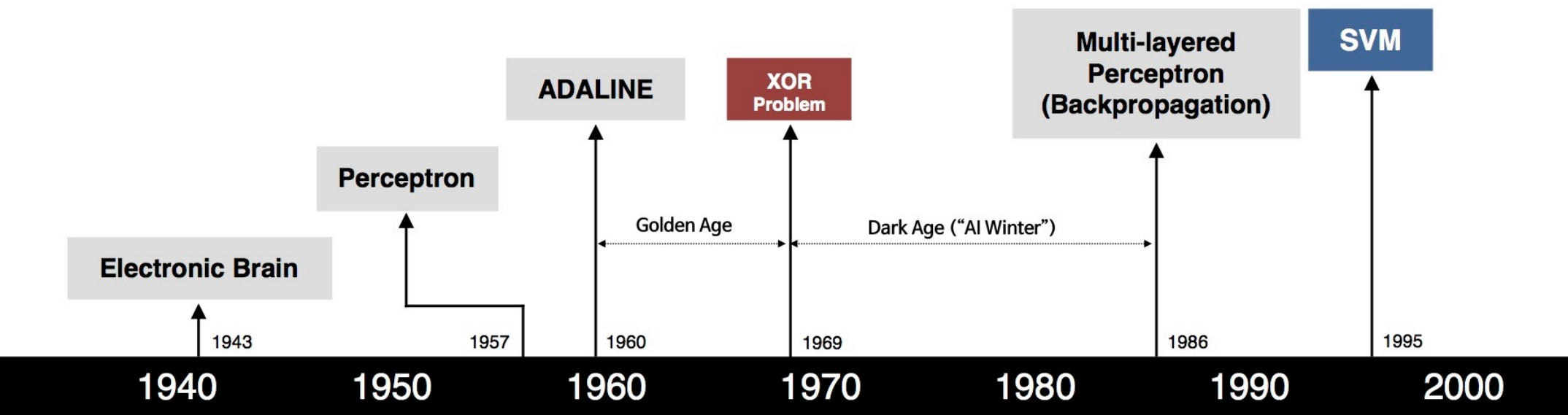
- XOR Problem



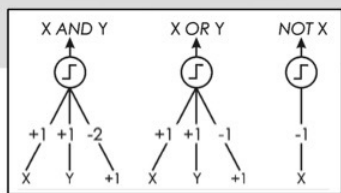
D. Rumelhart - G. Hinton - R. Williams



- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting



S. McCulloch – W. Pitts



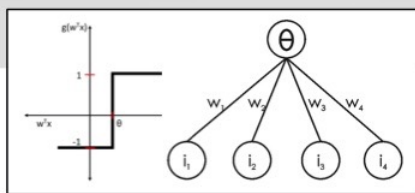
- Adjustable Weights
- Weights are not Learned



F. Rosenblatt



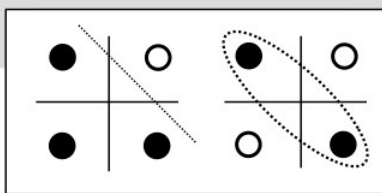
B. Widrow – M. Hoff



- Learnable Weights and Threshold



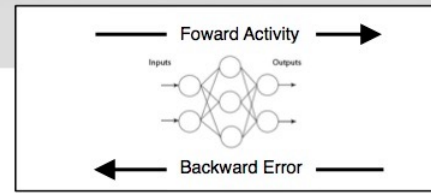
M. Minsky – S. Papert



- XOR Problem



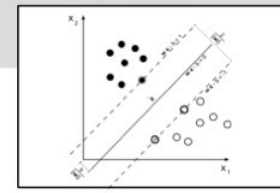
D. Rumelhart – G. Hinton – R. Williams



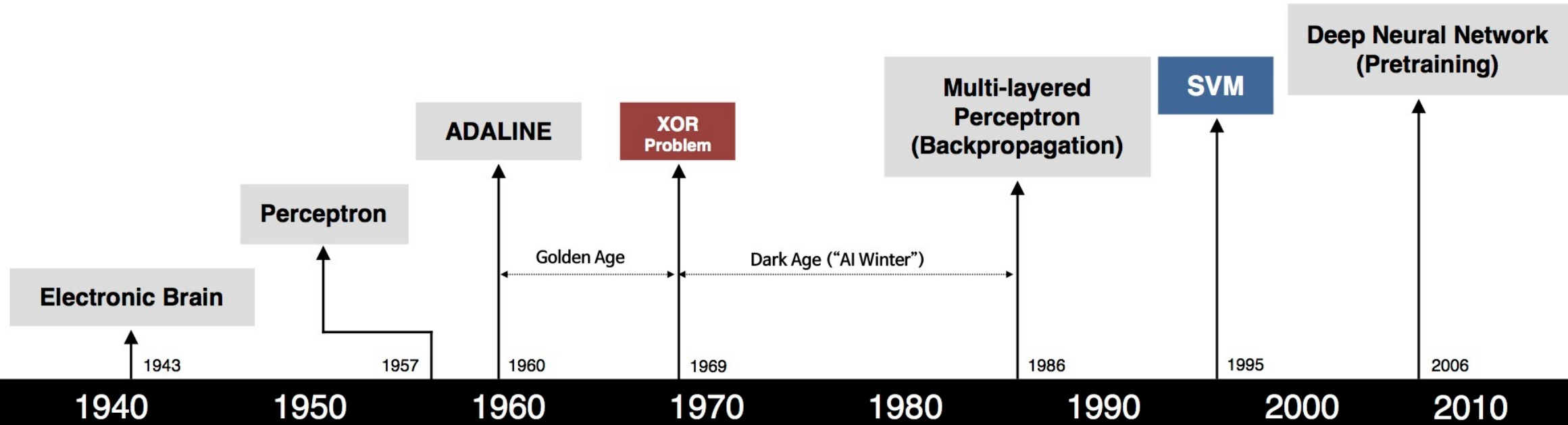
- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting



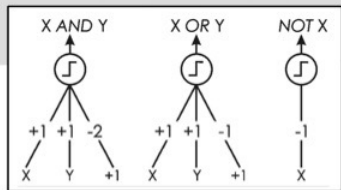
V. Vapnik – C. Cortes



- Limitations of learning prior knowledge
- Kernel function: Human Intervention



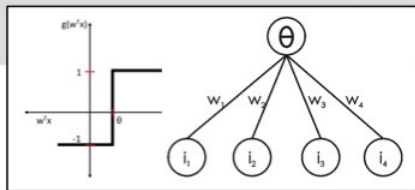
S. McCulloch – W. Pitts



- Adjustable Weights
- Weights are not Learned



F. Rosenblatt



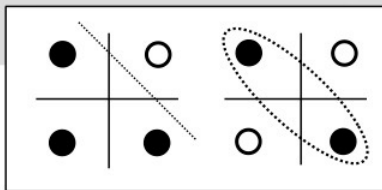
- Learnable Weights and Threshold



B. Widrow – M. Hoff



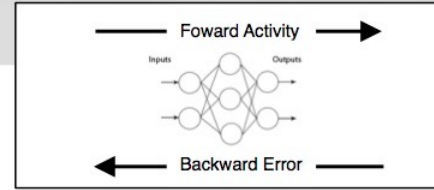
M. Minsky – S. Papert



- XOR Problem



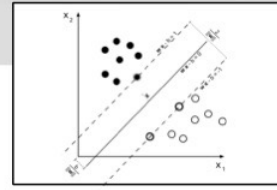
D. Rumelhart – G. Hinton – R. Williams



- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting



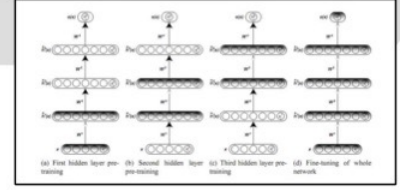
V. Vapnik – C. Cortes



- Limitations of learning prior knowledge
- Kernel function: Human Intervention



G. Hinton – S. Ruslan



- Hierarchical feature Learning

# ImageNet Challenge 2012

## Task 1: Classification



Car

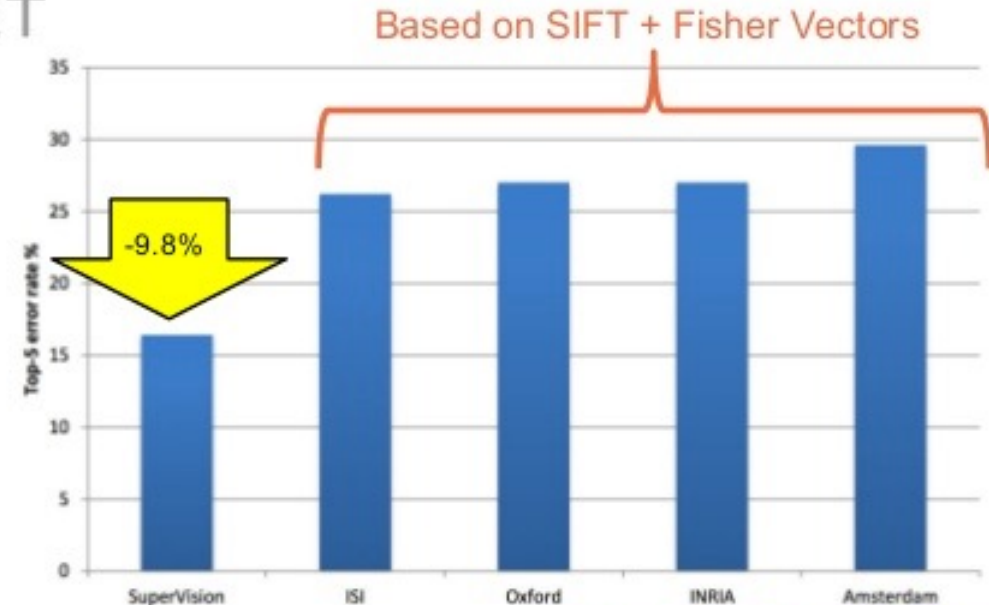
- Predict a class label
- 5 predictions / image
- 1000 classes
- 1,200 images per class for training
- Bounding boxes for 50% of training.

## ImageNet Challenge

Image Classification 2012

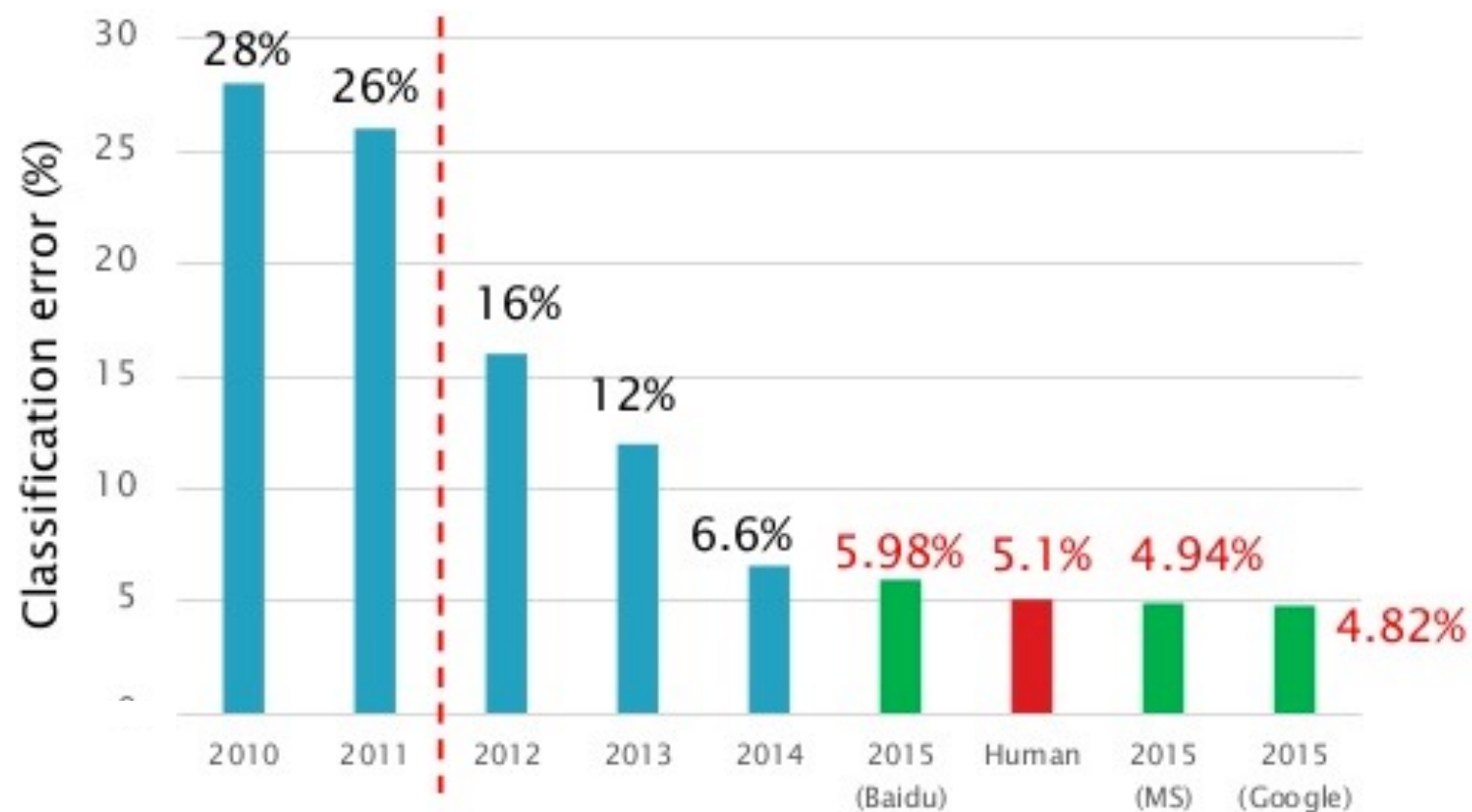
IMAGENET

Slide credit:  
[Rob Fergus](#) (NYU)



Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2014). [Imagenet large scale visual recognition challenge](#). *arXiv preprint arXiv:1409.0575*. [\[wsh\]](#)





He et al., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", arXiv, 2015.

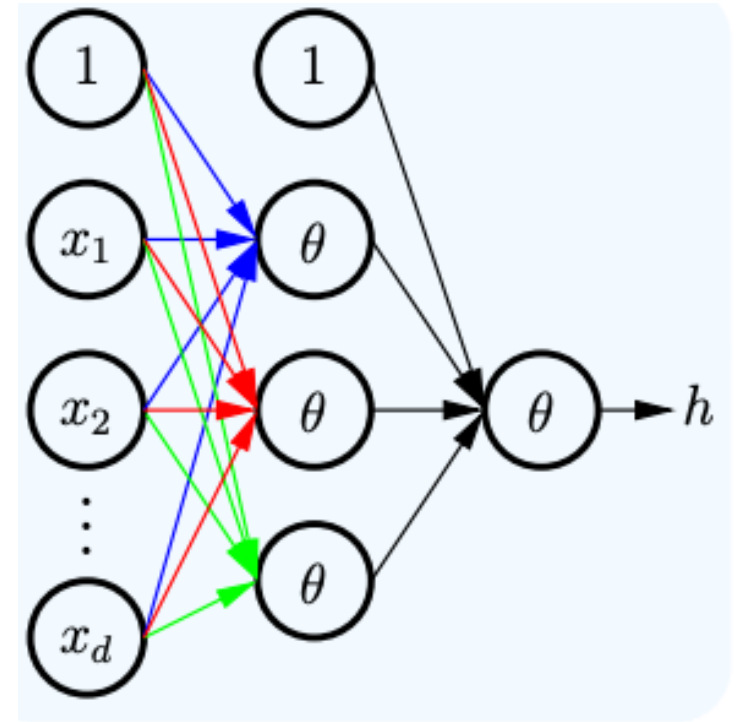
Ioffe et al., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv, 2015.

# What is “Deep” Learning

Neural networks with many layers

# Single Hidden-Layer Neural Network

- How do we write a hypothesis in single-hidden layer NN mathematically?



# Single Hidden-Layer Neural Network

- How do we write a hypothesis in single-hidden layer NN mathematically?

- $$h(\vec{x}) = \theta \left( w_{0,1}^{(2)} + \sum_{j=1}^{d^{(1)}} w_{j,1}^{(2)} x_j^{(1)} \right)$$
$$= \theta \left( w_{0,1}^{(2)} + \sum_{j=1}^{d^{(1)}} w_{j,1}^{(2)} \theta \left( \sum_{i=0}^{d^{(0)}} w_{i,j}^{(1)} x_i^{(0)} \right) \right)$$

- How do we write a linear model with nonlinear transform

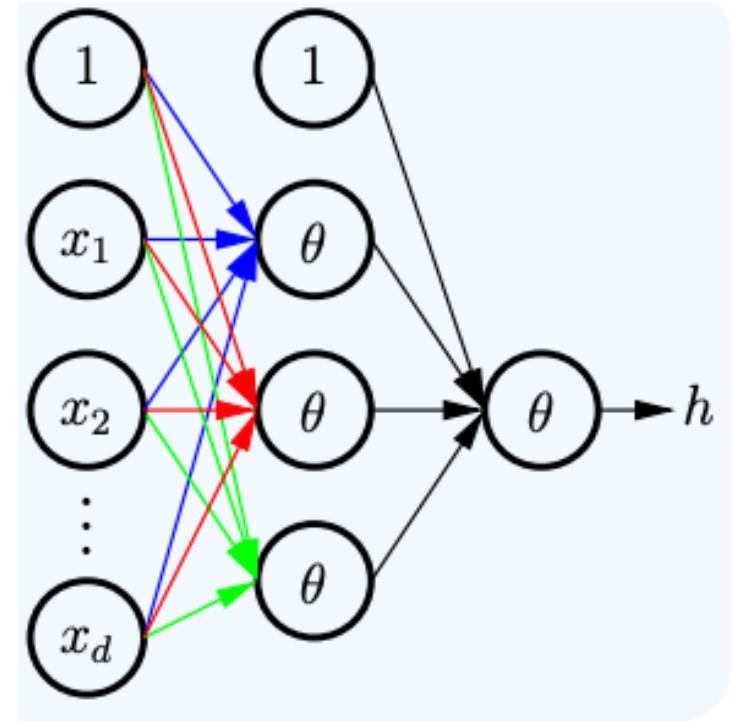
- $$h(\vec{x}) = \theta(w_0 + \sum w_i \phi_i(\vec{x}))$$

- How do we write a Kernel SVM hypothesis

- $$g(\vec{x}) = \theta \left( b^* + \sum_{\alpha_n^* > 0} \alpha_n^* y_n K(\vec{x}_n, \vec{x}) \right)$$

- Interpretation:

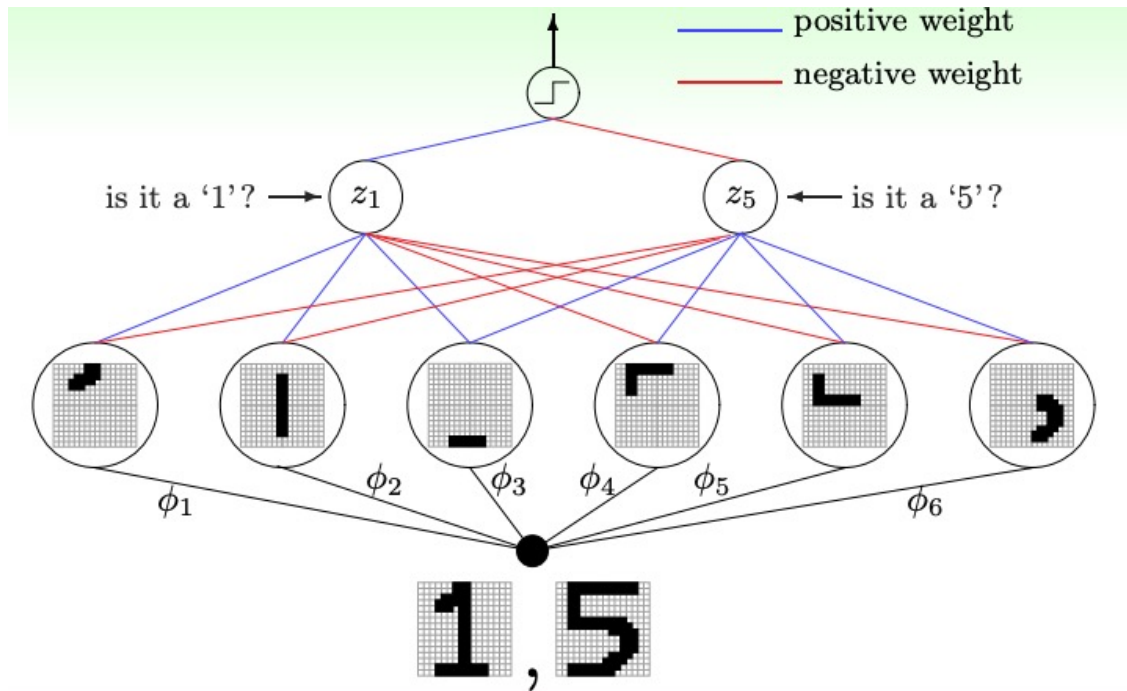
- The hidden layer is like **feature transform**
  - Shallow learning vs. deep learning





# Deep Neural Network

- “Shallow” neural network is powerful (universal approximation theorem holds with a single hidden layer). Why “deep” neural networks?



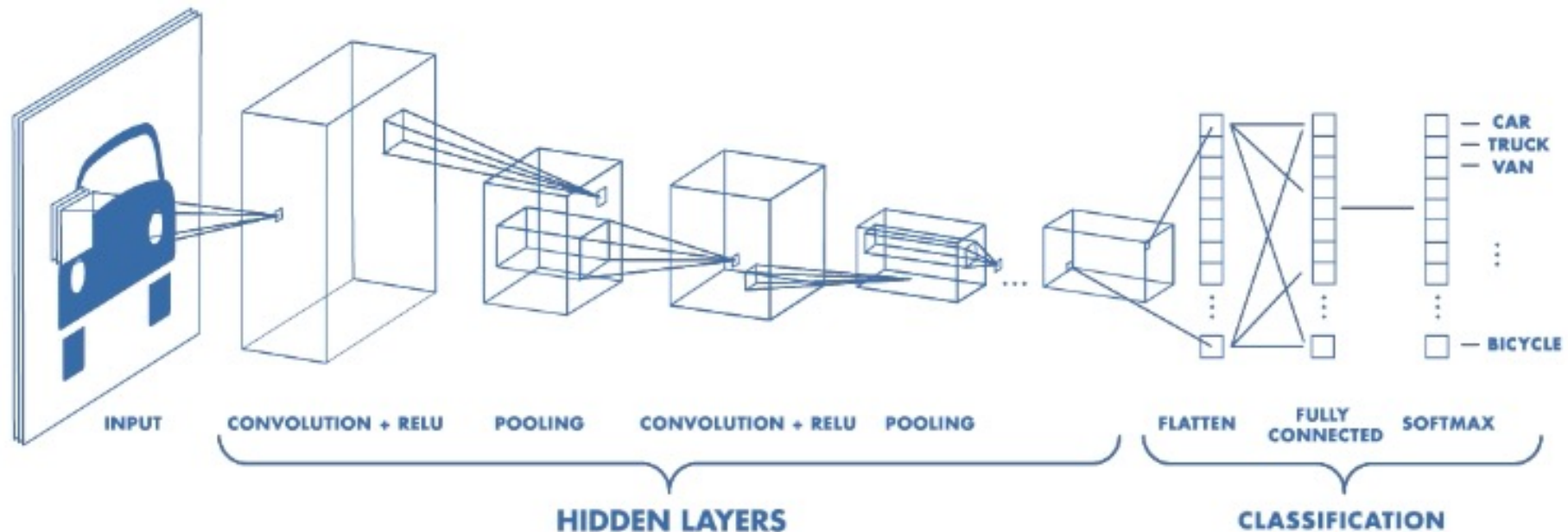
Each layer captures **features** of the previous layers.

We can use “raw data” (e.g., pixels of an image) as input. The hidden layer are extracting the **features**.

Design different **network architectures** to incorporate domain knowledge.

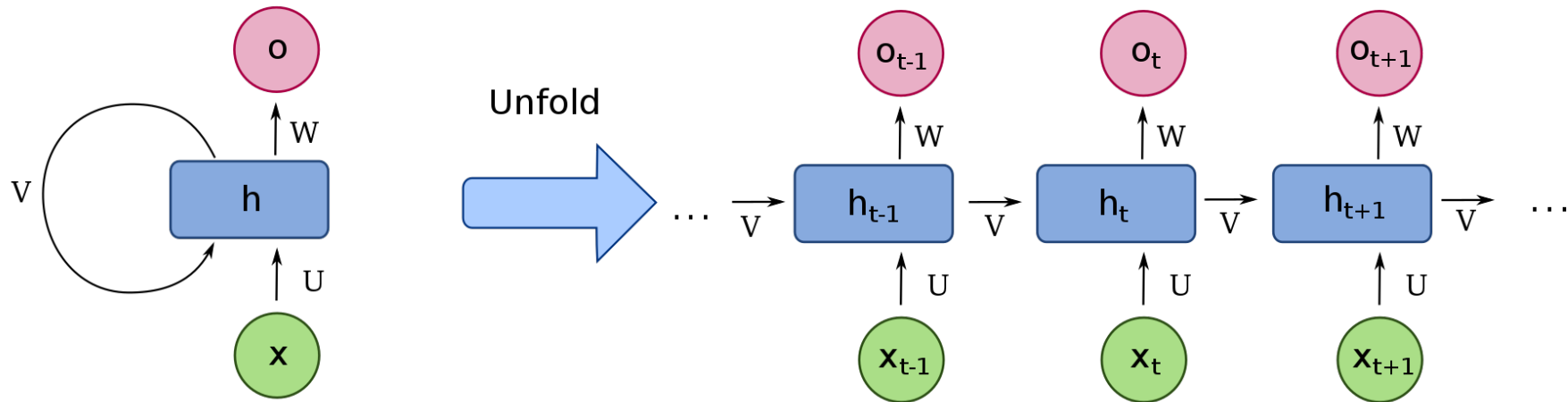
# Example Network Structure [\[Safe to Skip for the Exam\]](#)

- Convolutional Neural Networks (CNN)
  - Captures the localized properties of features
    - Particularly suitable for computer vision (images)
    - Go (AlphaGo) is another famous application of CNN



# Example Network Structure [\[Safe to Skip for the Exam\]](#)

- Recurrent Neural Network (RNN)
  - Aim to deal with time-series data, such as natural language processing
  - Using hidden layers to store temporal information
  - Allow previous outputs to be used as inputs and keep hidden states

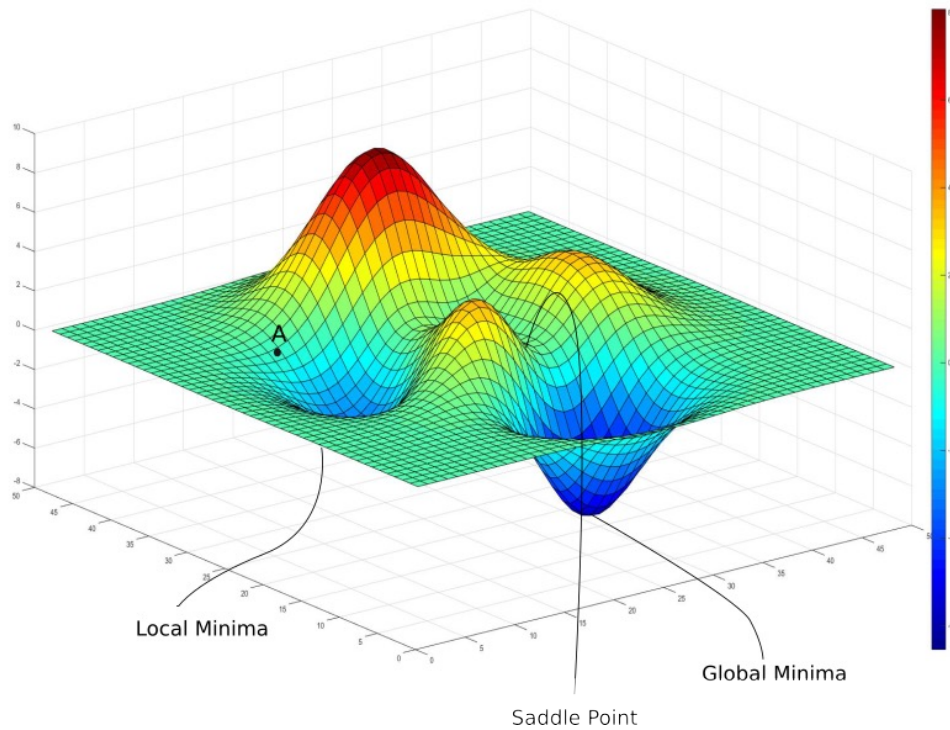


# Some Techniques in Improving Deep Learning

- Regularization to mitigate overfitting
  - Weight-based, early stopping, dropout, etc
- Incorporating domain knowledges
  - Network architectures (e.g., Convolutional Neural Nets)
- Improving computation with huge amount of data
  - Hardware architecture to improve parallel computation
- Improving gradient-based optimization
  - Choosing better **initialization** points

Initialization

# Error is Nonconvex in Neural Networks



- We mostly adopt gradient-descent-style algorithms for optimization.
- No guarantee to converge to global optimal.
- Need to run it many times.
- Initialization matters!