# Machine Learning (CSE 446): Perceptron

Sham M Kakade

© 2018

University of Washington
cse446-staff@cs.washington.edu

## Announcements

- HW due this week.
  See detailed instructions in the hw.
  - One pdf file.
  - Answers and figures grouped together for each problem (in order).
  - Submit your code (only for problem 4 needed for HW1).
- Qz section this week. a little probability + more linear algebra
- **Updated late policy:**
  - You get 2 late days for the entire quarter, which will be automatically deducted per late day (or part thereof).
  - After these days are used up, 33% deducted per day.
- Today: the perceptron algo

## Review

## The General Recipe (for supervised learning)

The cardinal rule of machine learning: **Don't touch your test data.**

If you follow that rule, this recipe will give you accurate information:

1. Split data into training, (maybe) development, and test sets.
2. For different hyperparameter settings:
   2.1 Train on the training data using those hyperparameter values.
   2.2 Evaluate loss on development data.
3. Choose the hyperparameter setting whose model achieved the lowest development data loss.
   Optionally, retrain on the training and development data together.
4. Now you have a hypothesis.
   Evaluate that model on test data.

## Also...

- **supervised learning** algorithms we covered:
  - Decision trees: uses few features, “selectively”
  - Nearest neighbor: uses all features, “blindly”
- **unsupervised learning** algorithms we covered:
  - K-means: a clustering algorithm
    we show it converges later in the class
  - this algorithm does not use labels

## Probability you should know

- expectations (and notation)
- mean, variance
- unbiased estimate
- joint distributions
- conditional distributions

## Linear algebra you should know

- vectors
- inner products (and interpretation)
- Euclidean distance. Euclidean norm.
- soon:
  - matrices and matrix multiplication
  - "outer" products
  - a covariance matrix
  - how to write a vector $x$ in an "orthogonal" basis.
  - SVD/eigenvectors/eigenvalues...

## Today

## Is there a happy medium?

Decision trees (that aren't too deep): use relatively few features to classify.

$K$-nearest neighbors: all features weighted equally.

Today: use all features, but weight them.

## Is there a happy medium?

Decision trees (that aren't too deep): use relatively few features to classify.

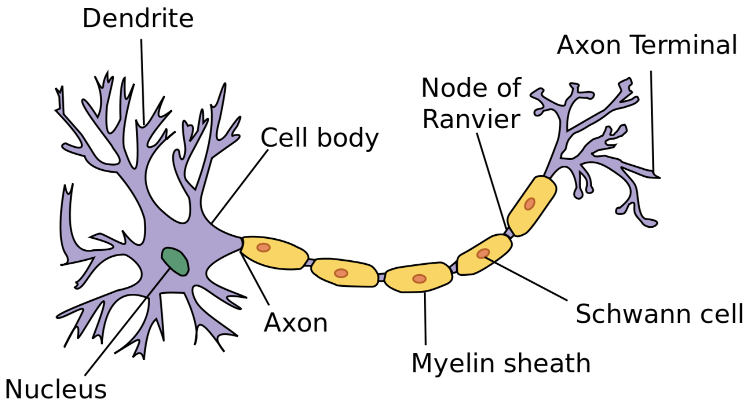
$K$-nearest neighbors: all features weighted equally.

Today: use all features, but weight them.

For today's lecture, assume that $\underbrace{y \in \{-1, +1\}}_{}$ instead of $\{0, 1\}$, and that $\mathbf{x} \in \mathbb{R}^d$.

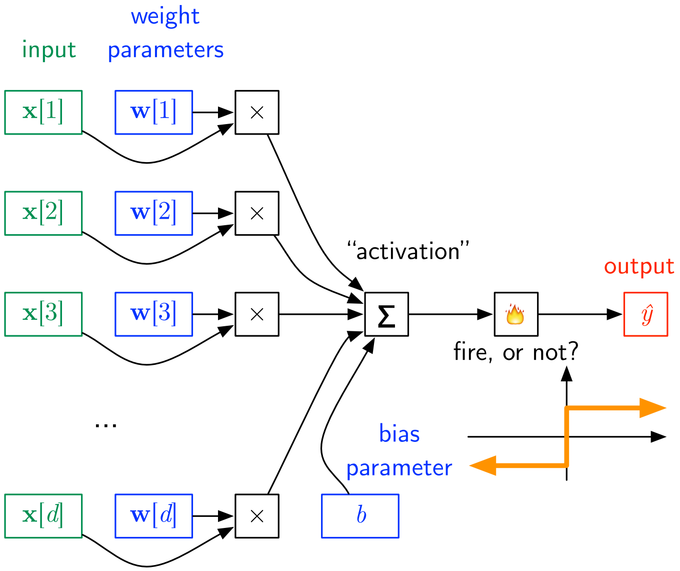
## Inspiration from Neurons

Image from Wikimedia Commons.

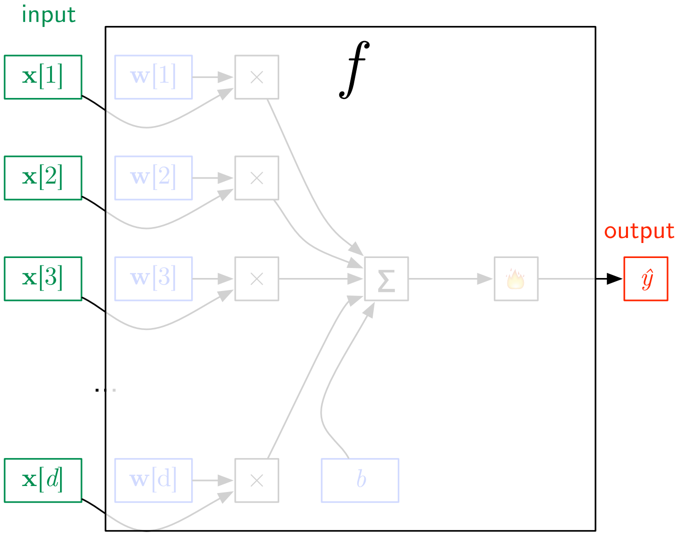


Input signals come in through dendrites, output signal passes out through the axon.

## Neuron-Inspired Classifier



## Neuron-Inspired Classifier



## A "parametric" Hypothesis

$$\text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z \leq 0 \end{cases}$$

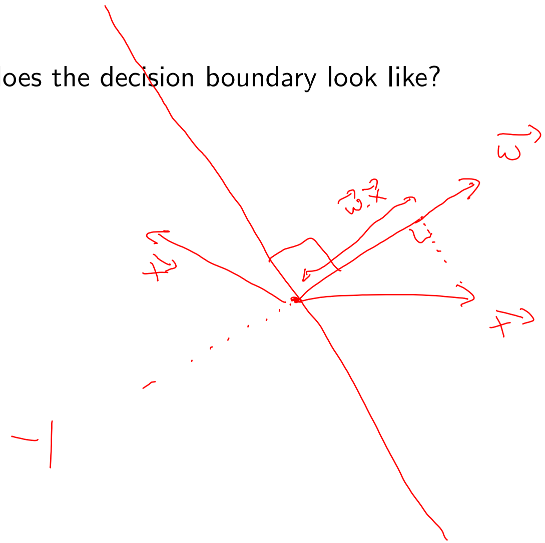
$$f(\mathbf{x}) = \text{sign}\left(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b\right)$$

remembering that: $\mathbf{w} \cdot \mathbf{x} = \sum_{j=1}^{d} \mathbf{w}[j] \cdot \mathbf{x}[j]$

Learning requires us to set the weights $\mathbf{w}$ and the bias $b$.

## Geometrically...

- What does the decision boundary look like?



assume
$b = 0$
$\|w\| = 1$

$+$ $)$

## "Online learning"

- Let's think of an **online** algorithm, where we try to update $w$ as we examine each training point $(x_i, y_i)$, one at a time.
- How should we change $w$ if we do not make an error on some point $(x, y)$?

  No

- How should we change $w$ if we make an error on some point $(x, \overset{+1}{y})$?

we predict $-1$, but truth is $y = +1$

$w \leftarrow w + x$

we predict $+1$, but $y = -1$

$w \leftarrow w - x$

$w \leftarrow w + yx$

## Perceptron Learning Algorithm

```
Data: $D = \langle(\mathbf{x}_n, y_n)\rangle_{n=1}^{N}$, number of epochs $E$
Result: weights $\mathbf{w}$ and bias $b$
initialize: $\mathbf{w} = \mathbf{0}$ and $b = 0$;
for $e \in \{1, \ldots, E\}$ do
    for $n \in \{1, \ldots, N\}$, in random order do
        # predict
        $\hat{y} = \text{sign}\left(\mathbf{w} \cdot \mathbf{x}_n + b\right)$;
        if $\hat{y} \neq y_n$ then
            # update
            $\mathbf{w} \leftarrow \mathbf{w} + y_n \cdot \mathbf{x}_n$;
            $b \leftarrow b + y_n$;
        end
    end
end
return $\mathbf{w}, b$
```

**Algorithm 1:** PerceptronTrain

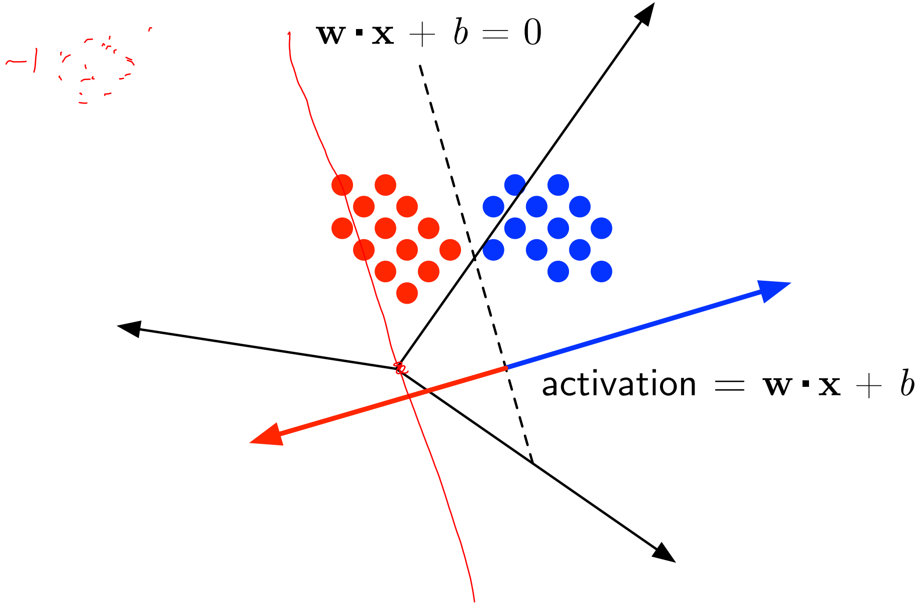
## Parameters and Hyperparameters

This is the first supervised algorithm we've seen that has **parameters** that are numerical values ($\mathbf{w}$ and $b$).

The perceptron learning algorithm's sole hyperparameter is $E$, the number of epochs (passes over the training data).

How should we tune $E$ using the development data?

## Linear Decision Boundary


$\mathbf{w} \cdot \mathbf{x} + b = 0$
activation $= \mathbf{w} \cdot \mathbf{x} + b$
$-1$

## Interpretation of Weight Values

What does it mean when . . .

- $w_1$ $\cancel{w_{12}} = 100?$
- $w_2$ $\cancel{w_{12}} = -1?$
- $w_3$ $\cancel{w_{12}} = 0?$

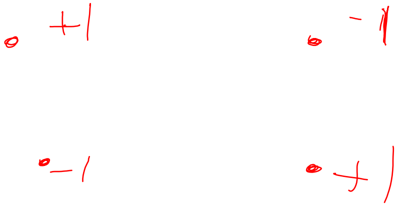
What about feature scaling?

## What can we say about convergence?

- Can you say what has to occur for the algorithm to converge?

happen to converge $\Longrightarrow$ data seperable

- Can we understand when it will *never* converge?



~~not~~ linearly seperable

**Stay tuned...**