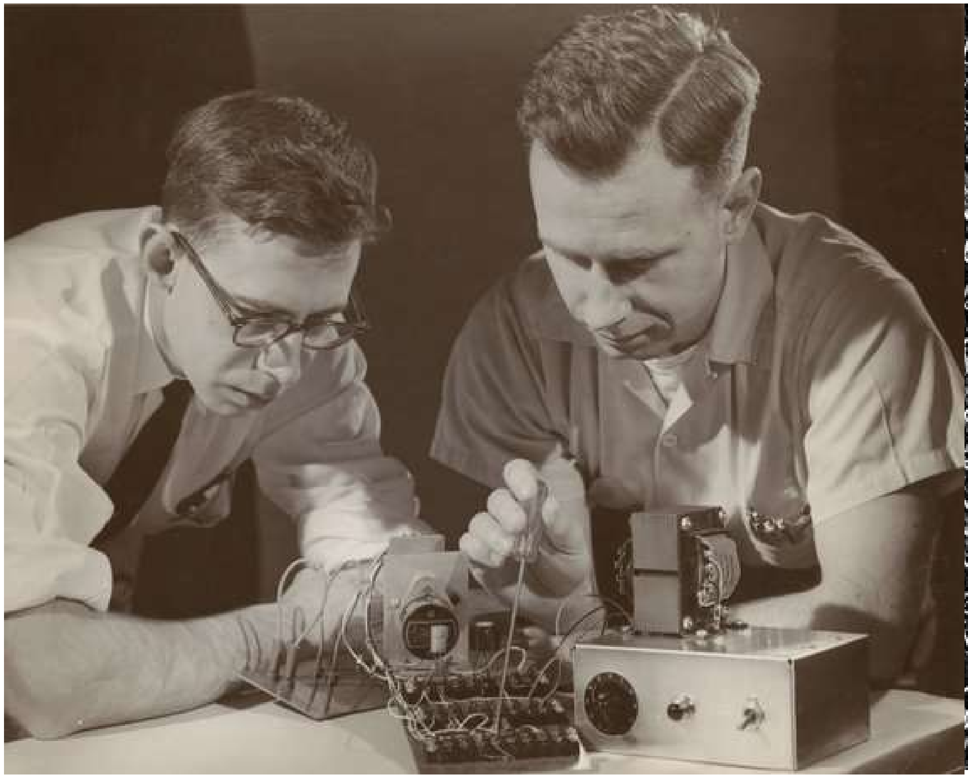
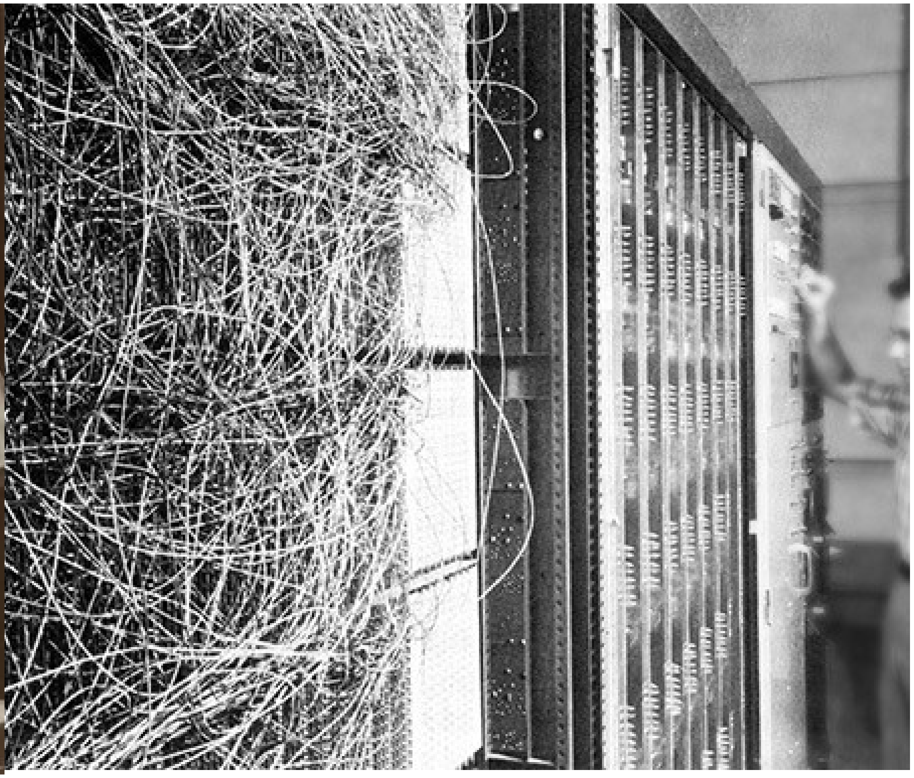
Machine Learning EN.601.475/675

DR. PHILIP GRAFF

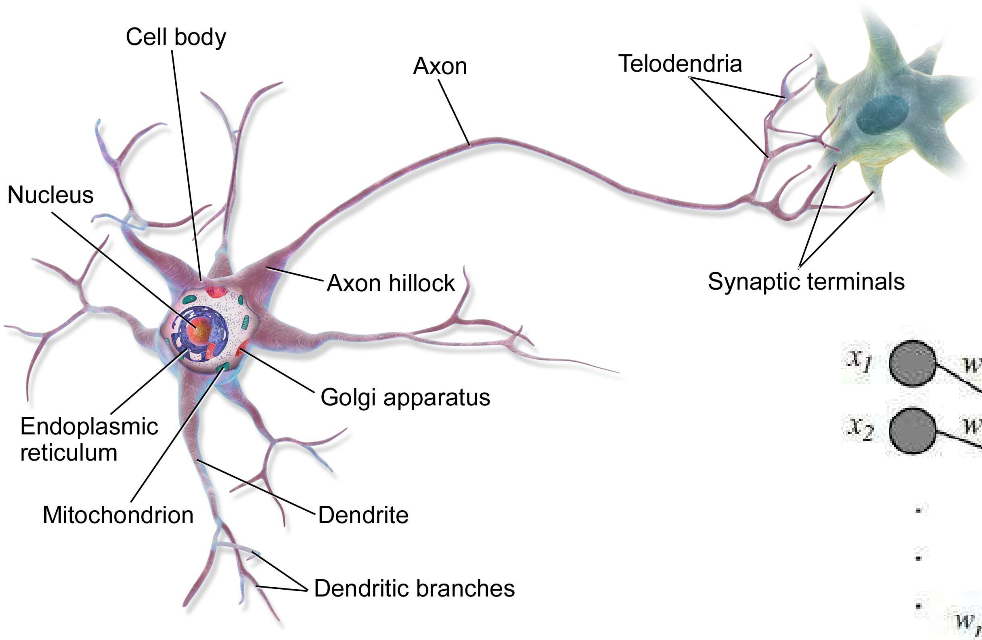
Perceptron

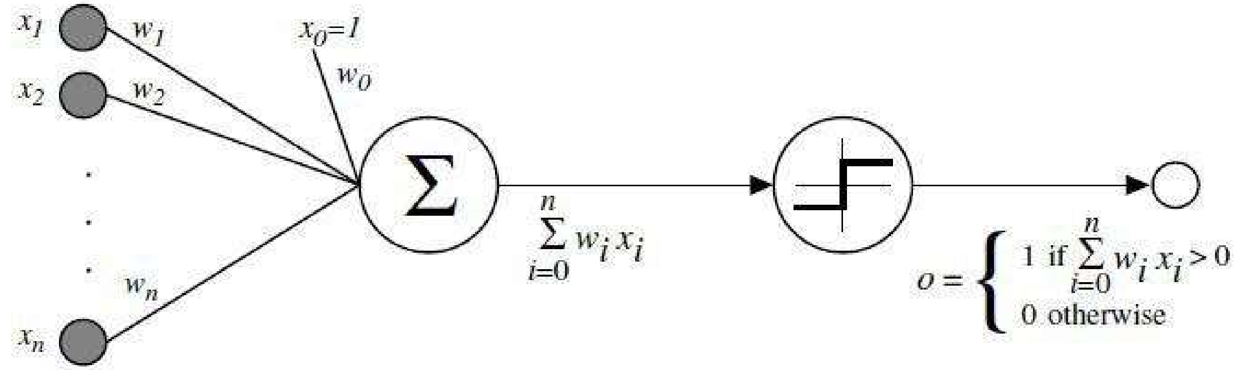
Perceptron Origins





The Neural View





Why the Perceptron?

The first learning algorithm

A way of thinking about data:

- Geometric interpretation of data

A way of using data:

- Online learning algorithms

Recall our Definition

Fitting a function to data

Fitting: Optimization, what parameters can we change?

Function: Model, loss function

Data: Data/model assumptions? How we use data?

ML Algorithms: minimize a function on some data

Setting

Stochastic gradient descent

- One example at a time (a.k.a. *online learning*)

Linear classifier: $\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x}) = \begin{cases} +1 & \mathbf{w}^T \mathbf{x} > 0 \\ -1 & \mathbf{w}^T \mathbf{x} < 0 \end{cases}$

Output: $y \in \{-1, +1\}$

Let's take the simplest and most direct loss function we can think of

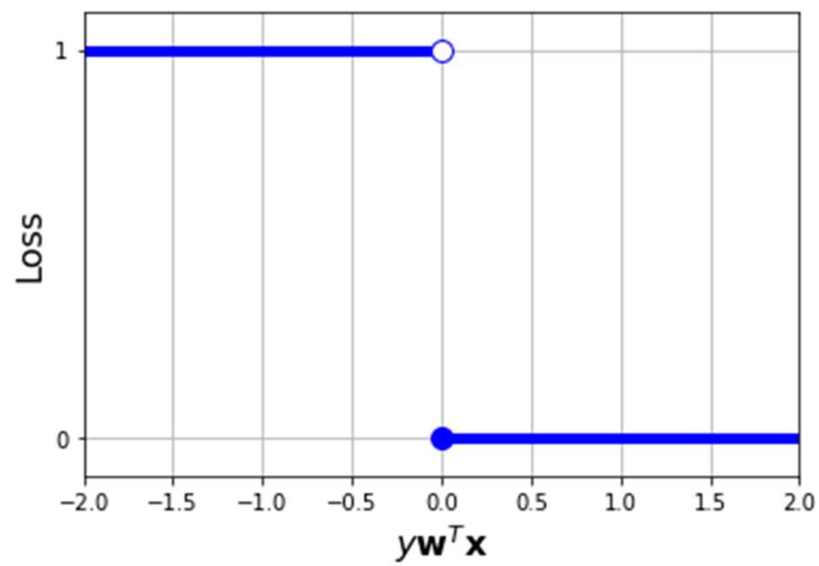
$0/1$ Loss Function

If we are wrong: loss of 1

If we are correct: loss of 0

Simplest thing we can think of

Implication: Only make a change when we make a mistake



Hard to Minimize

Minimizing $0/1$ loss function is difficult

- Step function without a gradient

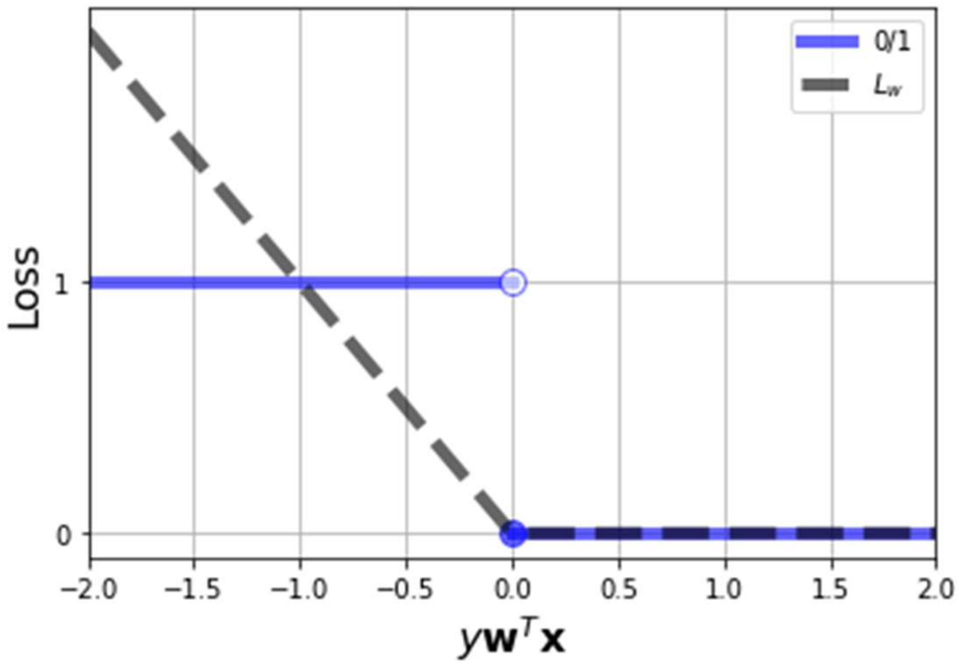
$$\operatorname{sign}(y\mathbf{w}^T\mathbf{x}) = \begin{cases} +1 & \mathbf{w}^T\mathbf{x} > 0 \text{ and } y = +1 \\ +1 & \mathbf{w}^T\mathbf{x} < 0 \text{ and } y = -1 \\ -1 & \mathbf{w}^T\mathbf{x} < 0 \text{ and } y = +1 \\ -1 & \mathbf{w}^T\mathbf{x} > 0 \text{ and } y = -1 \end{cases} = \begin{cases} +1 & \text{correct} \\ -1 & \text{incorrect} \end{cases}$$

Replace with something similar

Full dataset: $$L_w(y) = \sum_{i=1}^{N} \max\left(0, -y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right)$$

Single example: $$L_w(y_i) = \max\left(0, -y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right)$$

Alternative Loss Function



$$L_w(y_i) = \max\left(0, -y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right)$$

Gradient

Let's calculate the gradient of our loss function

$$\frac{\partial}{\partial \mathbf{w}} L_w(y_i) = \begin{cases} 0 & y_i \mathbf{w}^T \mathbf{x}_i > 0 \\ -y_i \mathbf{x}_i & y_i \mathbf{w}^T \mathbf{x}_i < 0 \end{cases}$$

What about $y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i = 0$?

- We can ignore this case, unlikely to occur exactly

Update Rule

What is the update to our current weights for each new point?

$$\mathbf{w}^{i+1} = \mathbf{w}^{i} - \eta \partial L_w(y_i)$$

$$\mathbf{w}^{i+1} = \mathbf{w}^{i} + \eta (y_i - \hat{y}_i)\mathbf{x}_i$$

Let η absorb the factor of 2

$$\hat{y}_i = \mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i) \qquad y_i - \hat{y}_i = \begin{cases} 0 & \text{correct} \\ +2 & \text{incorrect, } y_i = +1 \\ -2 & \text{incorrect, } y_i = -1 \end{cases} = \begin{cases} 0 & \text{correct} \\ 2y_i & \text{incorrect} \end{cases}$$

Update Rule

Why is this a good update? $\mathbf{w}^{i+1} = \mathbf{w}^{i} + \eta(y_i - \hat{y}_i)\mathbf{x}_i$

Start with simplification for incorrect predictions:

$$\mathbf{w}^{i+1} = \mathbf{w}^{i} + \eta y_i \mathbf{x}_i$$

$$\begin{aligned}
(\mathbf{w}^{i+1} \cdot \mathbf{x}_i) y_i &= (\mathbf{w}^{i} \cdot \mathbf{x}_i) y_i + \eta y_i (\mathbf{x}_i \cdot \mathbf{x}_i) y_i \\
&= (\mathbf{w}^{i} \cdot \mathbf{x}_i) y_i + \eta y_i y_i (\mathbf{x}_i \cdot \mathbf{x}_i) \\
&= (\mathbf{w}^{i} \cdot \mathbf{x}_i) y_i + \eta \|\mathbf{x}_i\|^2 \\
&> (\mathbf{w}^{i} \cdot \mathbf{x}_i) y_i
\end{aligned}$$

Update Rule

Our prediction has improved!

- When incorrect, $\mathbf{w} \cdot \mathbf{x}_i y_i < 0$, so an increase is better

What does this say about the next time we see this data point?

- Nothing - the prediction may still be incorrect
- However, we are ***moving in the right direction***

Perceptron Algorithm

Initialize **w** and η

On each round

1. Receive example **x**
2. Predict $\hat{y} = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x})$
3. Receive correct label $y \in \{-1, +1\}$
4. Suffer loss $l_{0/1}(y, \hat{y})$
5. Update **w** $\mathbf{w}^{i+1} = \mathbf{w}^{i} + \eta(y_i - \hat{y}_i)\mathbf{x}_i$

Perceptron

Fitting a function to data

Fitting: Stochastic gradient descent

Function: $0/1$ loss with linear function

Data: Update using single example at a time

A Way of Thinking About Data

How We Represent Data

A convenient way of representing data

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \qquad \mathbf{x}_i \in \Re^M \qquad y_i \in L$$

L is the set of class labels

Also: a convenient way of *thinking about data*

Geometric Representations

Each example, $\mathbf{x}_i$, represents a point in M-dimensional space

Classification

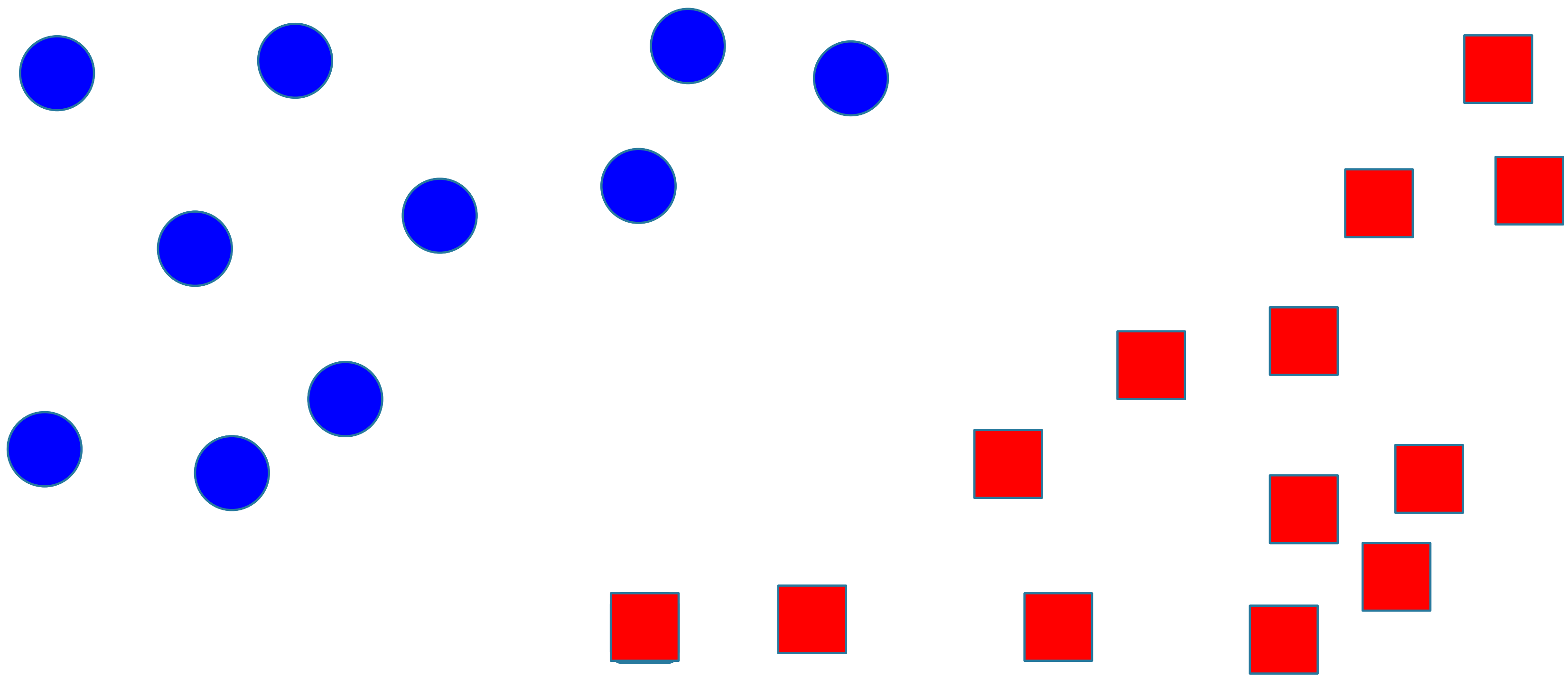
- Divides space into 2 (or more) parts

Examples labeled according to where they are

Linear classification:

- A linear decision boundary
- A hyper-plane (M-1 dimensions)

Geometric Representation



Discriminant Linear Classifiers

Previously: we forced prediction by thresholding output

Now: output -1 or 1 directly by passing through “sign()” function

Classification boundary represented by **w**

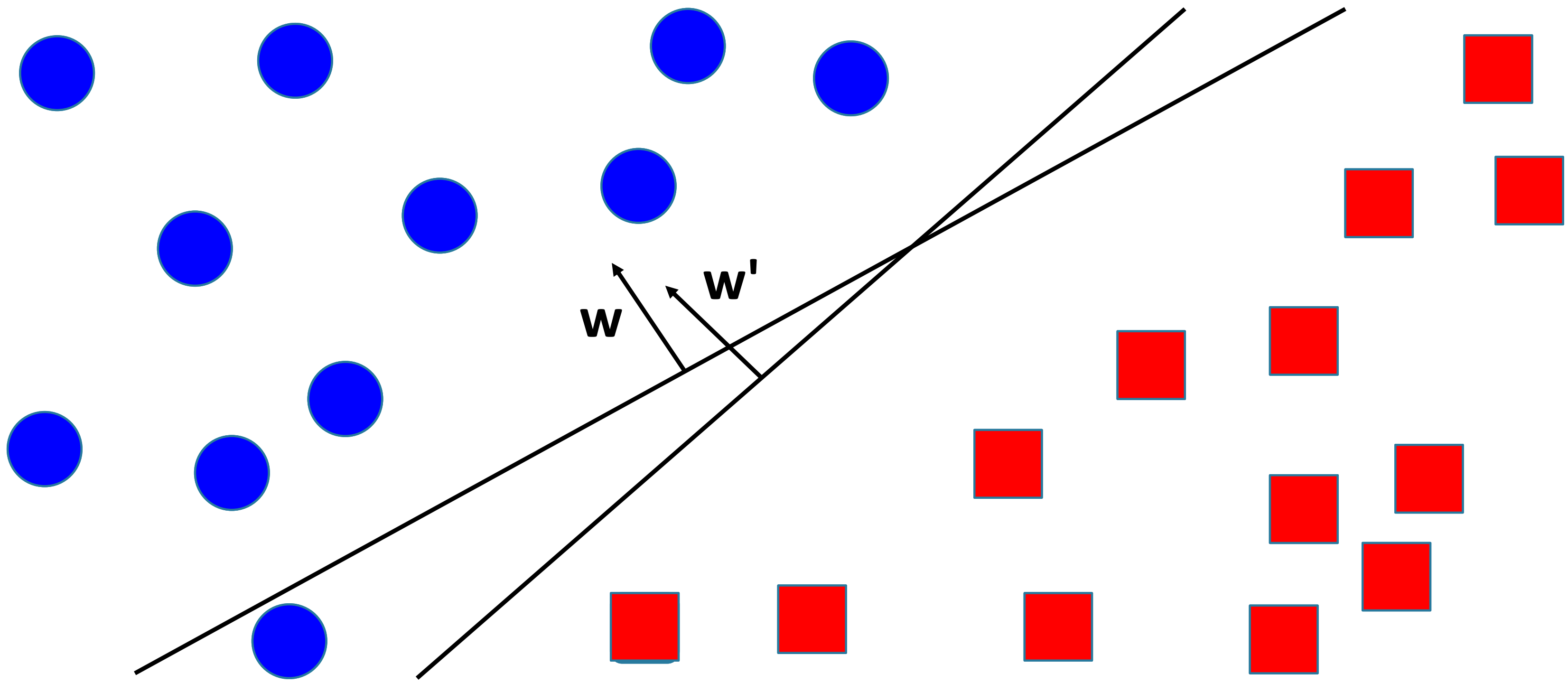
- **w** is a vector that is orthogonal (normal) to the decision boundary

Prediction

- The sign of the prediction indicates the side of the boundary

Assume decision boundary passes through origin

Geometric Representation



Generalized Linear Function

This is a linear function

$$\mathbf{w} \cdot \mathbf{x}$$

Pass the output through a non-linear function

$$\hat{y} = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x})$$

Generalized linear function!

In this case, the output is either $+1$ (positive) or -1 (negative)

Discriminant Linear Classifier

Magnitude of $\mathbf{w} \cdot \mathbf{x}$ doesn't matter
- Relative magnitudes matter (as we'll discuss soon)

Many representations of the same boundary
- Can scale $\mathbf{w}$ without changing prediction

A Way of Using Data

Batch Algorithms

Train
- Given training data {**X**,**Y**}
- Learn the parameters of the model

Test
- Given a previously unseen unlabeled example **x**
- Assign a label according to learned parameters

Batch algorithms
- Takes a large set of examples at once

Idea: to improve speed, use a stochastic optimization algorithm

Assumptions Behind Batch

The data is labeled with a consistent hypothesis

- There is one optimal hypothesis that fits all the data
- This hypothesis is not necessarily in our hypothesis class
- There may be some noise in the labels

Examples are drawn from a common distribution iid

- Identically: each draw is from the same distribution
- Independently: each draw is independent
- This allows us to decompose the data likelihood in logistic regression

Removing Assumptions

No train/test data
- Instead access to a data stream

Concept drift
- No consistency in hypothesis used to label each example

Examples not IID
- Data distribution may change
- Examples may be dependent

Adversary model
- An adversary controls the stream

Online Learning Algorithms

Online learning handles all these cases

Make no assumptions about the data

- Distribution of the data
- Hypothesis class to describe it

Online

- Model learning interacts with a data stream or a human
- Predictions needed after each example

Do the best you can on each single example

Why Online Learning?

Few assumptions means widely applicable

Strong theoretical foundation

Scales to large amounts of data
- Streaming processes one example at a time

Updates model without retraining
- Spam filter: a single new spam email can update the model

Handles a changing world
- No assumptions about how data should behave

Online Learning Framework

On each round:

1. Receive a single example **x**
2. Predict $\hat{y}$ for **x** using stored hypothesis
3. Receive correct label *y*
4. Suffer loss $l(y, \hat{y})$
5. Create new hypothesis using current hypothesis, **x**, and *y*

How to Update?

We know when to update, so how do we do it?

Change **w** so that it is *less wrong* on the given example

- Less wrong = smaller loss
- A gradient step in the right direction

We need to answer:

- What is our model? (e.g. linear classifiers)
- What is our loss function/objective?

Expected Behavior

We know that we improve with each update

What can be said about the expected number of mistakes over a data stream?

- A lot?
- A little?
- Infinite?

Linearly Separable

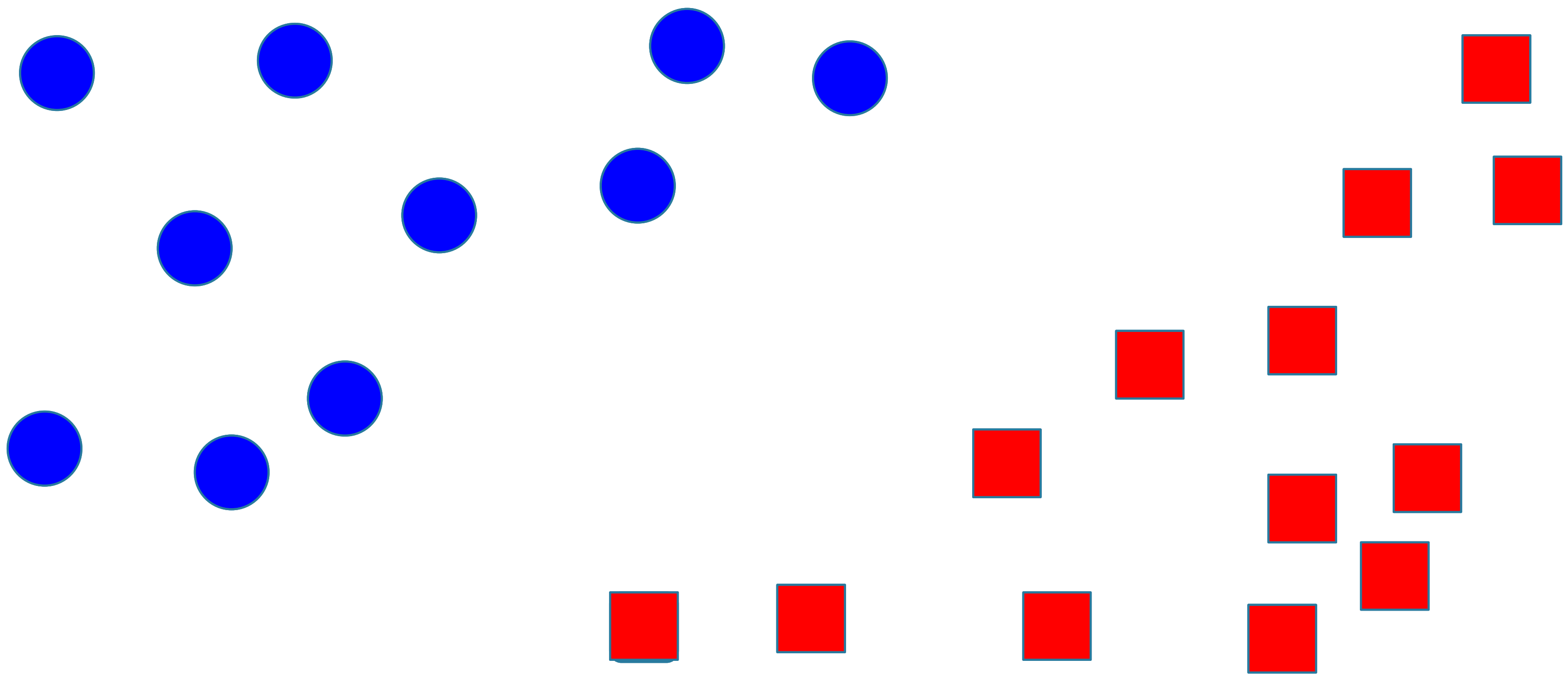
Question: Is there a linear boundary (hyper-plane) that correctly separates all of the examples?

- Yes → the examples are linearly separable
- No → the examples are not linearly separable

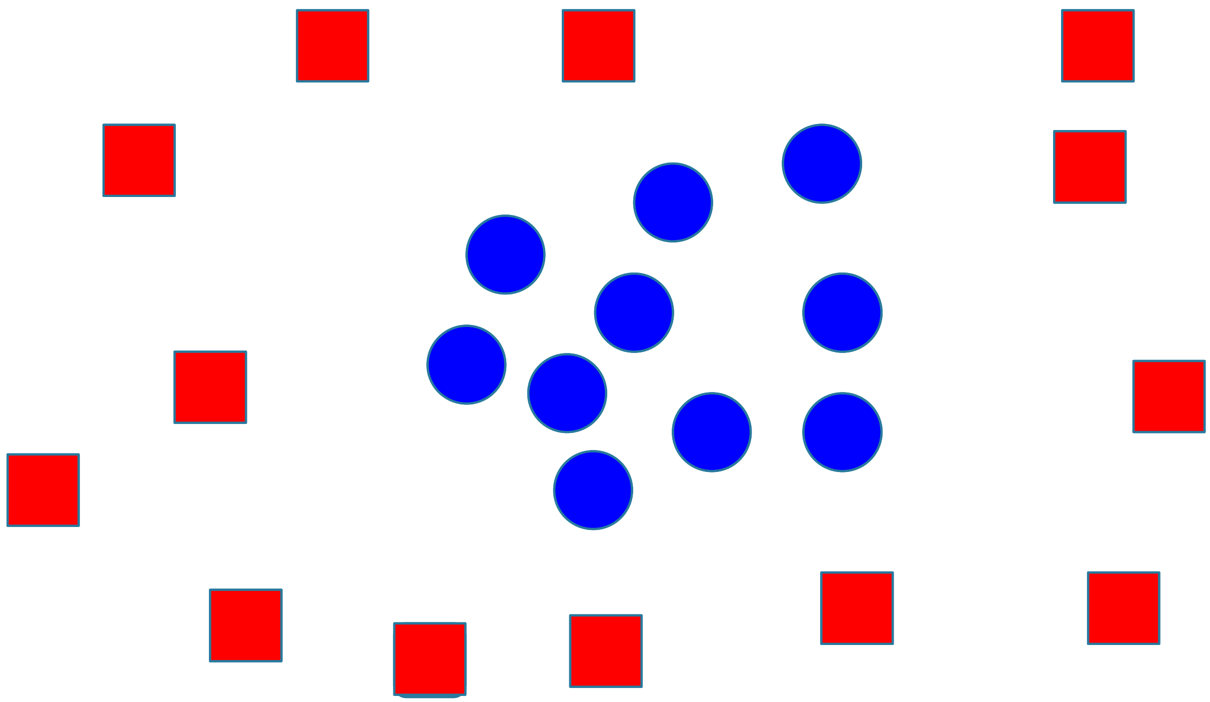
This is a separate issue from a consistent or optimal hypothesis

- Could be not linearly separable but consistent

Linearly Separable



Not Linearly Separable



Convergence

Assume the data are linearly separable

Will the perceptron algorithm converge? $\|\mathbf{w}^{i+1} - \mathbf{w}^{i}\|^2 < \epsilon$

- i.e. Will it stop making updates?

Yes! Perceptron is guaranteed to converge on linearly separable data

If it converges, then updates stop

- If updates stop, then it makes a finite number of mistakes

Convergence

Theorem: If the provided data are linearly separable with margin γ, then a perceptron will terminate in iterations linear with respect to the number of examples

- Different theoretical frameworks for analyzing online algorithms
- E.g. mistake bounds

There are many versions of this theorem and proof

Not Linearly Separable

If the data are not linearly separable then we will not necessarily converge

Trivial to make it linearly separable

- Add a unique feature to each example
- Not very useful in practice
- We'll learn better ways to do this

Oscillating Models

For non-separable data, **w** will oscillate

For separable data, it will converge

- But *which* decision boundary will it converge to?

Both cases mean that **w** is highly dependent on the order of the data in the stream

Model Combinations

Solution: Take the best model over time

- A model that was good on many examples should be trusted more than the latest model

Voting: Let models collectively vote on the prediction

- Save **w** on each round

$$\hat{y} = \text{sign}\left(\sum_i \text{sign}(\mathbf{w}_i \cdot \mathbf{x})\right)$$

Averaging: Average the models for a joint prediction

- Average the value of **w** from each round

$$\hat{y} = \text{sign}\left(\left(\sum_i \mathbf{w}_i\right) \cdot \mathbf{x}\right)$$

Online Algorithms on the Rise

Online algorithms are widely applicable to large-scale data problems

In practice, many algorithms now have stochastic optimizers

- Faster for lots of data
- Better for GPU hardware (mini-batch)

Lingering Questions

We have discussed online training of discriminant functions for linear classifiers

- What would we do if we saw all of the data? (batch)

Perceptron converges to a separating hyper-plane

- But there may be many
- Which one is best?

Our solution for non-linear data was pretty silly

- What can we do for non-linearly separable data?
- We'll find out next week!

Final Notes

Coming up in a few weeks: multi-layer perceptrons

- We can stack perceptrons on top of each other
- This creates a NON-linear function
- Overcomes historical limitations

Stacking of perceptrons (neurons) creates a ***neural network***

Next time: Support Vector Machines