CSE 417T
Introduction to Machine Learning

Lecture 22

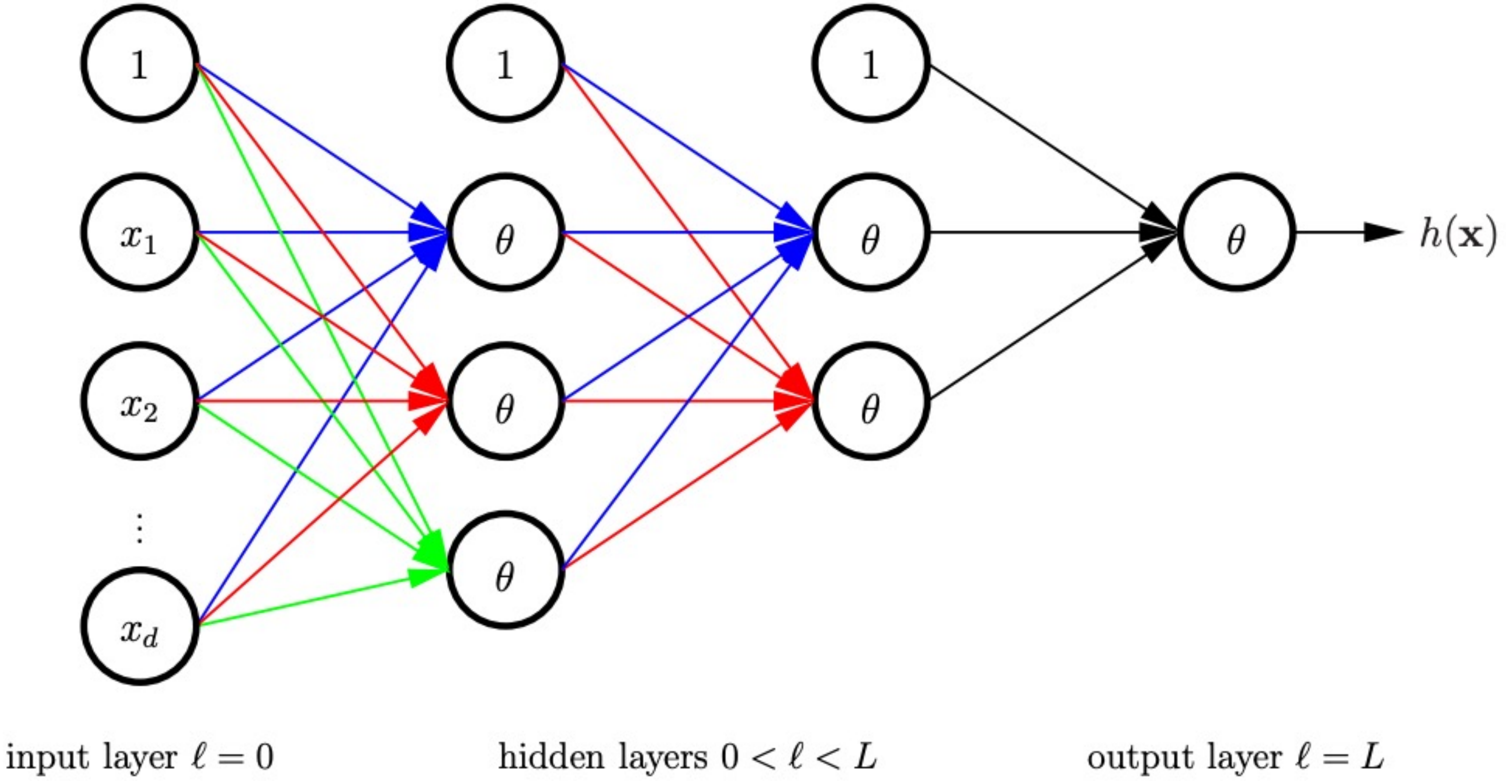
Instructor: Chien-Ju (CJ) Ho

- Homework 5: due **April 30** (Friday)

- Exam 2: (**May 4**, Tuesday)
  - Duration: 75+5 Minutes
  - Content: Focus on the content of 2nd half of the semester
    - Though knowledge is cumulative
  - Time: by default, everyone is expected to take it during lecture time
    - Please let me know by this Friday if you can't (I'll post a google form for this on Piazza)
  - Review lecture: Apr 29
    - Practice questions will be posted next week
  - Other logistics are the same as Exam 1
    - Format: Gradescope online exam + Zoom (with camera on)
    - Information access during exam:
      - Allowed: Textbook, slides, hardcopy materials (e.g., your own notes)
      - Not allowed: search for information online during exam, talk to any other persons
    - **Follow Piazza announcements** for updates/information

Recap

Neural Networks



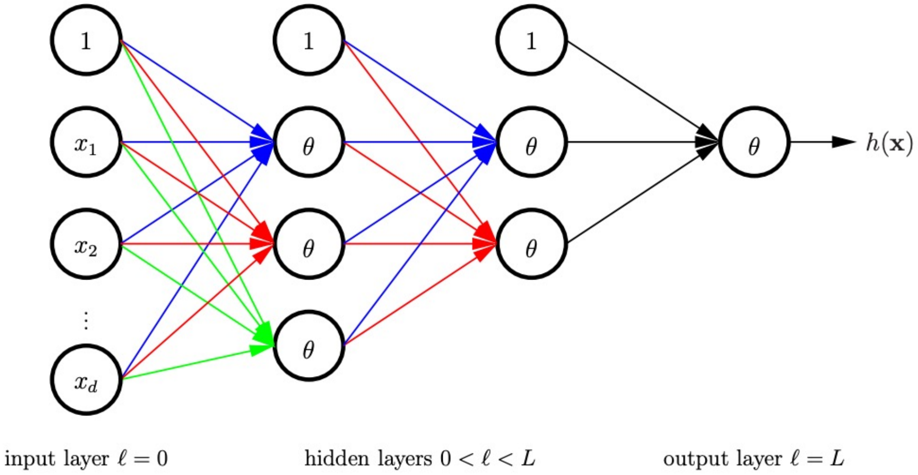
θ: activation function

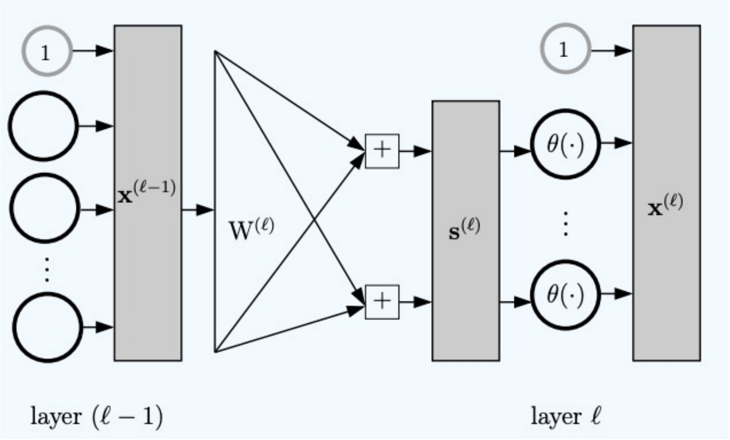
(Specify the "activation" of the neuron)

We mostly focus on feed-forward network structure

Notations of Neural Networks (NN)

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

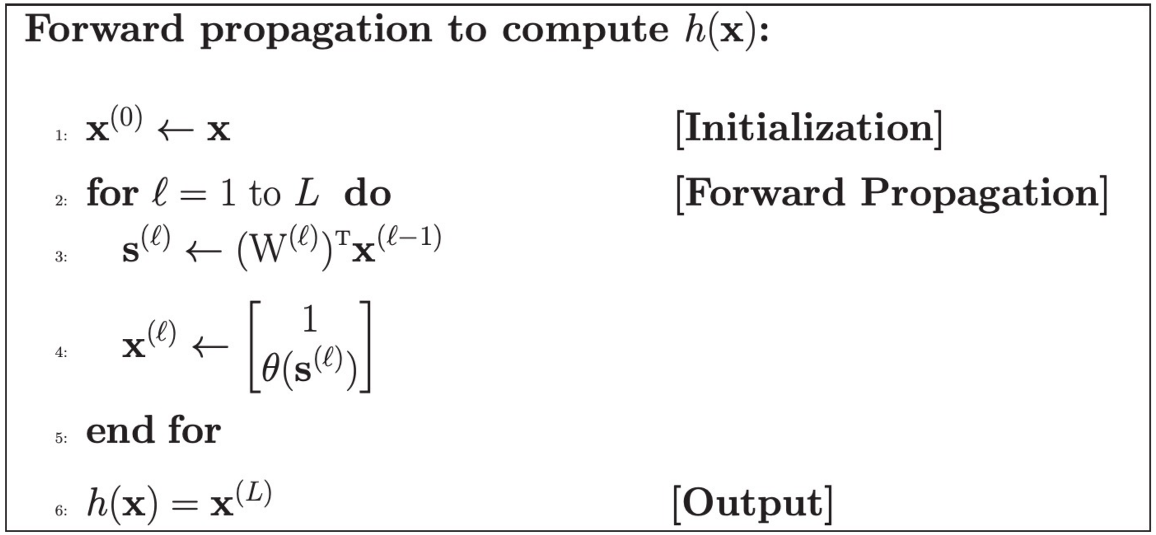




Forward Propagation (evaluate $h(\vec{x})$)

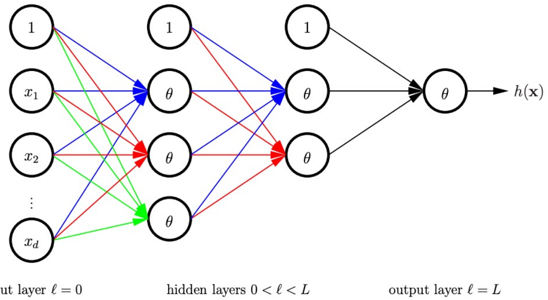
- A NN hypothesis h is characterized by $\left\{w_{i,j}^{(\ell)}\right\}$
- How to evaluate $h(\vec{x})$?

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



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

```
$\mathbf{x}^{(0)} \leftarrow \mathbf{x}$                                        [Initialization]
for $\ell = 1$ to $L$ do                                   [Forward Propagation]
    $\mathbf{s}^{(\ell)} \leftarrow (\mathrm{W}^{(\ell)})^{\mathrm{T}}\mathbf{x}^{(\ell-1)}$
    $\mathbf{x}^{(\ell)} \leftarrow \begin{bmatrix} 1 \\ \theta(\mathbf{s}^{(\ell)}) \end{bmatrix}$
end for
$h(\mathbf{x}) = \mathbf{x}^{(L)}$                                     [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.

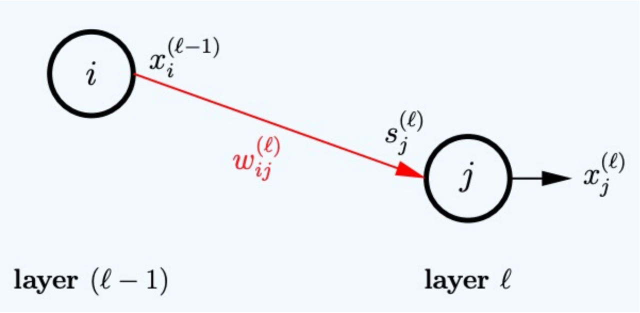
How to Learn NN From Data?

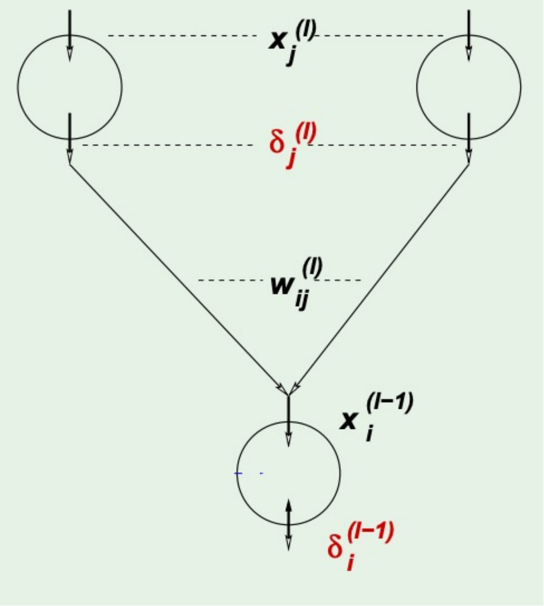
- Given D, how to learn the weights $W = \left\{w_{i,j}^{(\ell)}\right\}$?

- 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.
  - Three-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

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

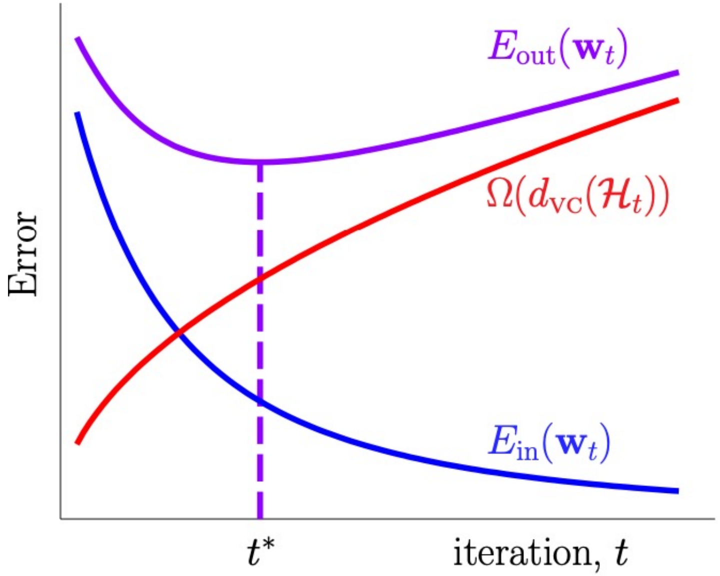
- Weight elimination

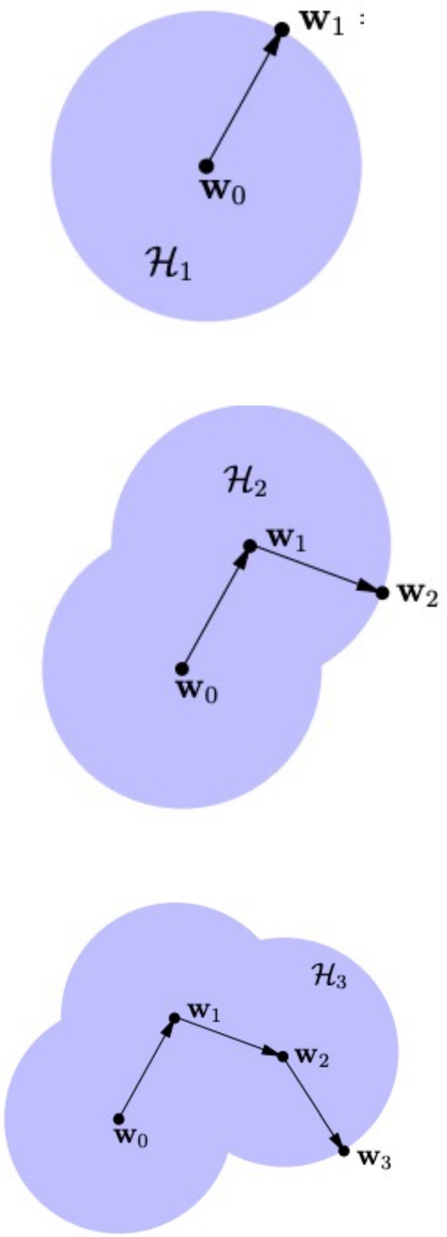
$$E_{aug}(W) = E_{in}(W) + \frac{\lambda}{N}\sum_{i,j,\ell}\frac{\left(w_{i,j}^{(\ell)}\right)^2}{1+\left(w_{i,j}^{(\ell)}\right)^2}$$

  - 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 $t = 2$
  - …
  - $H_1 \subseteq H_2 \subseteq H_3 \subseteq \cdots$

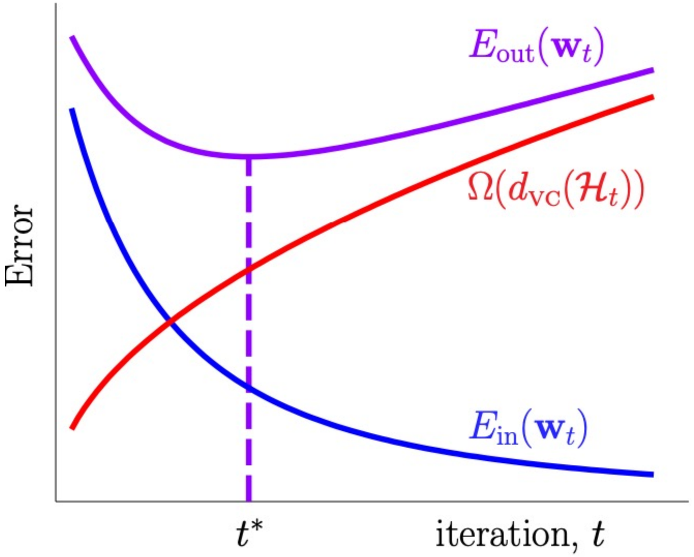


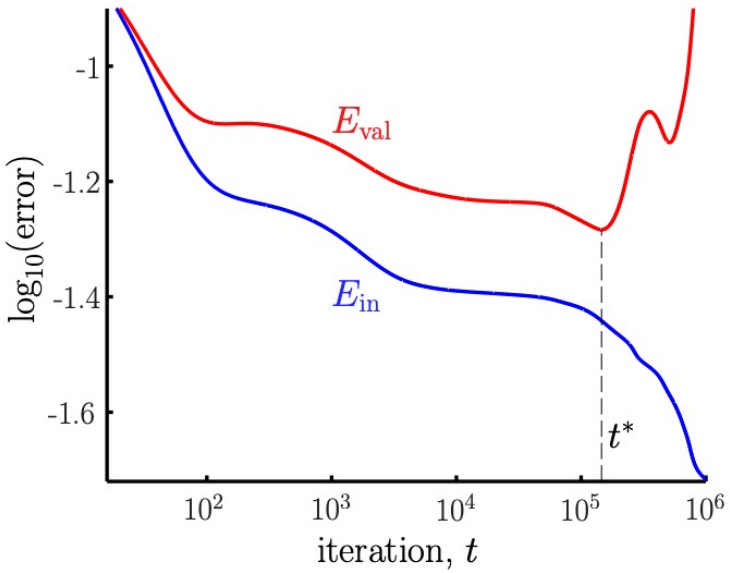


Early Stopping

- Stopping gradient descent early is a regularization method
  - Constrain the hypothesis set

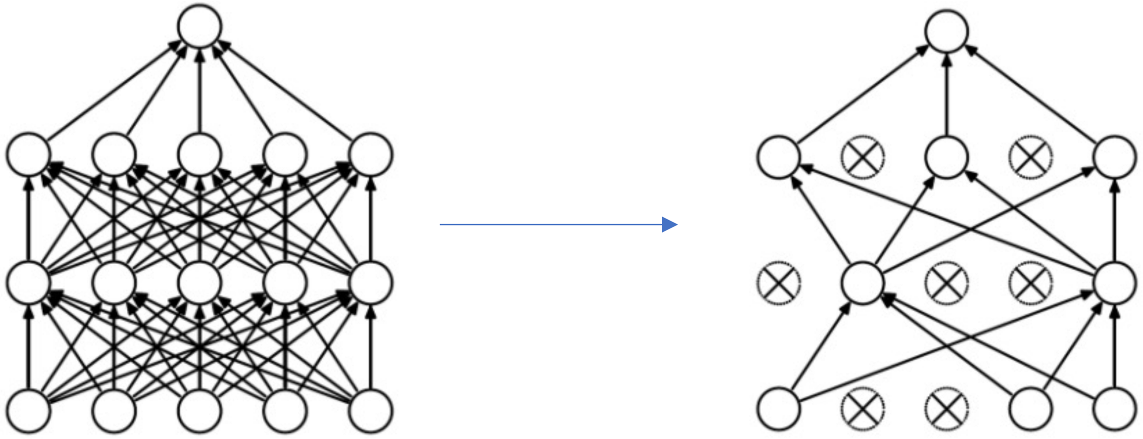
- How to find the optimal stopping point t^*?
  - Using validation is a common approach





Dropout

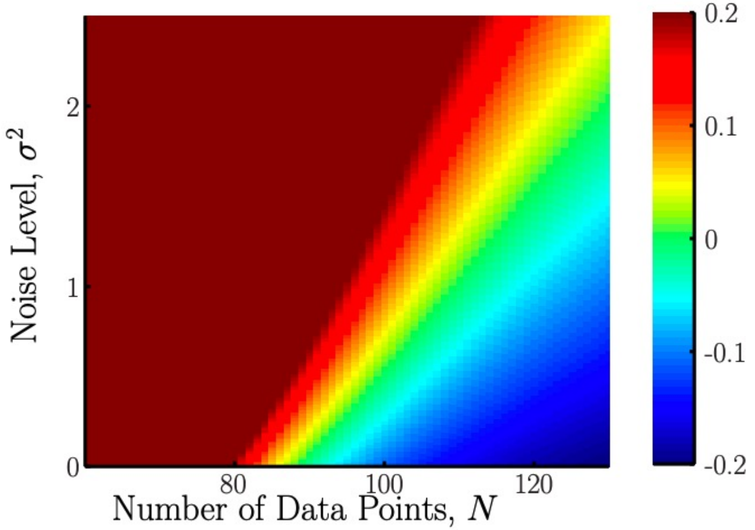
- Neural networks is very expressive (low bias, potentially high variance)
- Dropout
  - Randomly drop p portion of the weights during training

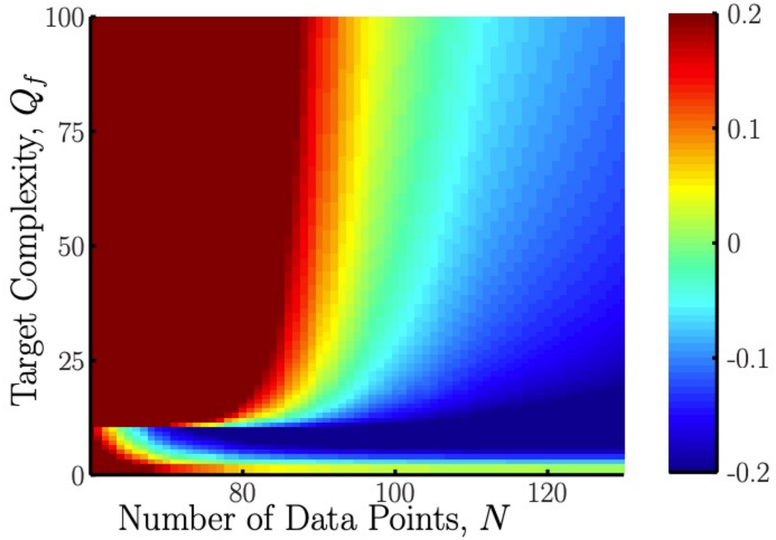


  - Learn many models with dropout
  - Average them during prediction (reduce weights by a ratio of p)

A Nontraditional Method to Avoid Overfitting

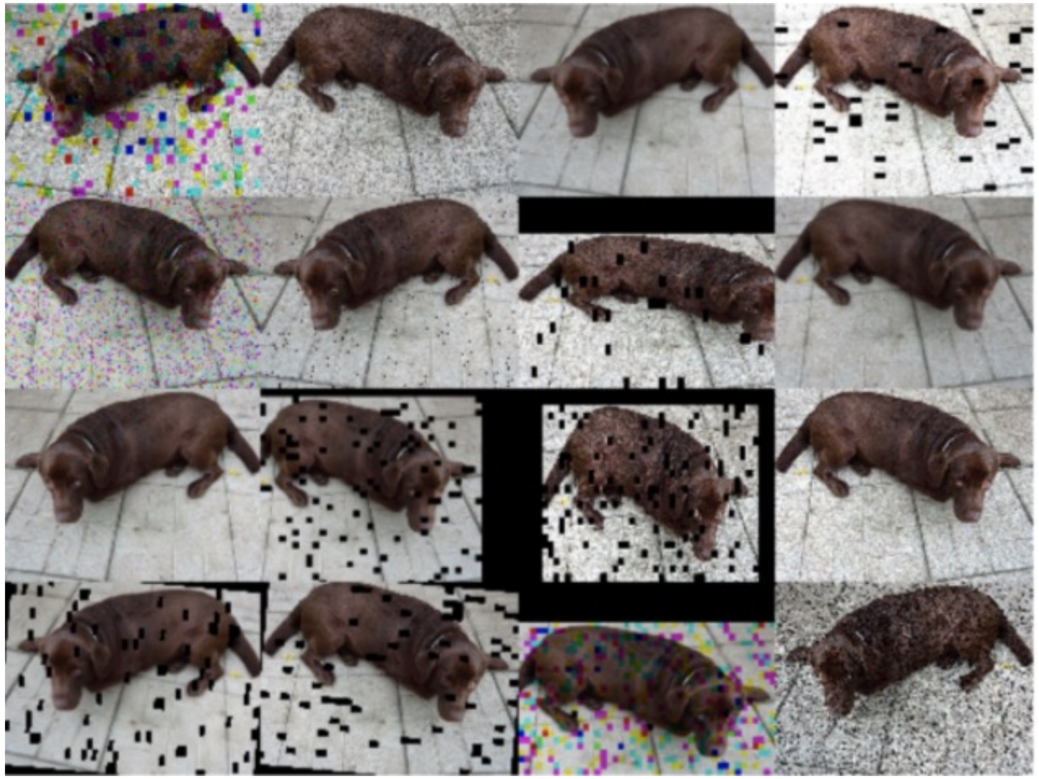
- What's the cause of overfitting?





- 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

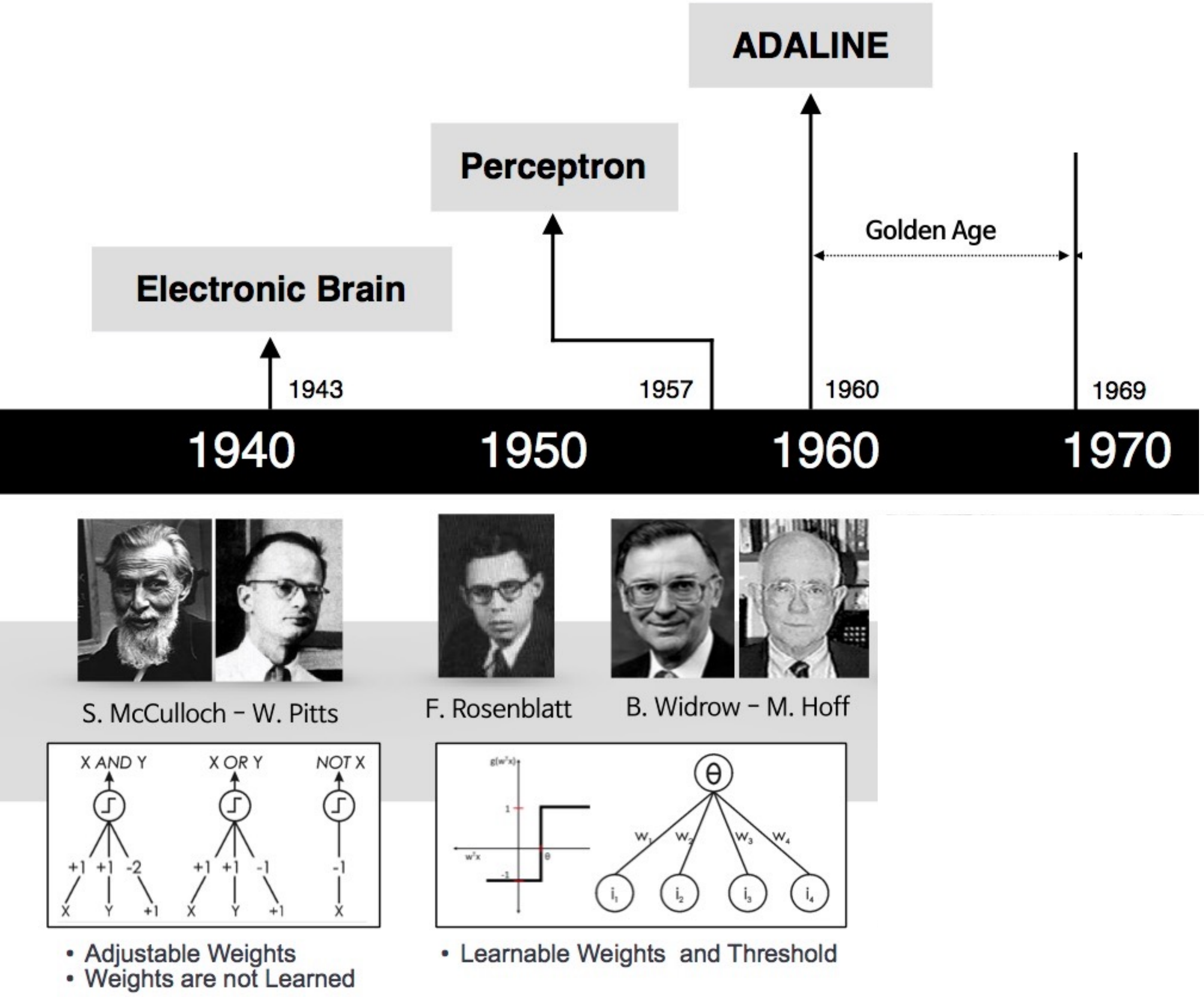


Short Break and Q&A

Deep Learning

Brief/Informal History

Rosenblatt: “[The perceptron is] the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”



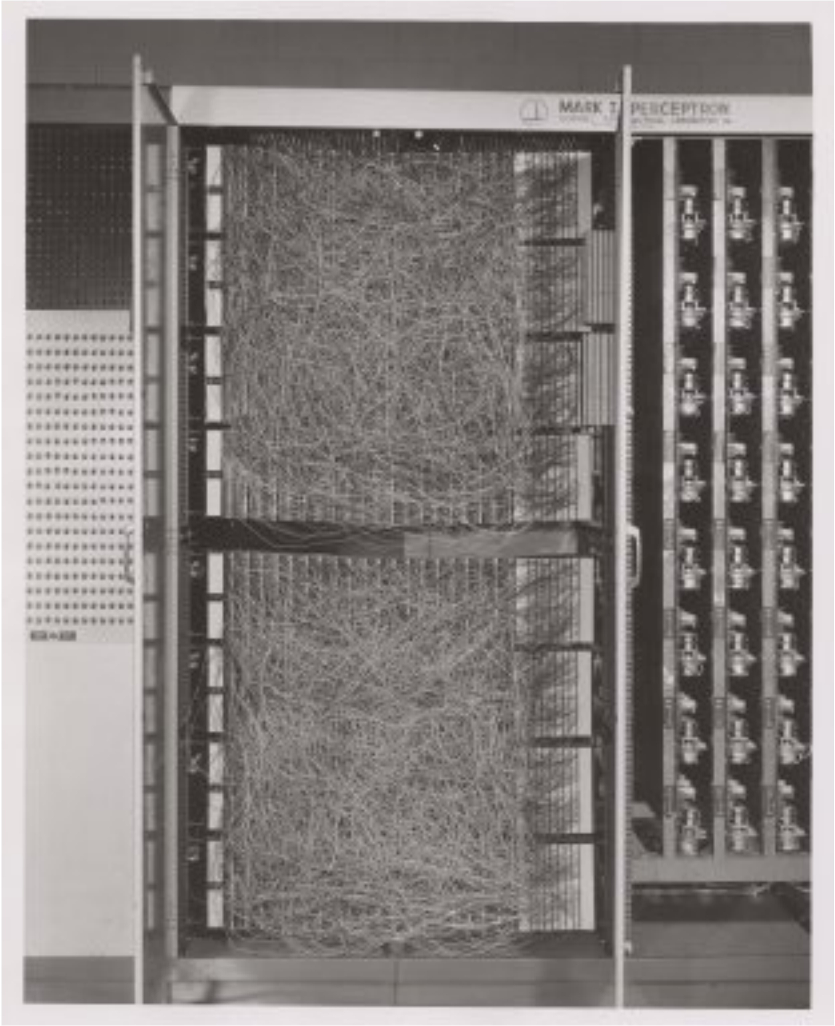


Image source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

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.”

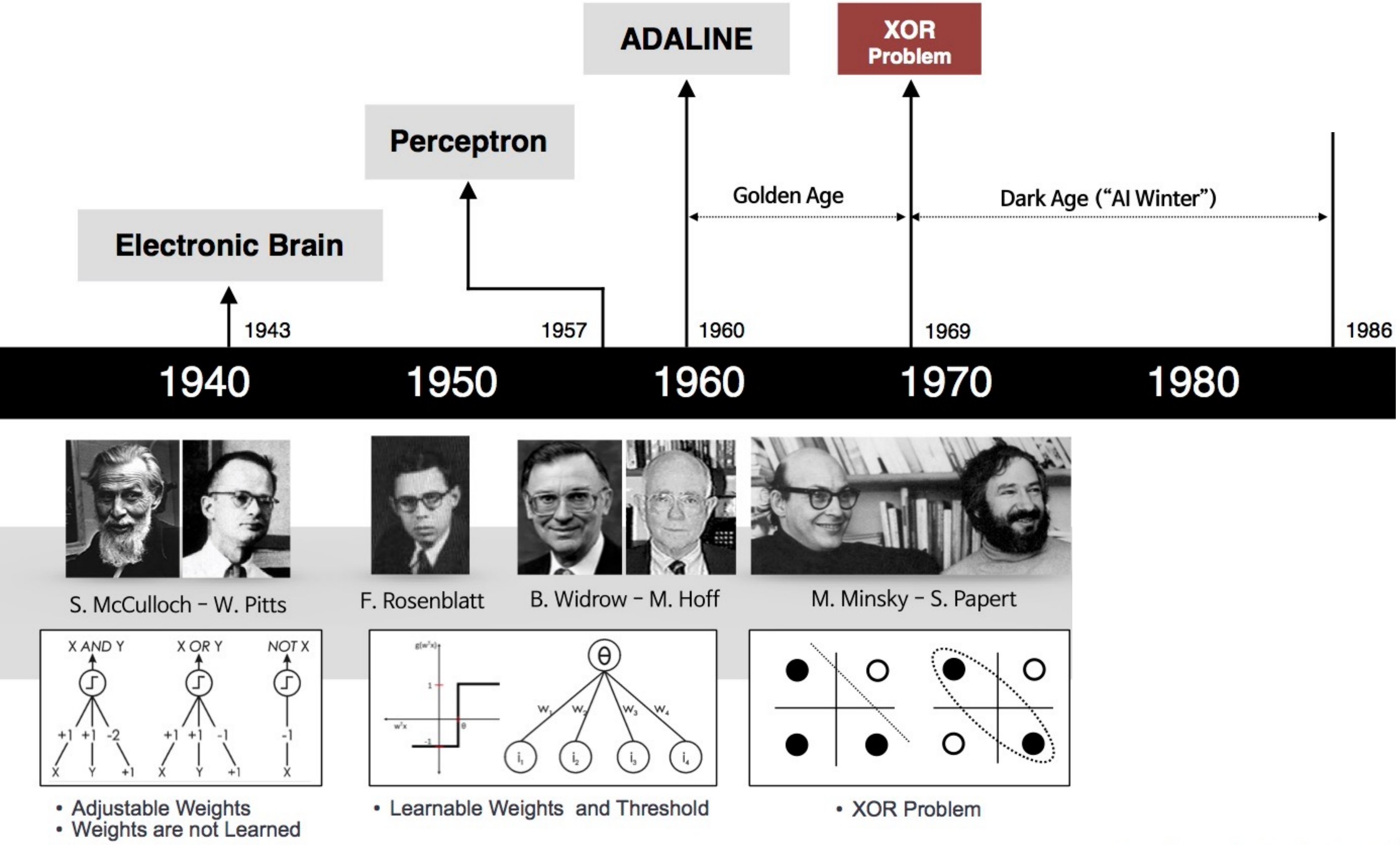
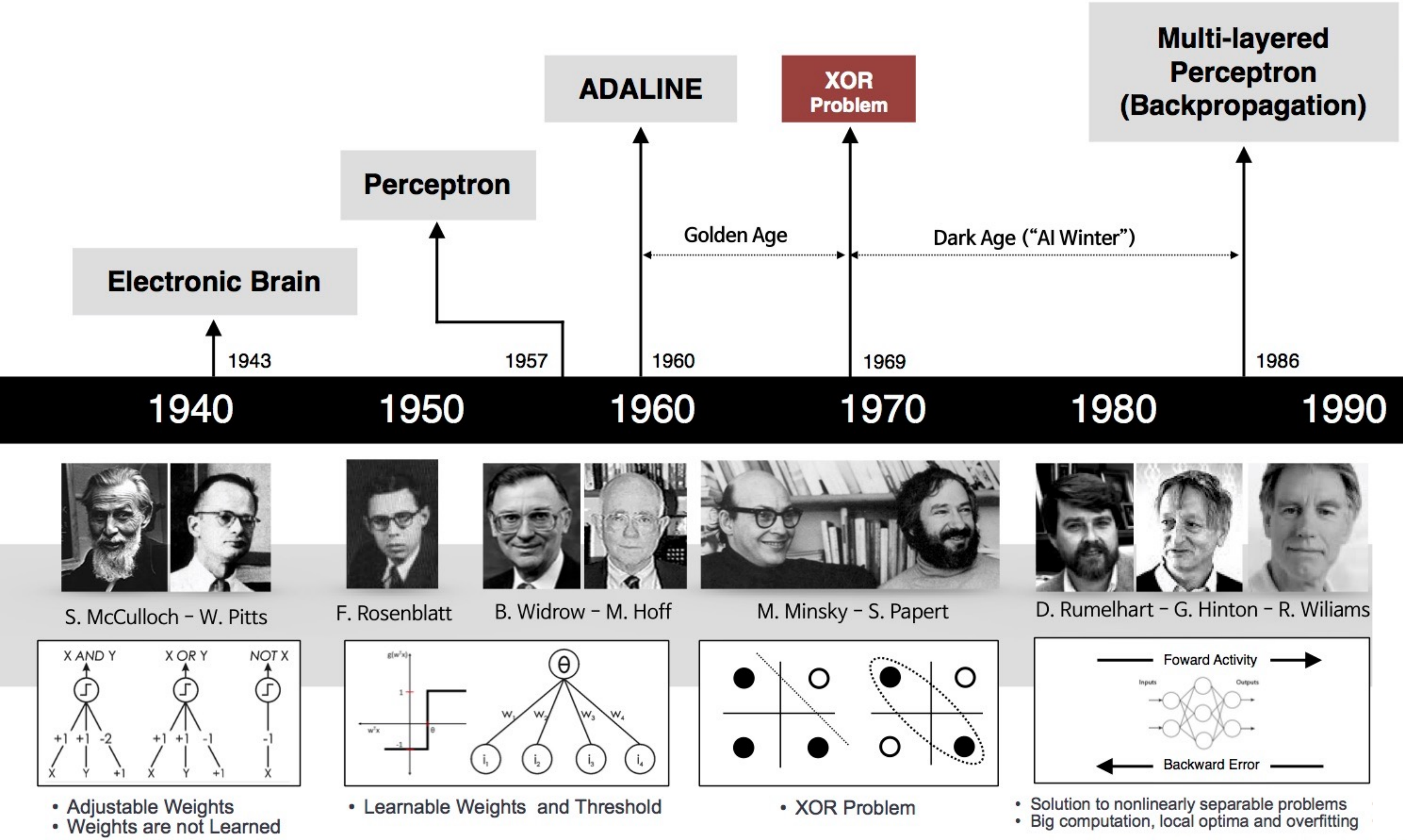


Image source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html



Electronic Brain

Perceptron

ADALINE

XOR Problem

Multi-layered Perceptron (Backpropagation)

Golden Age

Dark Age (“AI Winter”)

1943

1957

1960

1969

1986

1940

1950

1960

1970

1980

1990

S. McCulloch – W. Pitts

F. Rosenblatt

B. Widrow – M. Hoff

M. Minsky – S. Papert

D. Rumelhart – G. Hinton – R. Wiliams

- Adjustable Weights
- Weights are not Learned

- Learnable Weights and Threshold

- XOR Problem

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

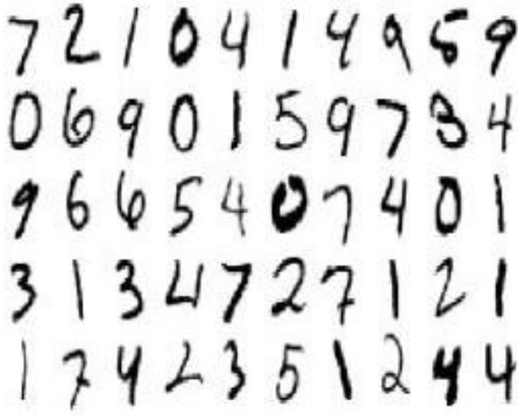
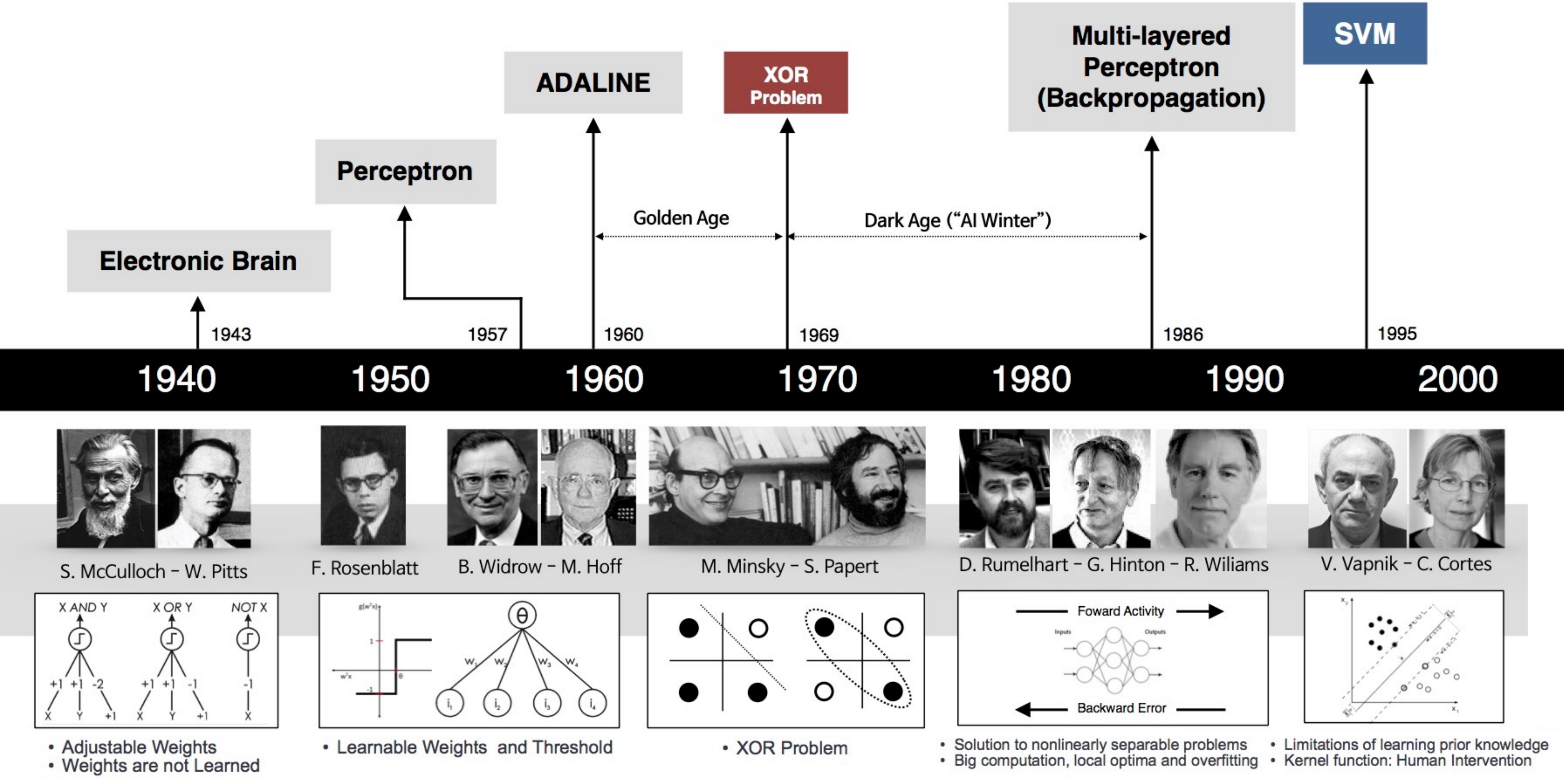


Image source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html



Electronic Brain

Perceptron

ADALINE

XOR Problem

Multi-layered Perceptron (Backpropagation)

SVM

Golden Age

Dark Age (“AI Winter”)

1943 1957 1960 1969 1986 1995

1940 1950 1960 1970 1980 1990 2000

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

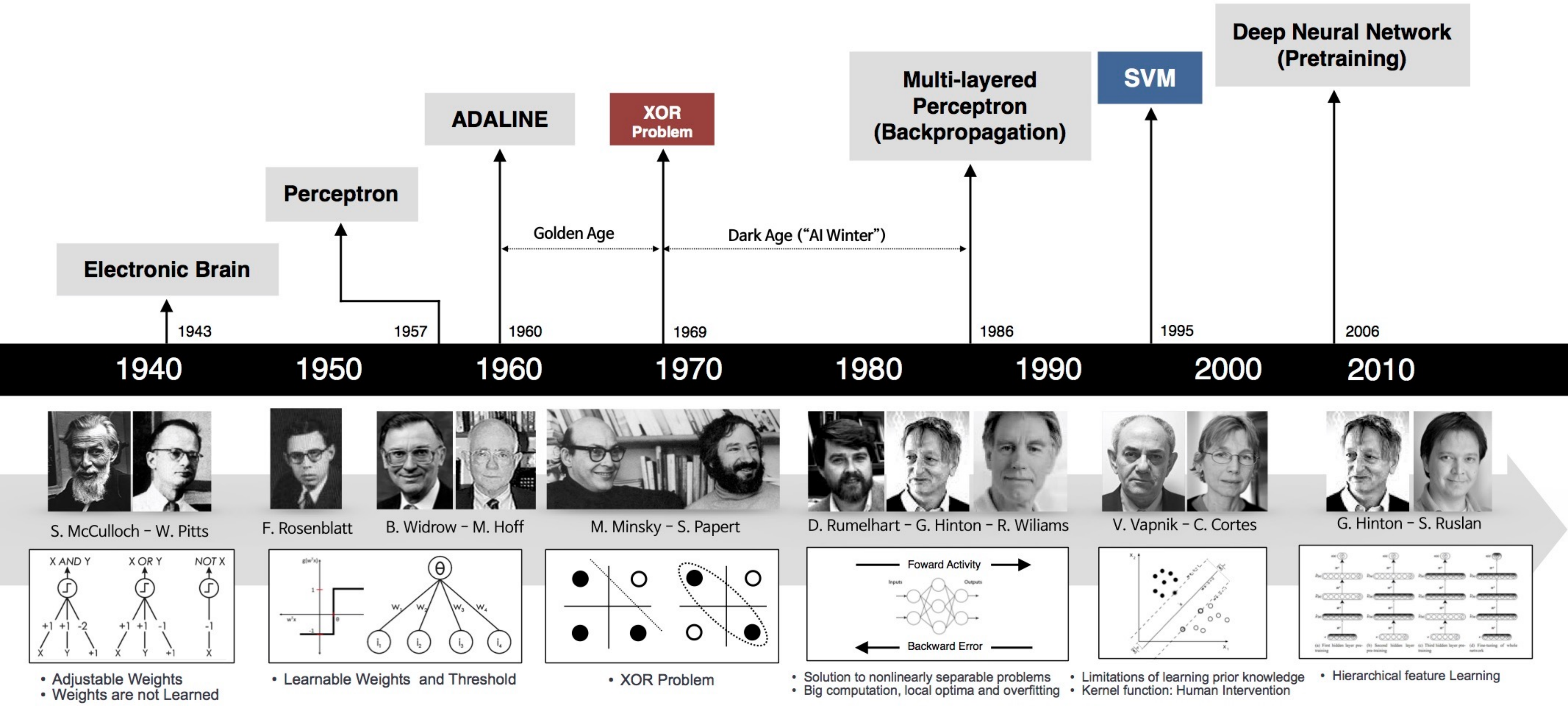
D. Rumelhart – G. Hinton – R. Wiliams

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

V. Vapnik – C. Cortes

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

Image source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html



Electronic Brain — 1943

Perceptron — 1957

ADALINE — 1960

XOR Problem — 1969

Golden Age

Dark Age (“AI Winter”)

Multi-layered Perceptron (Backpropagation) — 1986

SVM — 1995

Deep Neural Network (Pretraining) — 2006

1940 1950 1960 1970 1980 1990 2000 2010

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. Wiliams
- 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

Image source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

ImageNet Challenge 2012

Task 1: Classification

Car

- Predict a class label

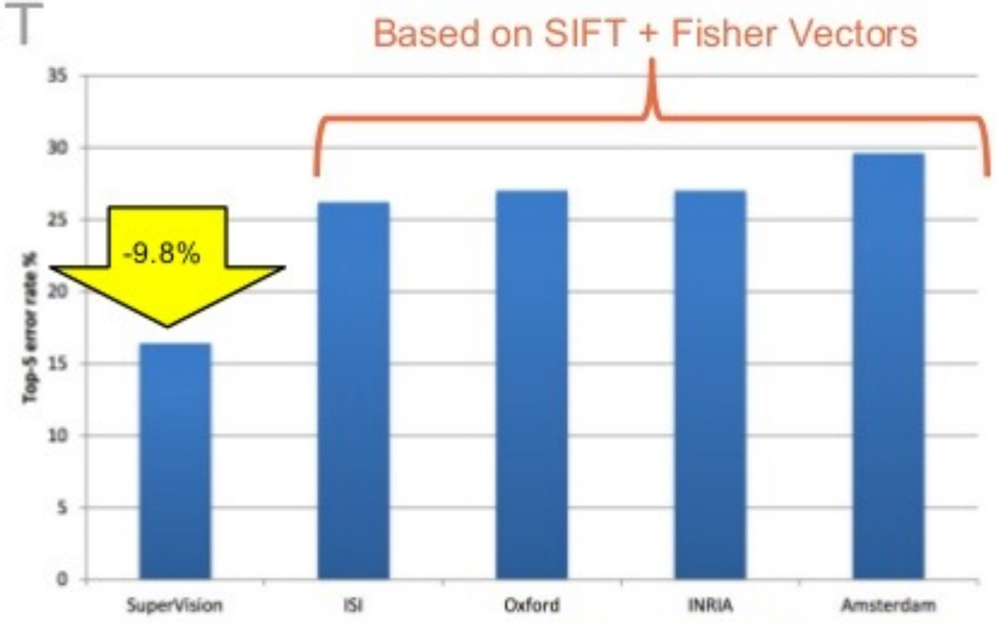
- 5 predictions / image
- 1000 classes
- 1,200 images per class for training
- Bounding boxes for 50% of training.

ImageNet Challenge

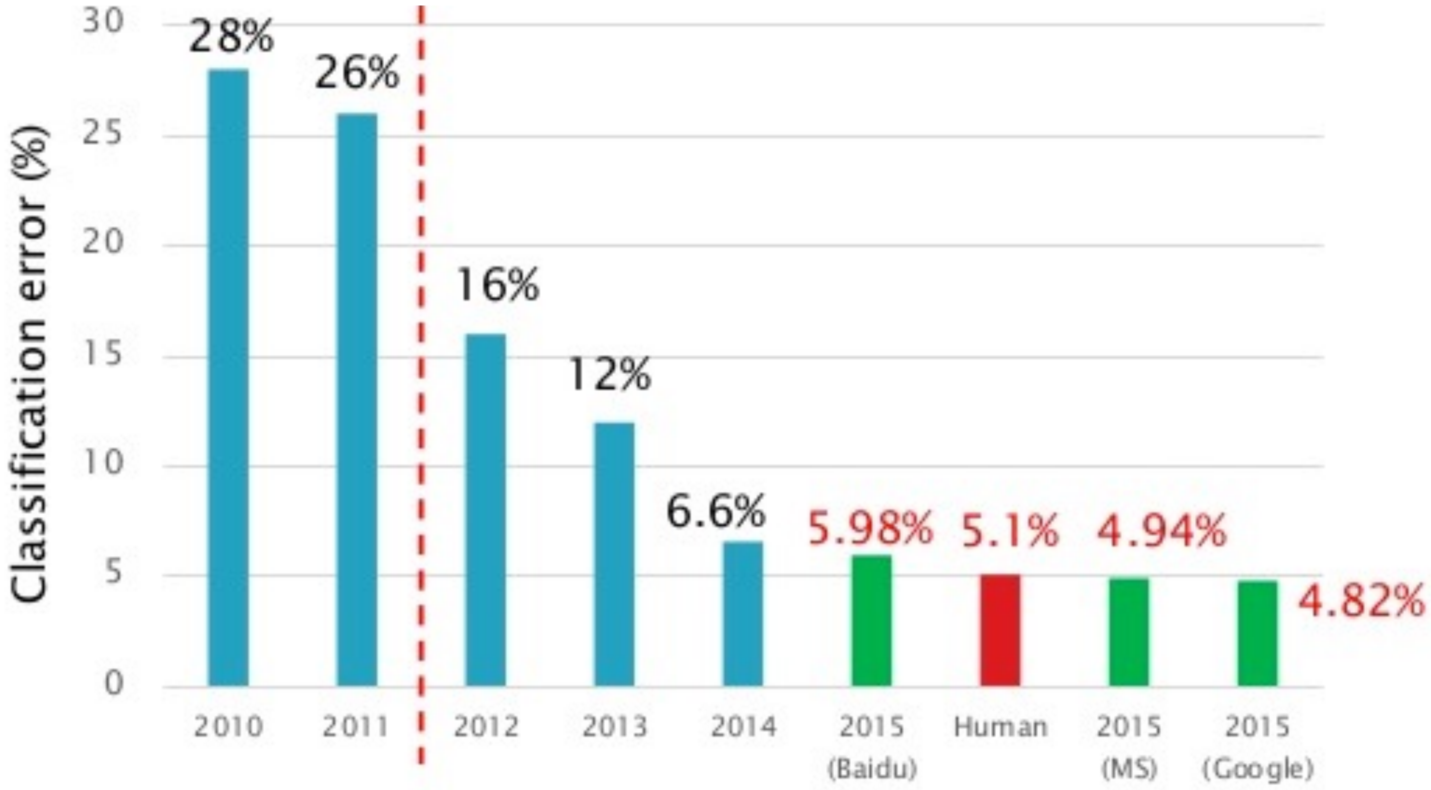
Image Classification 2012

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.* [web]



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

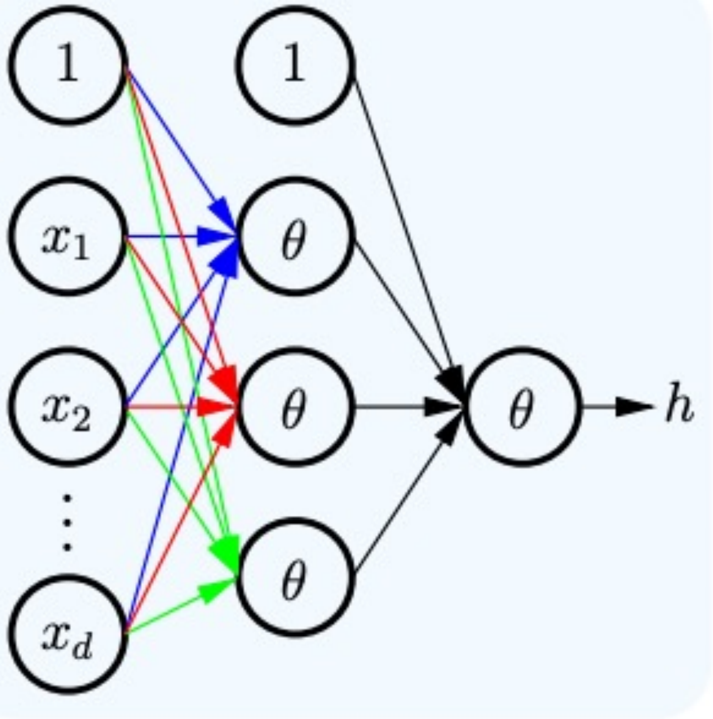
Ioffe et al., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv, 2015.

What is “Deep” Learning

Neural networks with many layers

Single Hidden-Layer Neural Network

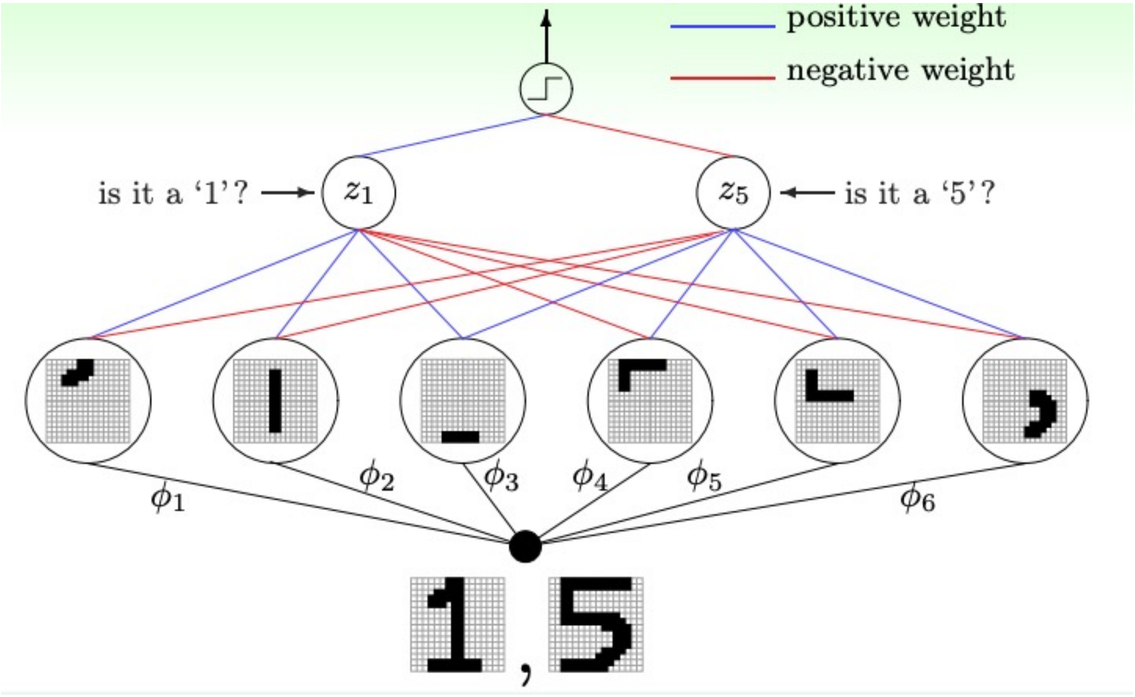
- How do we write a hypothesis in single-hidden layer NN mathematically?
  - $h(\vec{x}) = \theta\left(w_{0,1}^{(2)} + \sum_{j=1}^{d^{(1)}} w_{j,1}^{(2)} x_j^{(1)}\right)$
    $= \theta\left(w_{0,1}^{(2)} + \sum_{j=1}^{d^{(1)}} w_{j,1}^{(2)} \theta\left(\sum_{i=0}^{d^{(0)}} w_{i,j}^{(1)} x_i\right)\right)$
- 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?



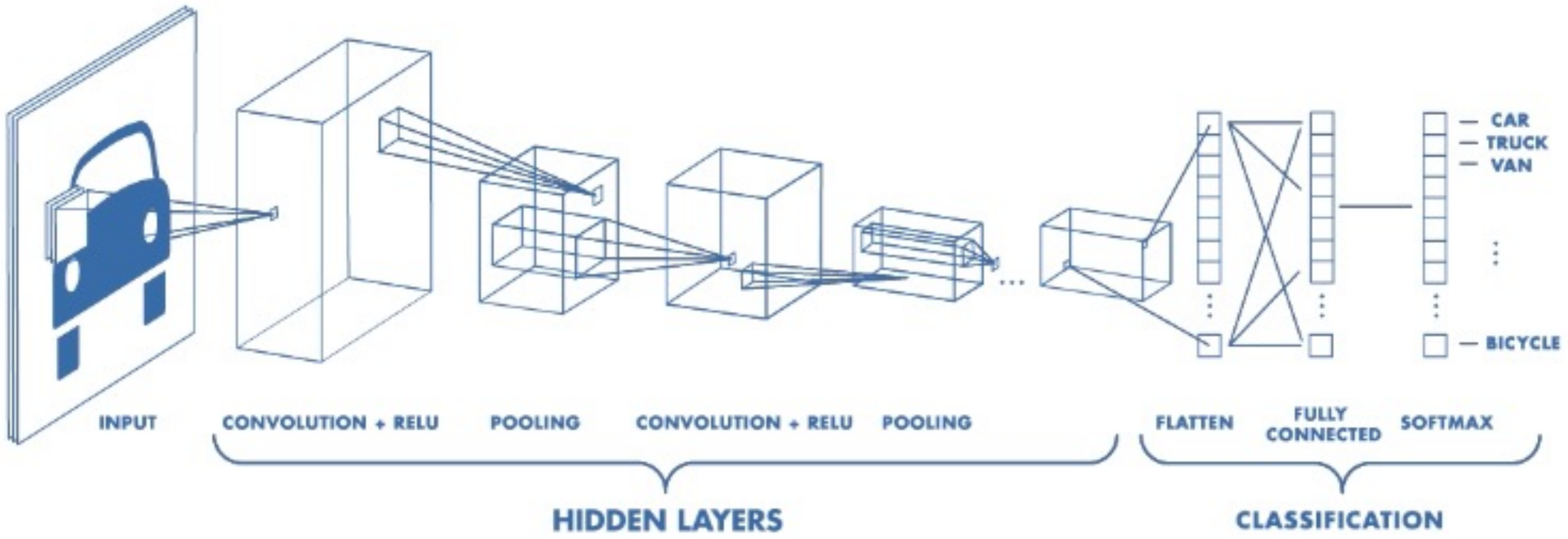
Each layer captures features of the previous layers.

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.

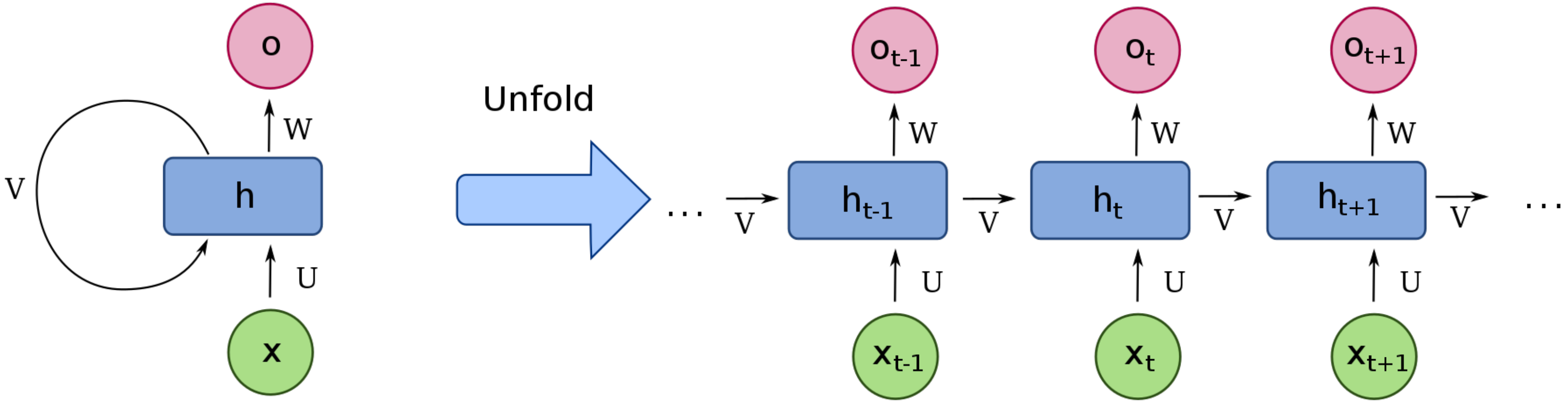
Example Network Structure [Safe to Skip for the Exam]

- 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



[Images from Wikipedia]

Some Techniques in Improving Deep Learning

- Regularization to mitigate overfitting
    - Weight-based, early stopping, dropout, etc

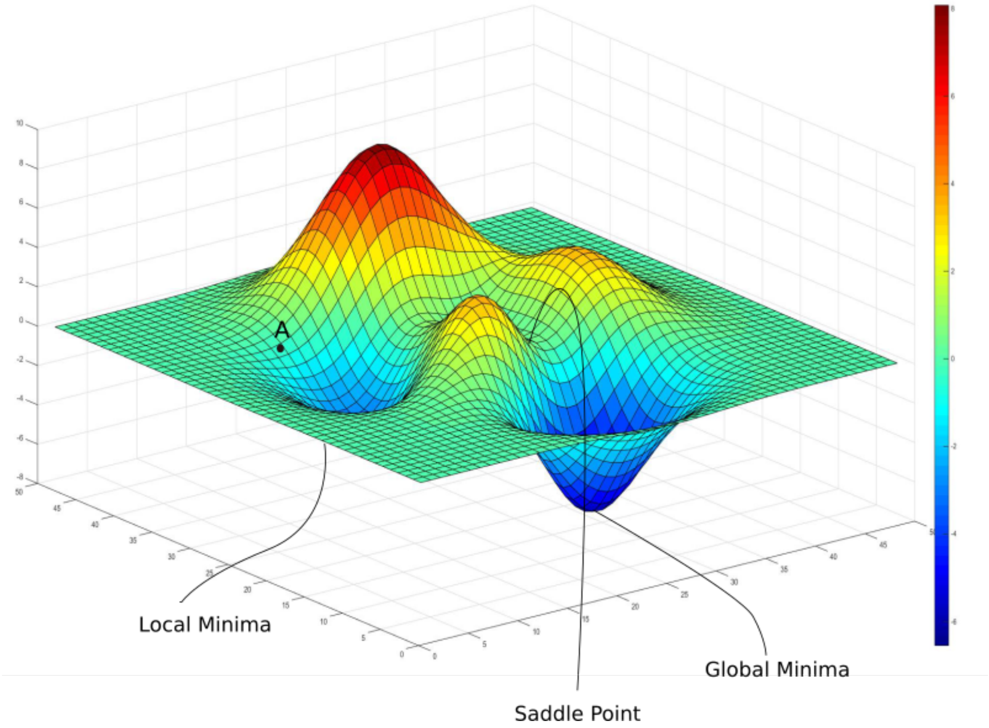
- Incorporating domain knowledges
    - Network architectures (e.g., Convolutional Neural Nets)

- 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!