

DS 4400

Machine Learning and Data Mining I Spring 2024

David Liu

Khoury College of Computer Science
Northeastern University

March 15 2024

Outline

- Ensemble learning
- Boosting
 - AdaBoost
 - Properties of boosting
 - Bagging vs Boosting
- Introduction to deep learning
 - History of deep learning
 - Perceptron and limitations

Ensemble Learning

Consider a set of classifiers h_1, \dots, h_L

Idea: construct a classifier $H(\mathbf{x})$ that combines the individual decisions of h_1, \dots, h_L

- e.g., could have the member classifiers vote, or
- e.g., could use different members for different regions of the instance space

Successful ensembles require **diversity**

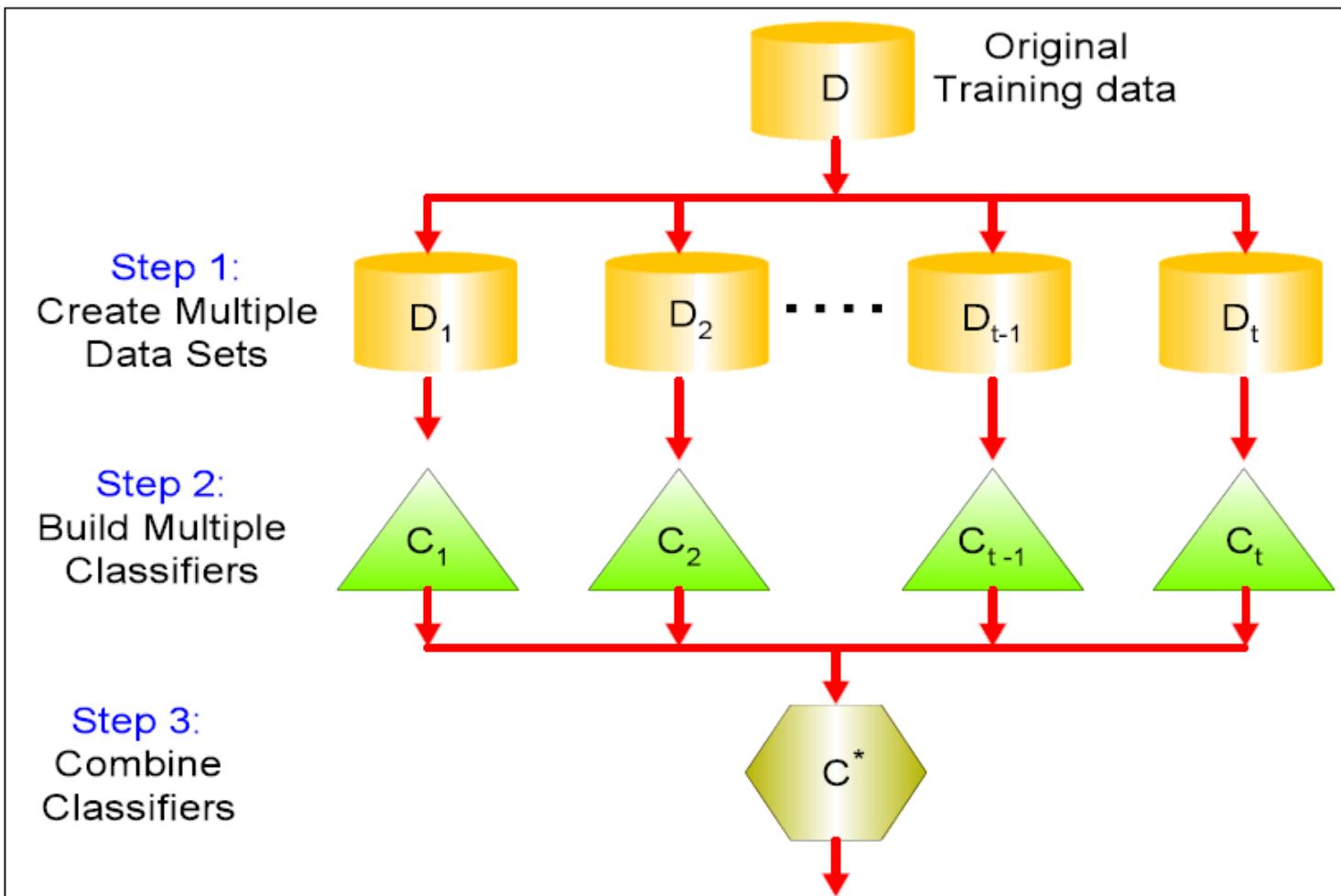
- Classifiers should make different mistakes
- Can have different types of base learners

How to Achieve Diversity

- Avoid overfitting
 - Randomize the training data
- Features are noisy
 - Randomize the set of features

Two main ensemble learning methods

Bagging



Majority Votes

Random Forests

- Ensemble method specifically designed for decision tree classifiers
- Introduce two sources of randomness: “Bagging” and “Random input vectors”
 - **Bagging method**: each tree is grown using a bootstrap sample of training data
 - **Random vector method**: **At each node**, best split is chosen from a random sample of m attributes instead of all attributes

AdaBoost

- A meta-learning algorithm with great theoretical and empirical performance
- Turns a base learner (i.e., a “weak hypothesis”) into a high performance classifier
- Creates an ensemble of weak hypotheses by repeatedly emphasizing mispredicted instances

Adaptive Boosting
Freund and Schapire 1997

Overview of AdaBoost

Sequential training process

- Mis-classified examples get higher weights
- Correct examples get lower weights

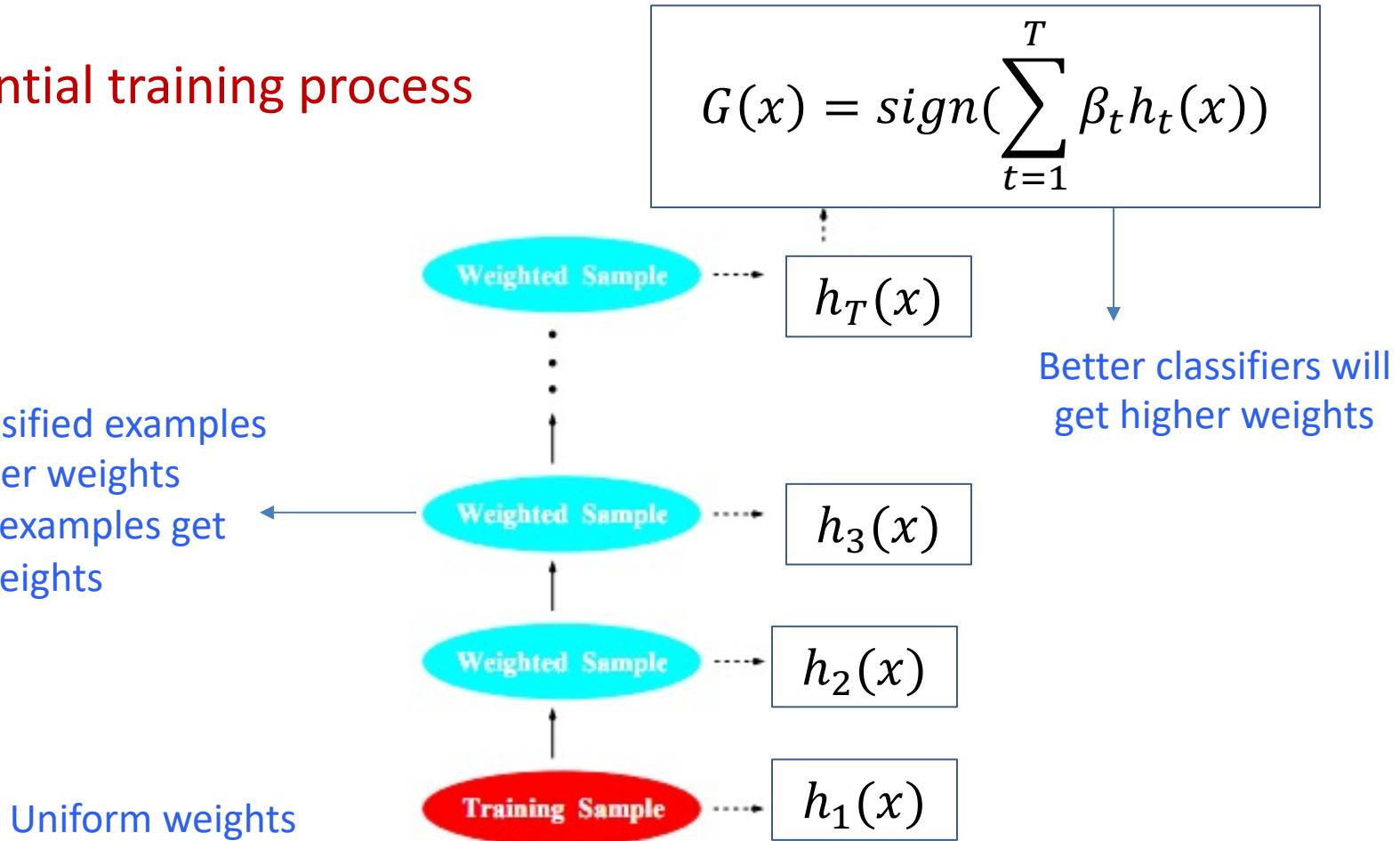
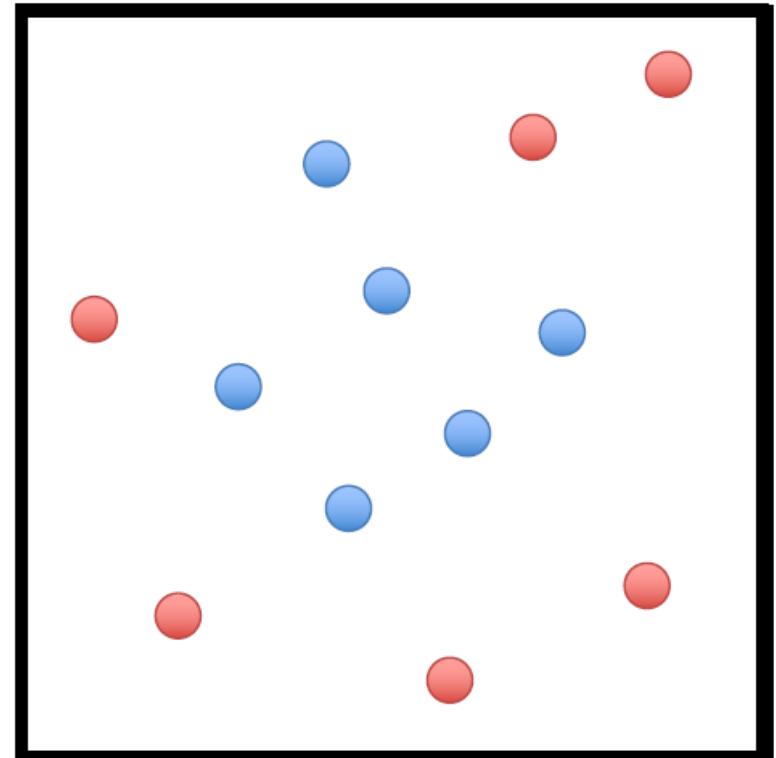


FIGURE 10.1. Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.

AdaBoost

```
1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



- Size of point represents the instance's weight

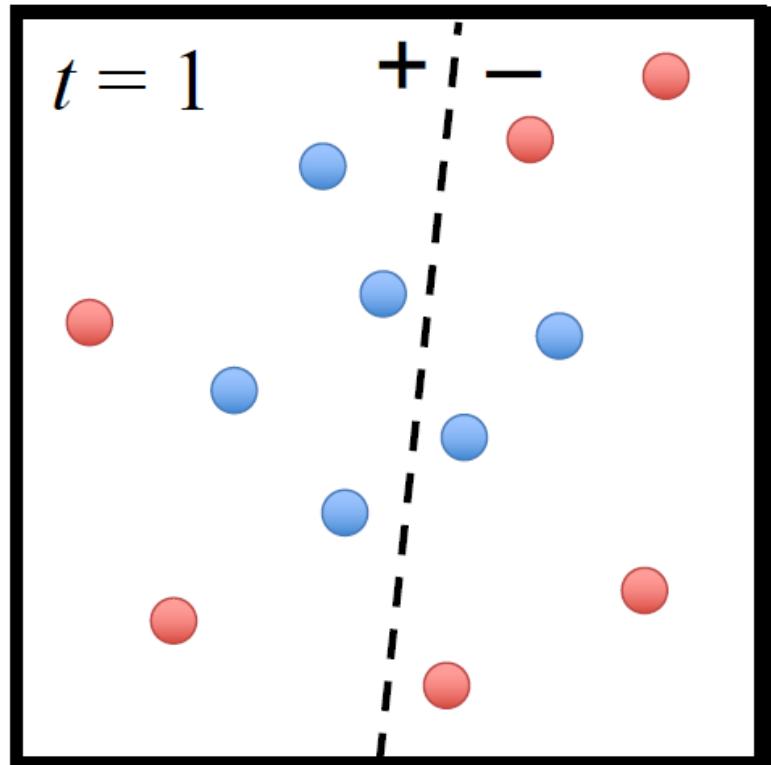
AdaBoost

```
1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:  


$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$$

7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
```

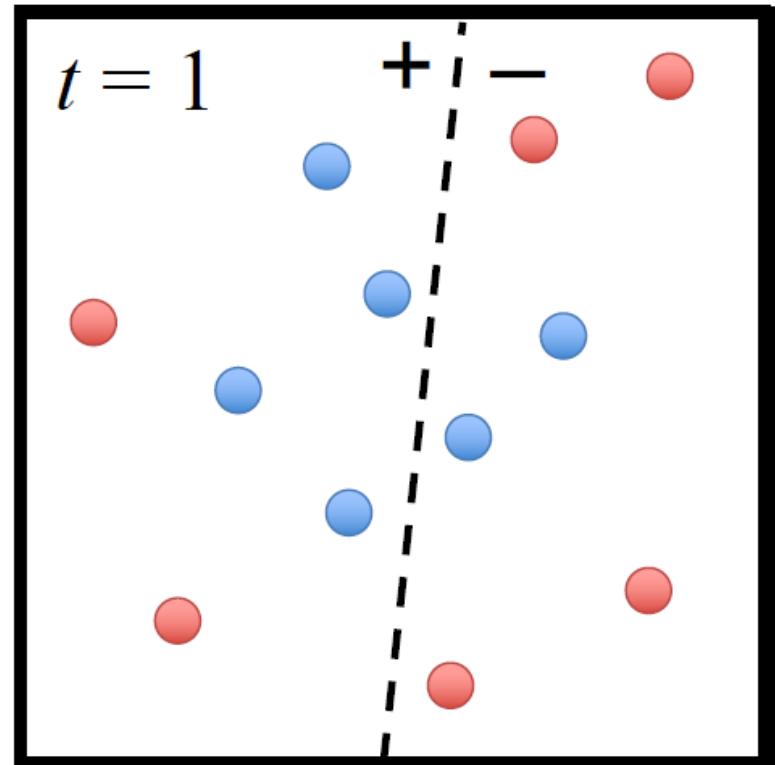
$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



AdaBoost

```
1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp (-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

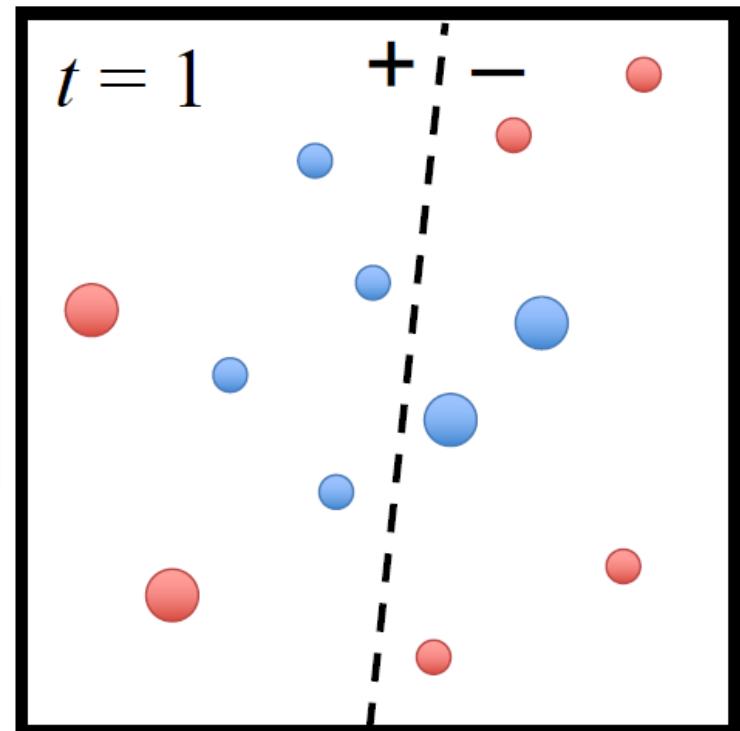


- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (can trivially guarantee)

AdaBoost

```
1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp (-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
```

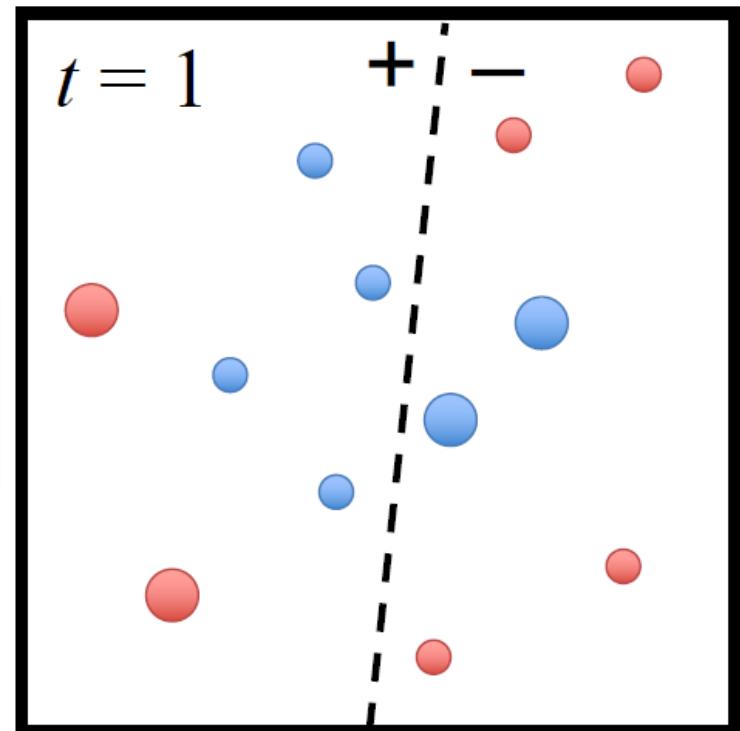
$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



AdaBoost

```
1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp (-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
```

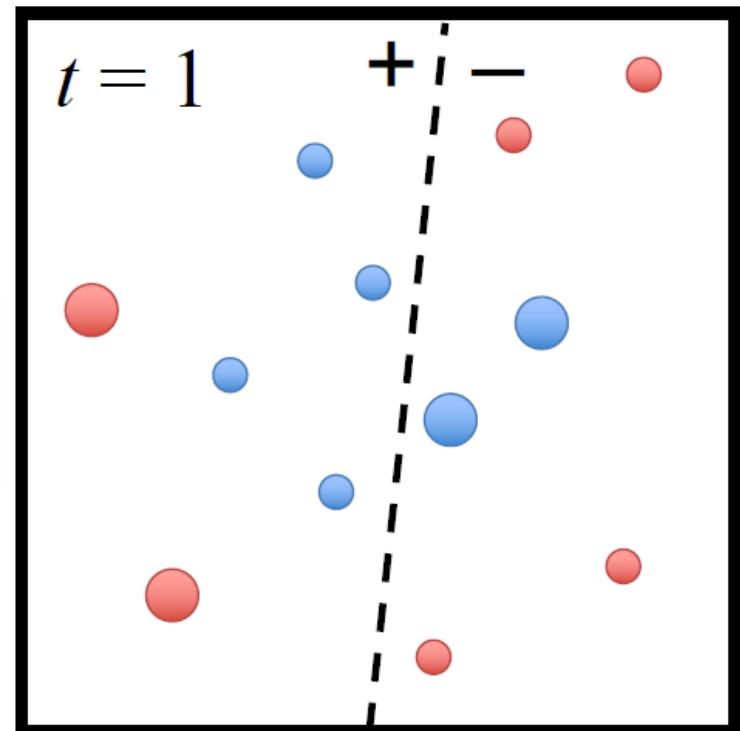
$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



AdaBoost

```
1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp (-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

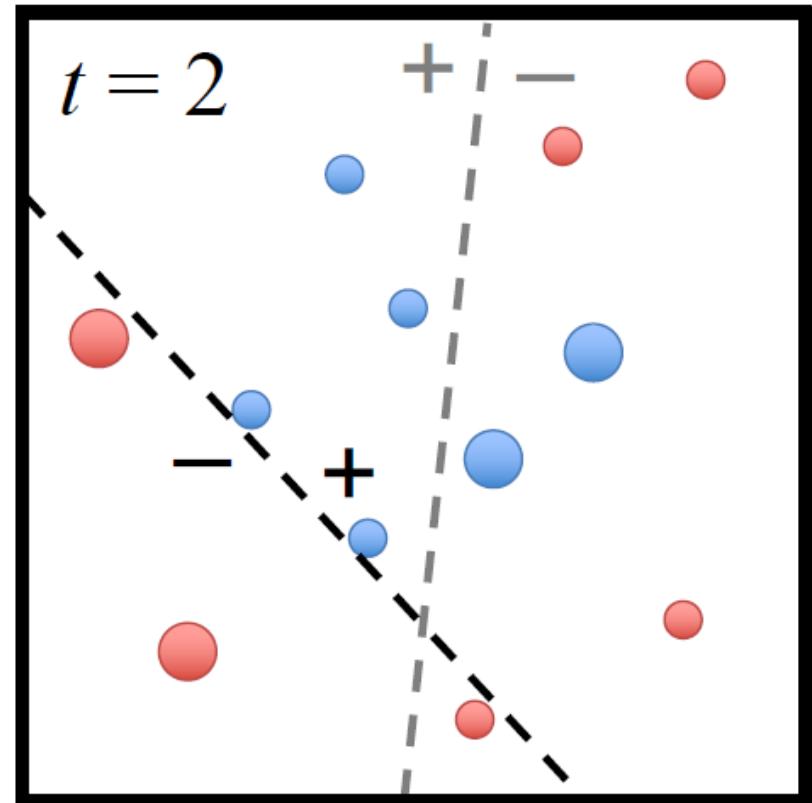


- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

AdaBoost

```
1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
```

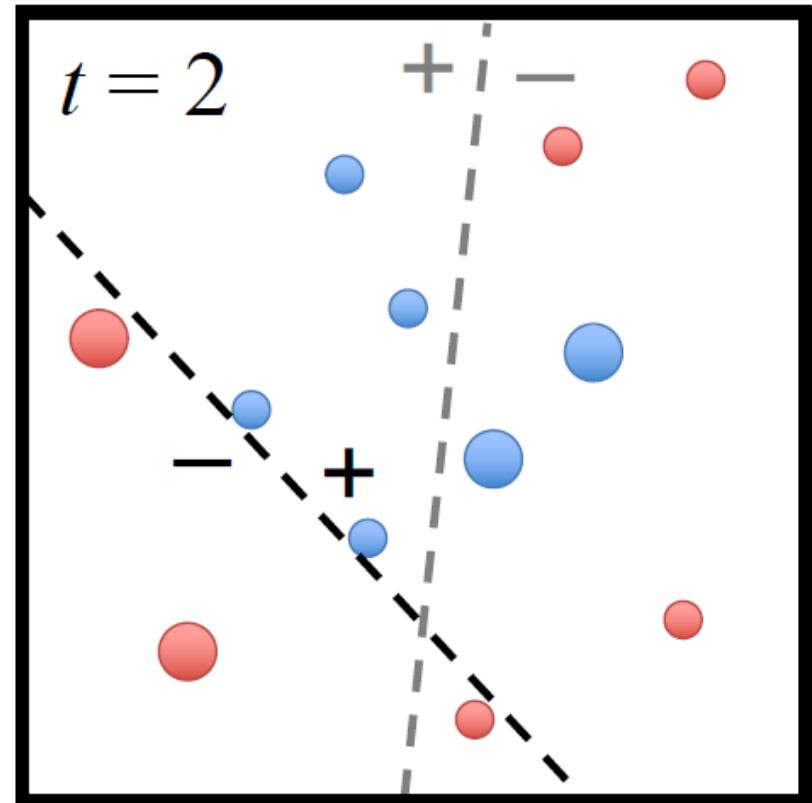
$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



AdaBoost

```
1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

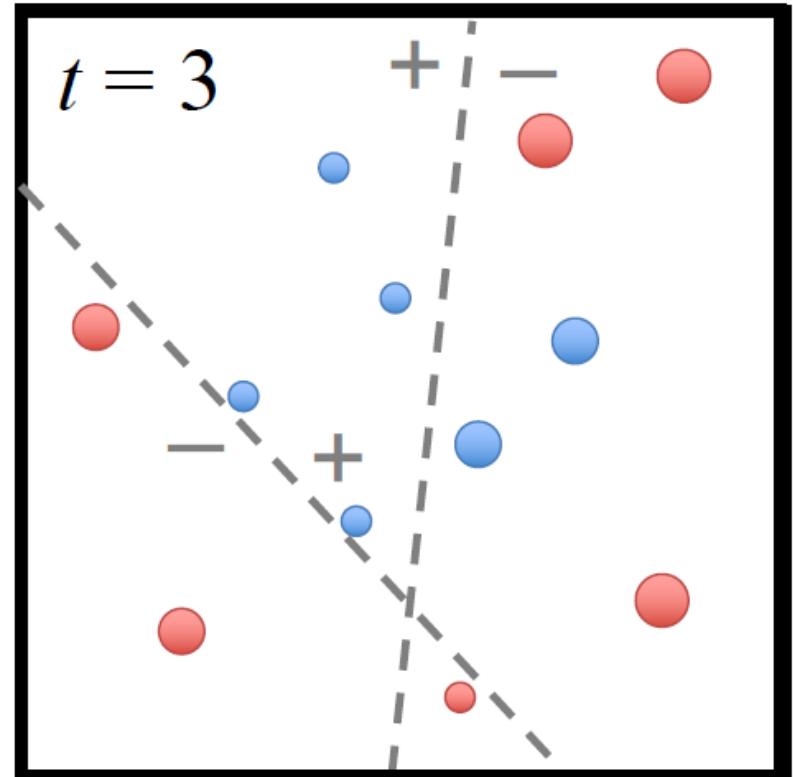


- Compute importance of hypothesis β_t
- Update weights w_t

AdaBoost

- 1: Initialize a vector of n uniform weights \mathbf{w}_1
- 2: **for** $t = 1, \dots, T$ **do**
- 3: Train model h_t on X, y with weights \mathbf{w}_t
- 4: Compute the weighted training error of h_t
- 5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6: Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$$
- 7: Normalize \mathbf{w}_{t+1} to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



AdaBoost

1: Initialize a vector of n uniform weights \mathbf{w}_1

2: **for** $t = 1, \dots, T$

3: Train model h_t on X, y with weights \mathbf{w}_t

4: Compute the weighted training error of h_t

5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

6: Update all instance weights:

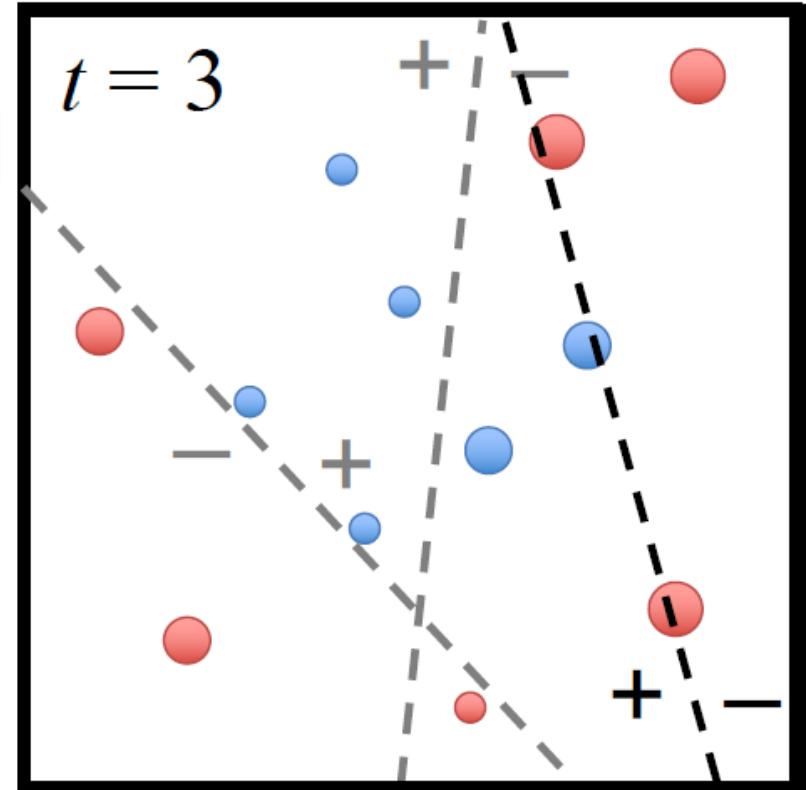
$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$$

7: Normalize \mathbf{w}_{t+1} to be a distribution

8: **end for**

9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



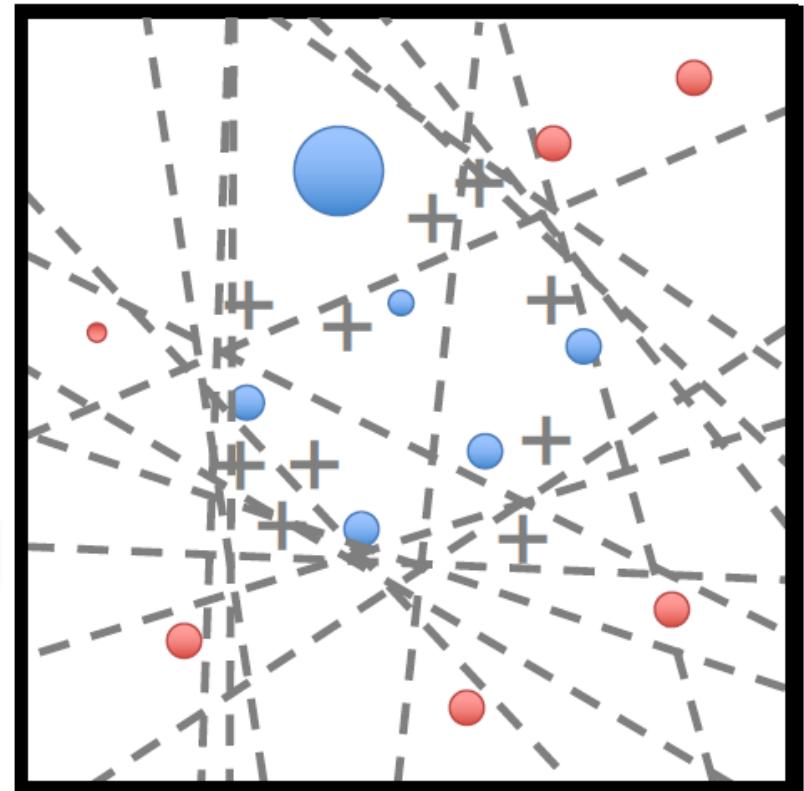
- Compute importance of hypothesis β_t
- Update weights w_t

AdaBoost

$t = T$

- 1: Initialize a vector of n uniform weights \mathbf{w}_1
- 2: **for** $t = 1, \dots, T$
- 3: Train model h_t on X, y with weights \mathbf{w}_t
- 4: Compute the weighted training error of h_t
- 5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6: Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$$
- 7: Normalize \mathbf{w}_{t+1} to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



- Final model is a weighted combination of members
 - Each member weighted by its importance

AdaBoost

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$

2: **for** $t = 1, \dots, T$

3: Train model h_t on X, y with instance weights \mathbf{w}_t

4: Compute the weighted training error rate of h_t :

$$\epsilon_t = \sum_{i:y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$$

5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i)) \quad \forall i = 1, \dots, n$$

7: Normalize \mathbf{w}_{t+1} to be a distribution:

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^n w_{t+1,j}} \quad \forall i = 1, \dots, n$$

8: **end for**

9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

Train with Weighted Instances

Train with Weighted Instances

- For algorithms like logistic regression, can simply incorporate weights w into the cost function
 - Essentially, weigh the cost of misclassification differently for each instance

$$J_{\text{reg}}(\boldsymbol{\theta}) = - \sum_{i=1}^n w_i [y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1 - y_i) \log (1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i))] + \lambda \|\boldsymbol{\theta}_{[1:d]}\|_2^2$$

- For algorithms that don't directly support instance weights (e.g., ID3 decision trees, etc.), use weighted bootstrap sampling
 - Form training set by resampling instances with replacement according to w

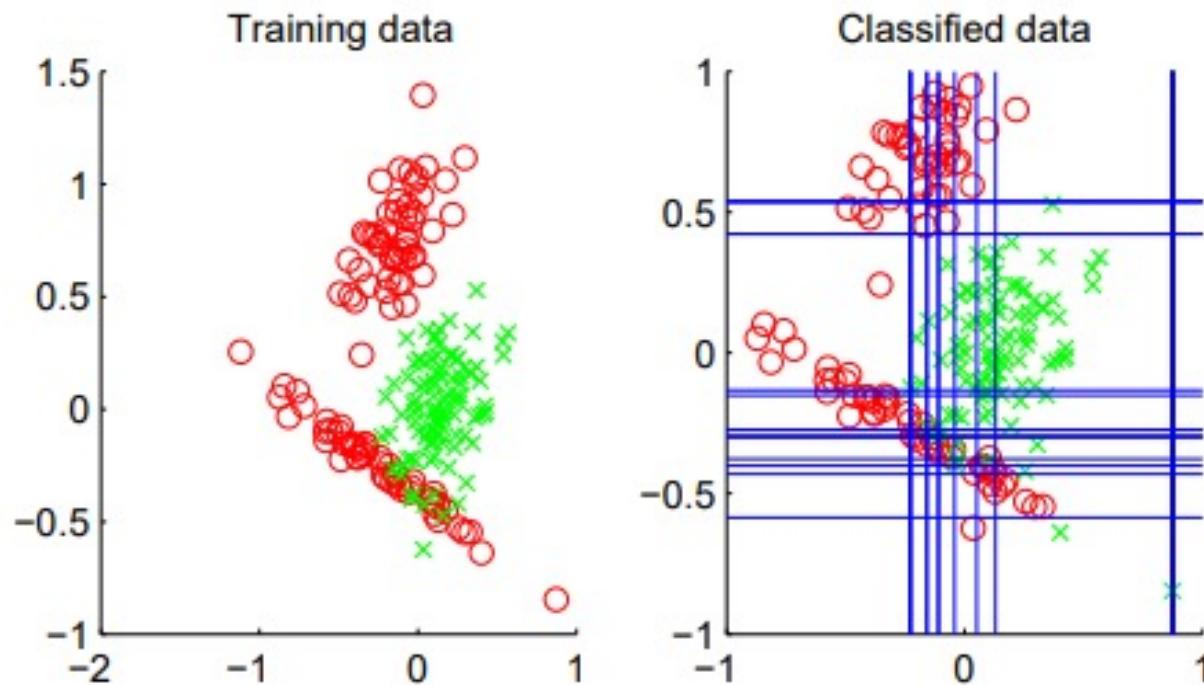
Properties

- If a point is repeatedly misclassified
 - Its weight is increased every time
 - Eventually it will generate a hypothesis that correctly predicts it
- In practice AdaBoost does not typically overfit
- Does not use explicitly regularization

Base Learner Requirements

- AdaBoost works best with “weak” learners
 - Should not be complex
 - Typically high bias classifiers
 - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
 - Can prove training error goes to 0 in $O(\log n)$ iterations
- Examples:
 - Decision stumps (1 level decision trees)
 - Depth-limited decision trees
 - Linear classifiers

AdaBoost with Decision Stumps



AdaBoost in Practice

Strengths:

- Fast and simple to program
- No parameters to tune (besides T) Learn with Cross-Validation
- No assumptions on weak learner Error less than $\frac{1}{2}$

When boosting can fail:

- Given insufficient data
- Overly complex weak hypotheses
- Can be susceptible to noise
- When there are a large number of outliers

Bagging vs Boosting

Bagging vs Boosting

Bagging

Resamples data points

Weight of each classifier
is the same

Only variance reduction

Applicable to complex
models with low bias,
high variance

vs.

Boosting

Reweights data points (modifies their distribution)

Weight is dependent on
classifier's accuracy

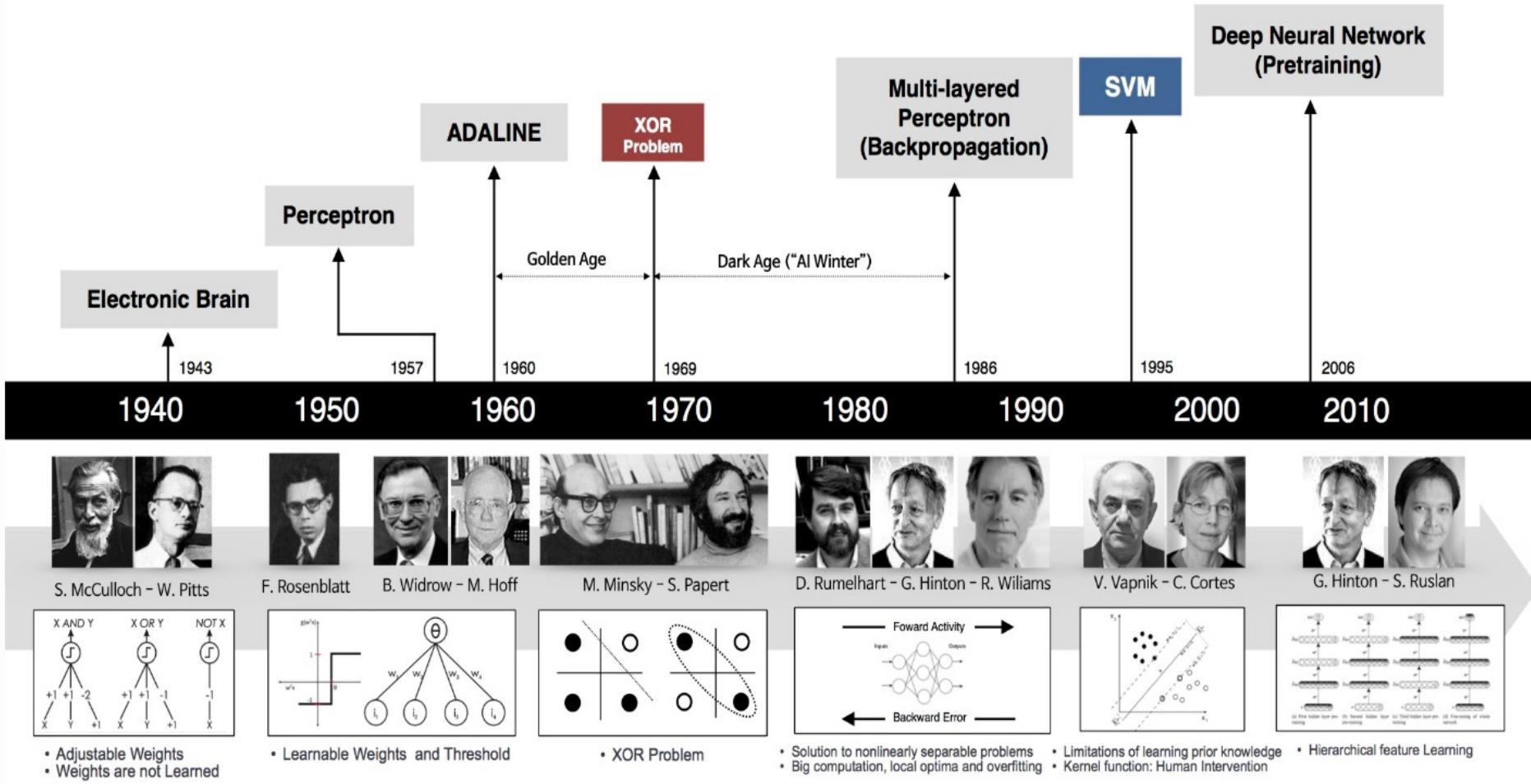
Both bias and variance reduced –
learning rule becomes more complex
with iterations

Applicable to weak
models with high bias,
low variance

Review

- Ensemble learning are powerful learning methods
 - Better accuracy than standard classifiers
- Bagging uses bootstrapping (with replacement), trains T models, and averages their prediction
 - Random forests vary training data and feature set at each split
- Boosting is an ensemble of T weak learners that emphasizes mis-predicted examples
 - AdaBoost has great theoretical and experimental performance
 - Can be used with linear models or simple decision trees (stumps, fixed-depth decision trees)

History of Deep Learning

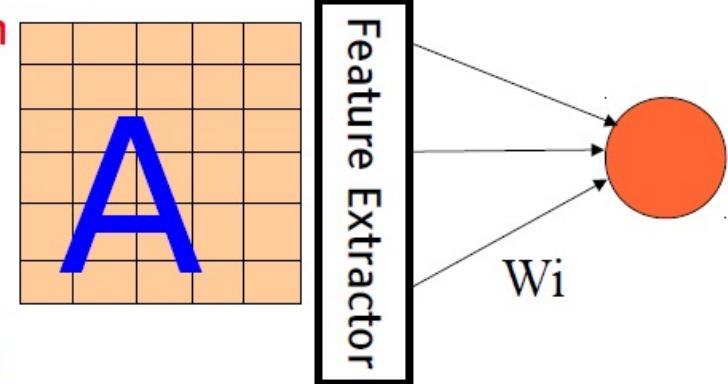


References

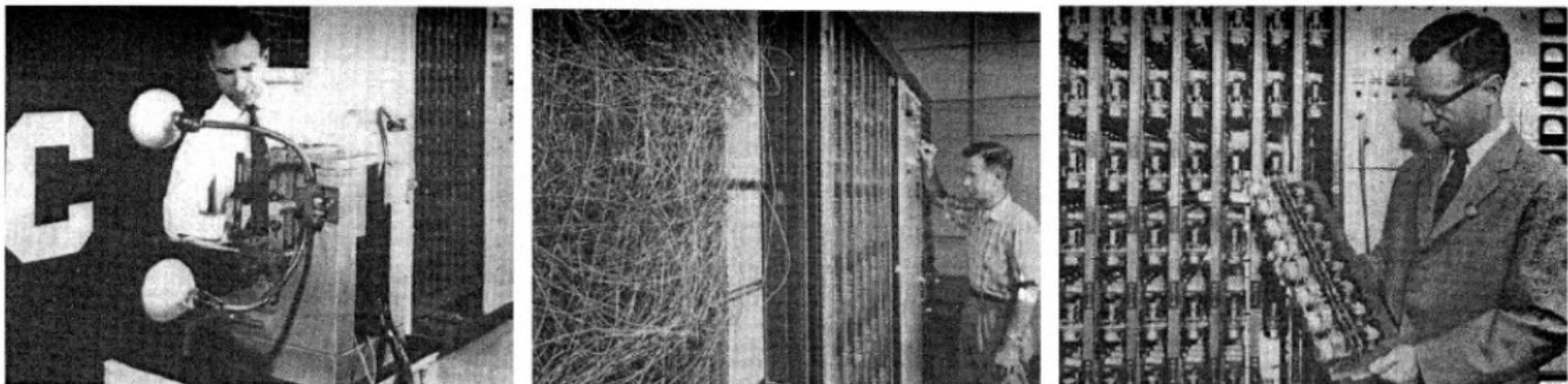
- Deep Learning books
 - <https://d2l.ai/> (D2L)
 - <https://www.deeplearningbook.org/> (advanced)
- Stanford notes on deep learning
 - http://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf

Early Models

- The first learning machine: the Perceptron
 - ▶ Built at Cornell in 1960
- The Perceptron was a linear classifier on top of a simple feature extractor
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.
- Designing a feature extractor requires considerable efforts by experts.

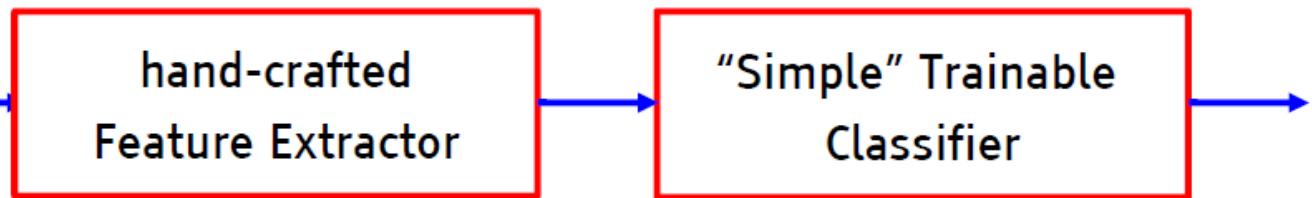


$$y = \text{sign} \left(\sum_{i=1}^N W_i F_i(X) + b \right)$$

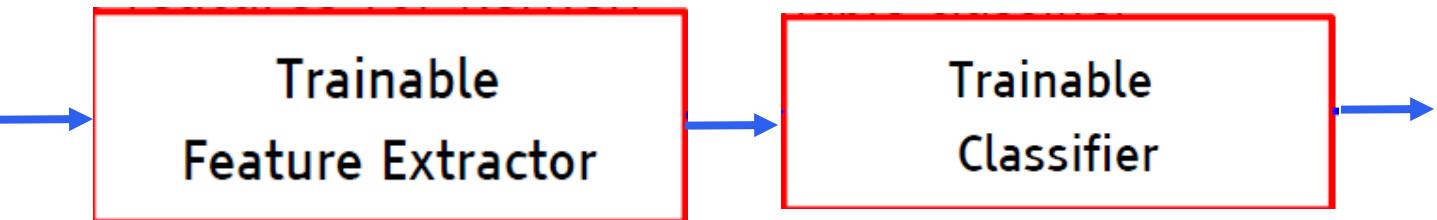


Deep Learning

- The traditional model of pattern recognition (since the late 50's)
 - ▶ Fixed/engineered features (or fixed kernel) + trainable classifier

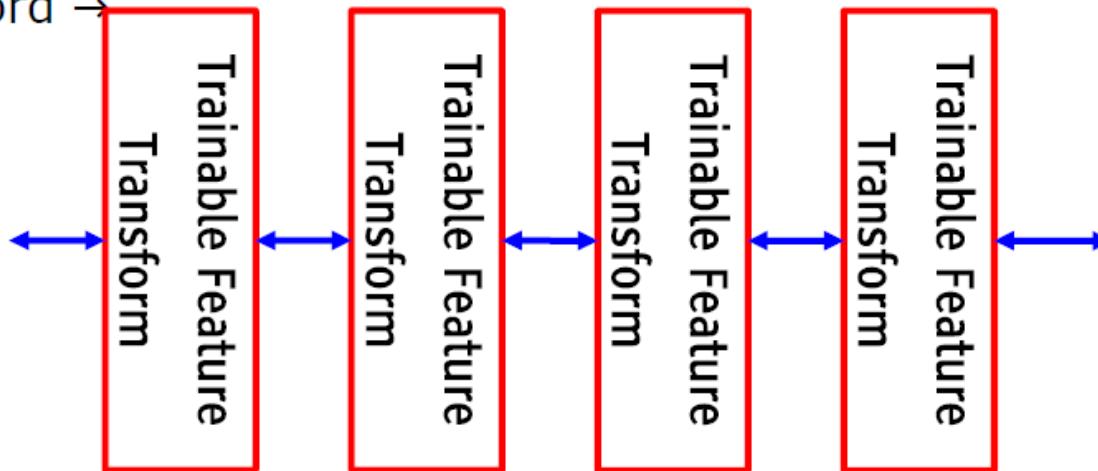


- End-to-end learning / Feature learning / Deep learning

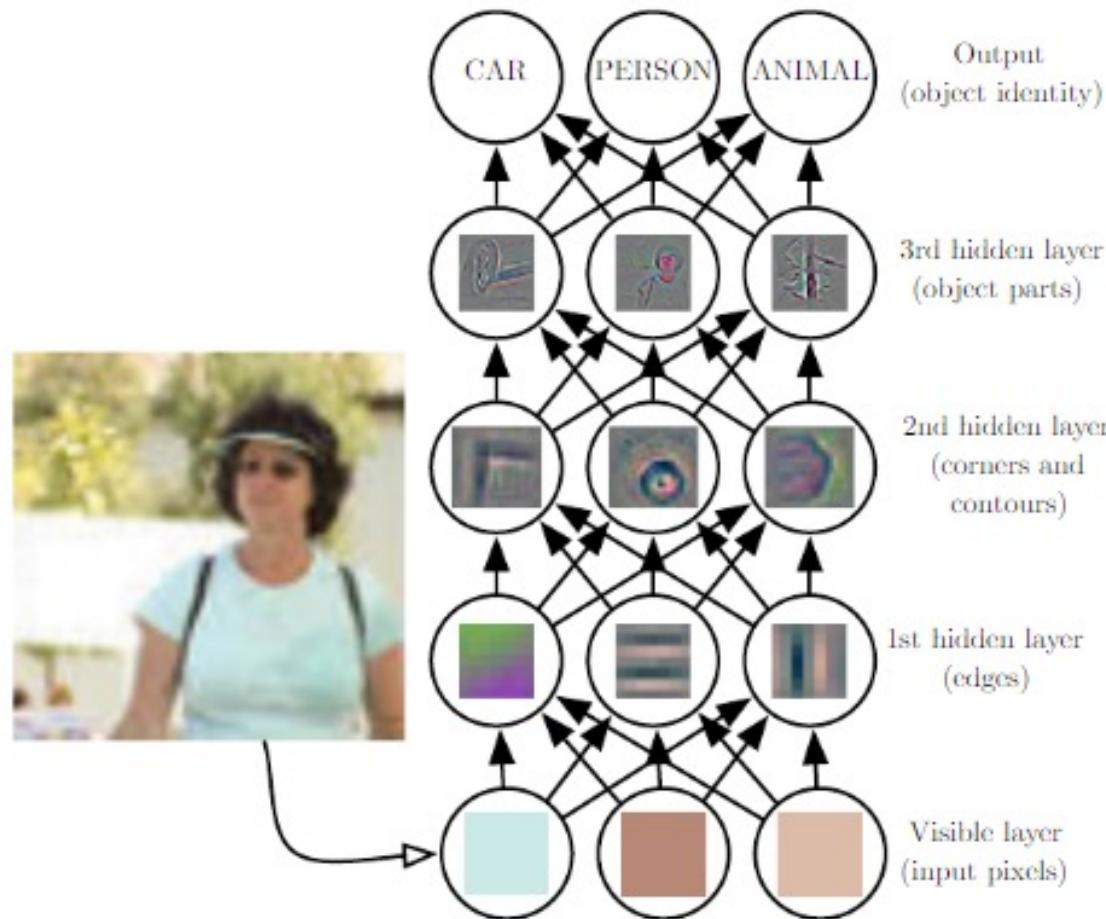


Trainable Feature Hierarchy

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition
 - ▶ Pixel → edge → texton → motif → part → object
- Text
 - ▶ Character → word → word group → clause → sentence → story
- Speech
 - ▶ Sample → spectral band → sound → ... → phone → phoneme → word →



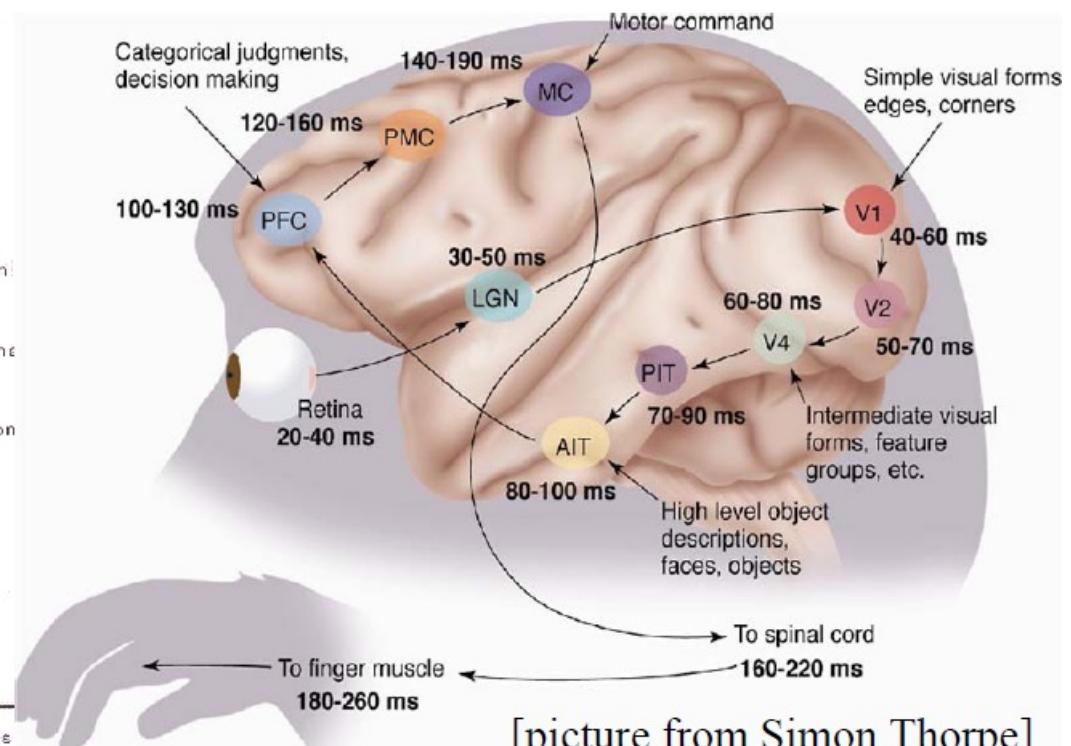
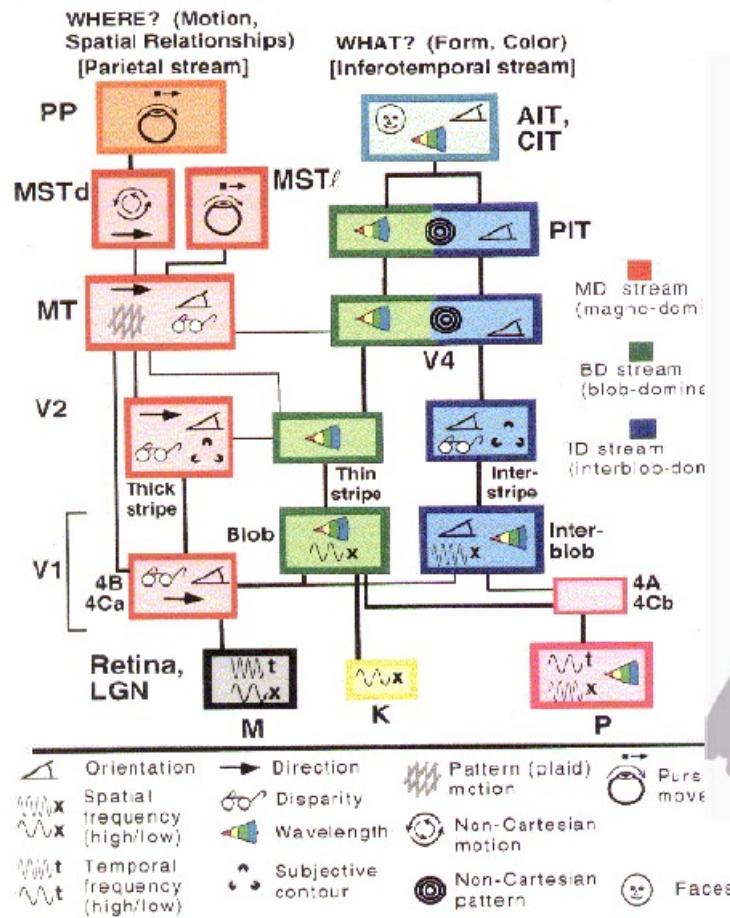
Learning Representations



Deep Learning addresses the problem of learning hierarchical representations

The Visual Cortex is Hierarchical

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT
- Lots of intermediate representations

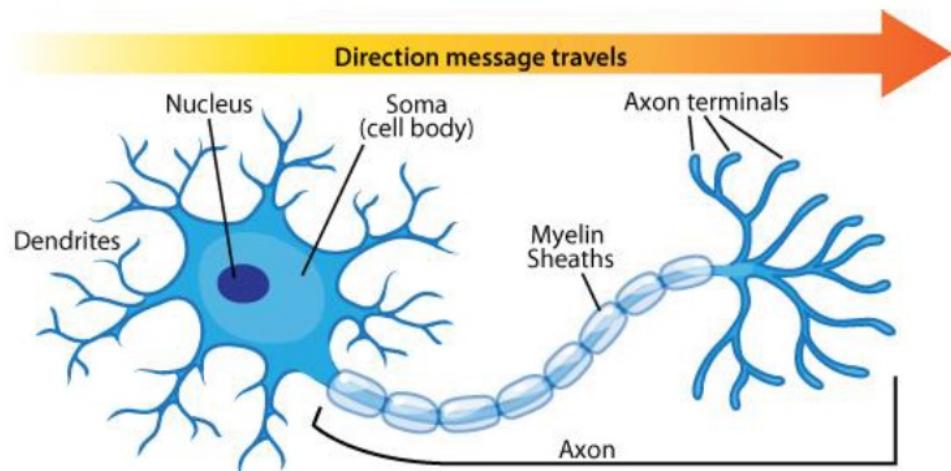


[picture from Simon Thorpe]

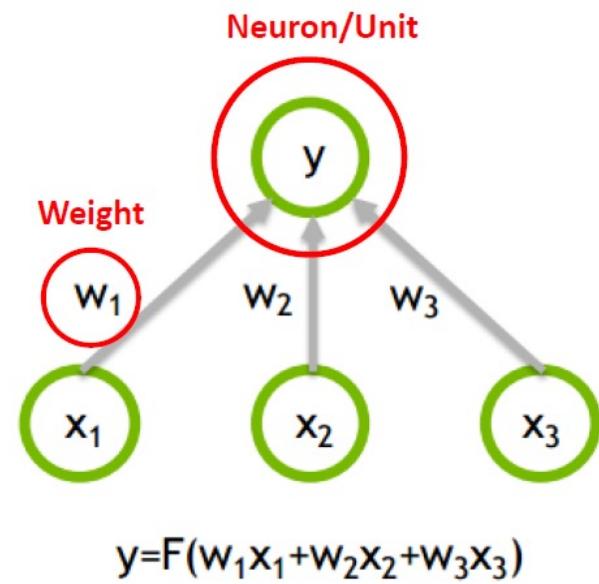
[Gallant & Van Essen]

Analogy to Human Brain

Human Brain



Biological Neuron



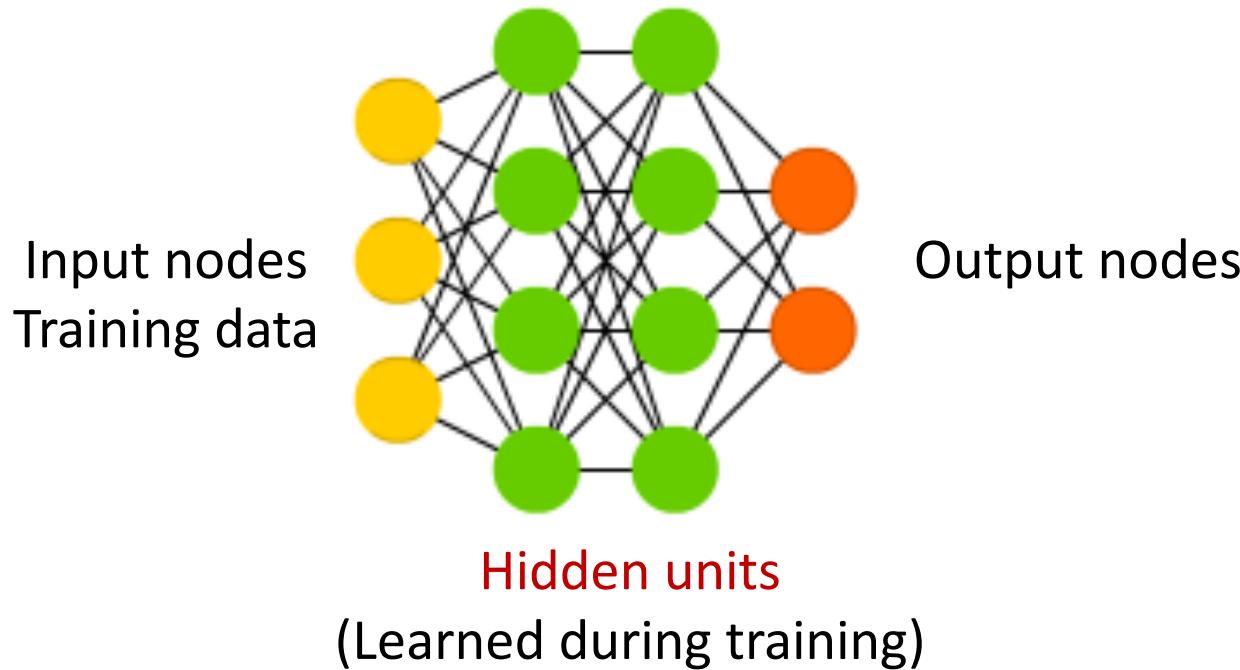
Artificial Neuron

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

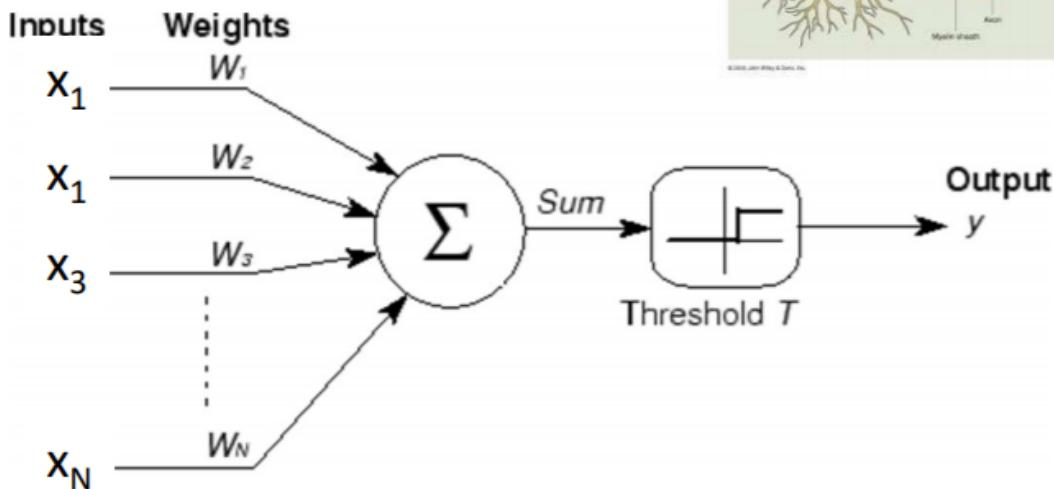
Neural Networks

Deep Feed Forward (DFF)



- Neural networks: made up from node / units connected by links
- Each link has a weight and activation function
- Feed-forward neural networks
 - Training data is input to left and prediction are output on right

Perceptron



$$y = \begin{cases} 1 & \text{if } \sum_i w_i x_i \geq T \\ 0 & \text{else} \end{cases}$$

- A threshold unit
 - “Fires” if the weighted sum of inputs exceeds a threshold

The Perceptron

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- The perceptron uses the following update rule each time it receives a new training instance (\mathbf{x}_i, y_i)

$$\theta_j \leftarrow \theta_j - \frac{1}{2} (h_{\theta}(\mathbf{x}_i) - y_i) x_{ij}$$

- If the prediction matches the label, make no change
- Otherwise, adjust $\boldsymbol{\theta}$

The Perceptron

- The perceptron uses the following update rule each time it receives a new training instance (x_i, y_i)

$$\theta_j \leftarrow \theta_j - \frac{1}{2} (h_\theta(x_i) - y_i)x_{ij}$$

The Perceptron

- The perceptron uses the following update rule each time it receives a new training instance (x_i, y_i)

$$\theta_j \leftarrow \theta_j - \frac{1}{2} (h_\theta(x_i) - y_i)x_{ij}$$

either 2 or -2

- Re-write as

$$\theta_j \leftarrow \theta_j + y_i x_{ij}$$

(only upon misclassification)

Perceptron Rule: If x_i is misclassified, do

$$\theta \leftarrow \theta + y_i x_i$$

Online Perceptron

Let $\theta \leftarrow [0, 0, \dots, 0]$

Repeat:

 Receive training example (x_i, y_i)

 If $y_i \theta^T x_i \leq 0$ // prediction is incorrect

$$\theta \leftarrow \theta + y_i x_i$$

Until stopping condition

Online learning – the learning mode where the model update is performed each time a single observation is received

Batch learning – the learning mode where the model update is performed after observing the entire training set

Batch Perceptron

Let $\theta \leftarrow [0, 0, \dots, 0]$

Repeat:

$$\Delta = [0, 0, \dots, 0]$$

For $i = 1$ to N // Consider all training examples

If $y_i \theta^T x_i \leq 0$ // Prediction is incorrect

$\Delta \leftarrow \Delta + y_i x_i$. // Accumulate all errors

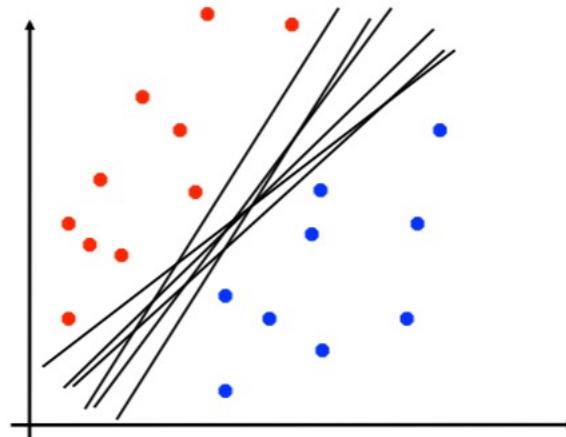
$\theta \leftarrow \theta + \frac{\Delta}{N}$ // Parameter update rule

Until stopping condition

- Guaranteed to find separating hyperplane if data is linearly separable

Perceptron Limitations

- Is dependent on starting point
- It could take many steps for convergence
- Perceptron can overfit
 - Move the decision boundary for every example



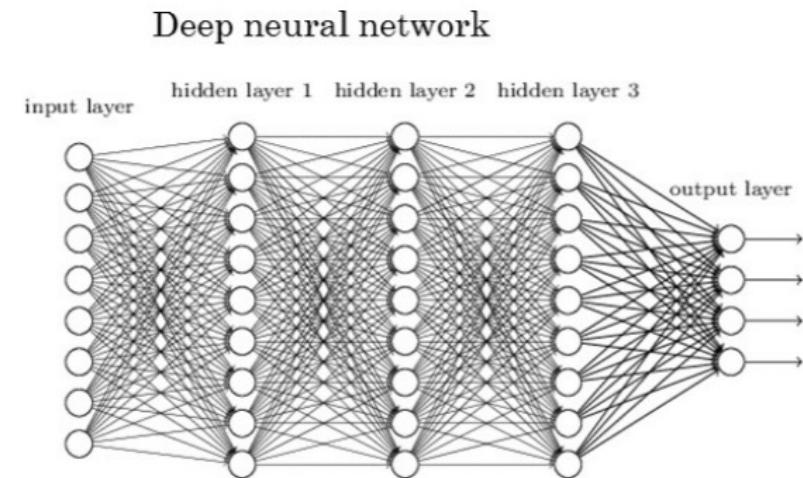
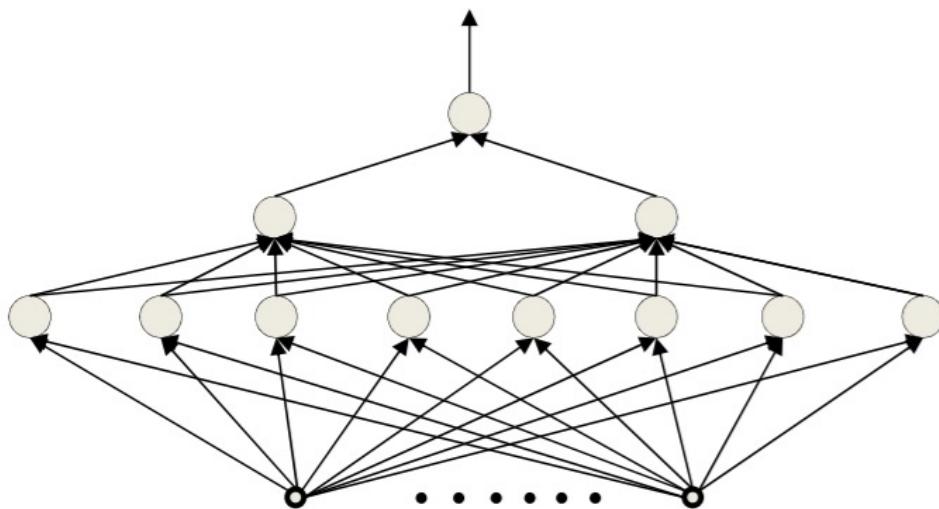
Which of this is
optimal?

History of Perceptrons

- They were popularised by Frank Rosenblatt in the early 1960's.
 - They appeared to have a very powerful learning algorithm.
 - Lots of grand claims were made for what they could learn to do.
- In 1969, Minsky and Papert published a book called “Perceptrons” that analysed what they could do and showed their limitations.
 - Many people thought these limitations applied to all neural network models.
- The perceptron learning procedure is still widely used today for tasks with enormous feature vectors that contain many millions of features.

They are the basic building blocks for
Deep Neural Networks

Multi-Layer Perceptron

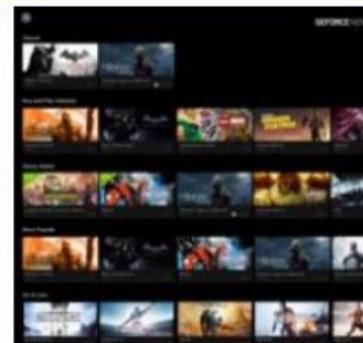
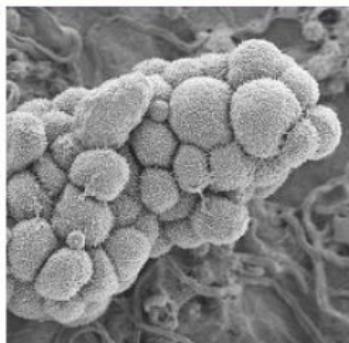


- A network of perceptrons
 - Generally “layered”



Deep Learning Applications

DEEP LEARNING EVERYWHERE



INTERNET & CLOUD

Image Classification
Speech Recognition
Language Translation
Language Processing
Sentiment Analysis
Recommendation

MEDICINE & BIOLOGY

Cancer Cell Detection
Diabetic Grading
Drug Discovery

MEDIA & ENTERTAINMENT

Video Captioning
Video Search
Real Time Translation

SECURITY & DEFENSE

Face Detection
Video Surveillance
Satellite Imagery

AUTONOMOUS MACHINES

Pedestrian Detection
Lane Tracking
Recognize Traffic Sign

Success stories: Speech recognition

www.technewsworld.com/story/84013.html

40 maps that explain Amazon Web Services Primers | Math & Procs deeplearning.net/tutor Deep Learning Tutors deep learning PHILIPS - Golden Ears Language Technology MyIDCare - Dashboard Other bookmarks

TECHNEWSWORLD EMERGING TECH

SEARCH

Computing Internet IT Mobile Tech Reviews Security Technology Tech Blog Reader Services

Microsoft AI Beats Humans at Speech Recognition

By Richard Adhikari Oct 20, 2016 11:40 AM PT

Print Email

G+ 5
Tweet 25
Share 45
in Share 11
Share 0
share 104



Image: Adobe Stock

How do you feel about Black Friday and Cyber Monday?

- They're great -- I get a lot of bargains!
- The deals are too spread out -- I'd prefer just one day.
- They're a fun way to kick off the holiday season.
- I don't like the commercialization of Thanksgiving Day.
- They're crucial for the retail industry and the economy.
- The deals typically aren't that good.

Vote to See Results

E-Commerce Times

Black Friday Shoppers Hungry for New Experiences, New Tech

Pay TV's Newest Innovation: Giving Users Control

Apple Celebrates Itself in \$300 Coffee Table Tome

AWS Enjoys Top Perch in IaaS, PaaS Markets

US Comptroller Gears Up for Blockchain and

Success stories: Machine Translation

The screenshot shows a web browser window displaying a blog post from the Google Products blog. The URL in the address bar is <https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate/>. The page title is "Found in translation: More accurate, fluent sentences in Google Translate". The author is Barak Turovsky, a Product Lead at Google Translate. The date of the post is November 15, 2016. The content discusses the improvements in Google Translate's accuracy and fluency over time. A small blue circular icon with a white arrow is visible in the bottom right corner of the page.

TRANSLATE NOV 15, 2016

Found in translation: More accurate, fluent sentences in Google Translate

Barak Turovsky
PRODUCT LEAD, GOOGLE TRANSLATE

In 10 years, Google Translate has gone from supporting just a few languages to 103, connecting strangers, reaching across language barriers and even helping

Success stories: Image segmentation

