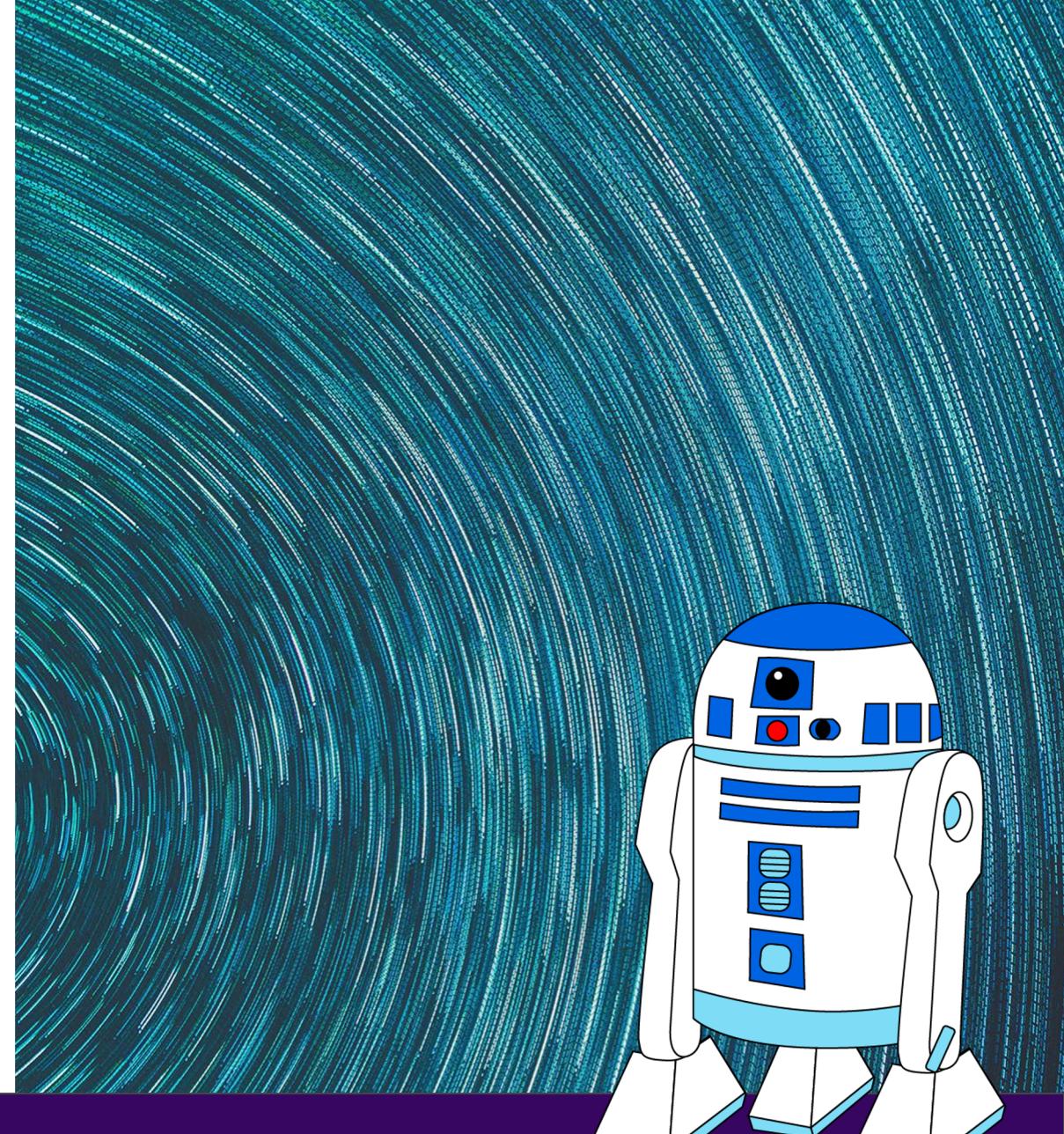


CIS 421/521:  
ARTIFICIAL INTELLIGENCE

# Machine Learning

Read AIMA Chapter 19.1-19.6



# Machine Learning

Up until now: how use a model to make optimal decisions

Machine learning: how to acquire a model from data / experience

- Learning parameters (e.g. probabilities)
- Learning structure (e.g. BN graphs)
- Learning hidden concepts (e.g. clustering)

Today: model-based classification with Naive Bayes and Perceptrons

# Spam Classification

Input: an email

Output: spam/ham

Setup:

- Get a large collection of example emails, each labeled "spam" or "ham"
- Note: someone has to hand label all this data!
- Want to learn to predict labels of new, future emails

Features: The attributes used to make the ham / spam decision

- Words: FREE!
- Text Patterns: \$dd, CAPS
- Non-text: SenderInContacts
- ...



Dear Sir.

First, I must solicit your confidence in this transaction, this is by virtue of its nature as being utterly confidential and top secret. ...

TO BE REMOVED FROM FUTURE MAILINGS, SIMPLY REPLY TO THIS MESSAGE AND PUT "REMOVE" IN THE SUBJECT.

99 MILLION EMAIL ADDRESSES FOR ONLY \$99

Ok, I know this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

# Digit Recognition

Input: images / pixel grids

Output: a digit 0-9

Setup:

- Get a large collection of example images, each labeled with a digit
- Note: someone has to hand label all this data!
- Want to learn to predict labels of new, future digit images

Features: The attributes used to make the digit decision

- Pixels:  $(6,8)=\text{ON}$
- Shape Patterns: NumComponents, AspectRatio, NumLoops
- ...



0



1



2



1



??

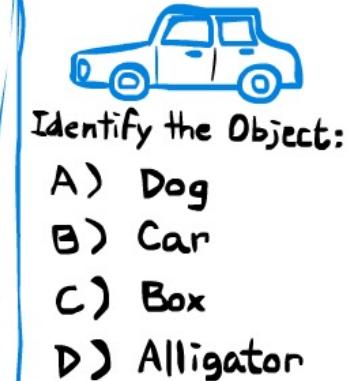
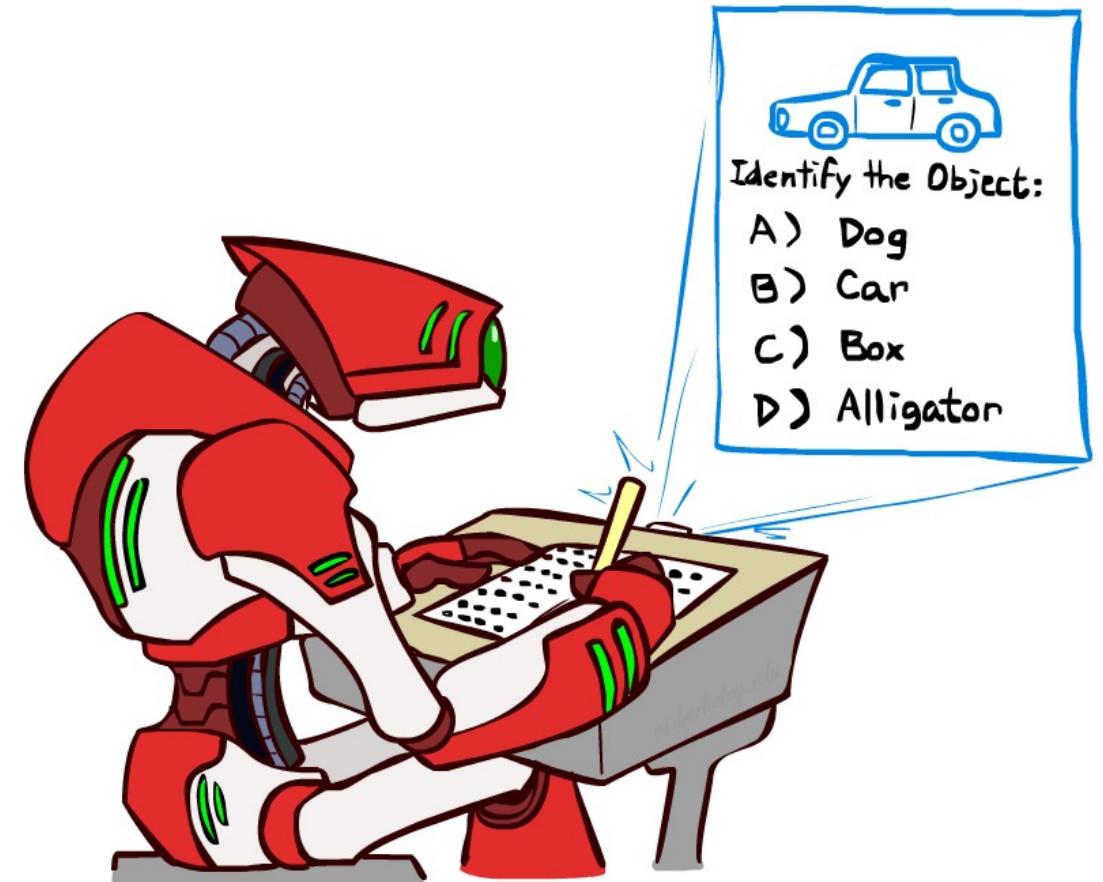
# Review Other Classification Tasks

Classification: given inputs  $x$ , predict labels  $y$

Examples:

- Spam detection (input: document, classes: spam / ham)
- OCR (input: images, classes: characters)
- Medical diagnosis (input: symptoms, classes: diseases)
- Automatic essay grading (input: document, classes: grades)
- Fraud detection (input: account activity, classes: fraud / no fraud)
- Customer service email routing
- ... many more

Classification is an important commercial technology!



Identify the Object:  
A) Dog  
B) Car  
C) Box  
D) Alligator

# Model-Based Classification

## Model-based approach

- Build a model (e.g. Bayes' net) where both the label and features are random variables
- Instantiate any observed features
- Query for the distribution of the label conditioned on the features

## Challenges

- What structure should the BN have?
- How should we learn its parameters?

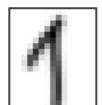


# Naïve Bayes for Digits

Naïve Bayes: Assume all features are independent effects of the label

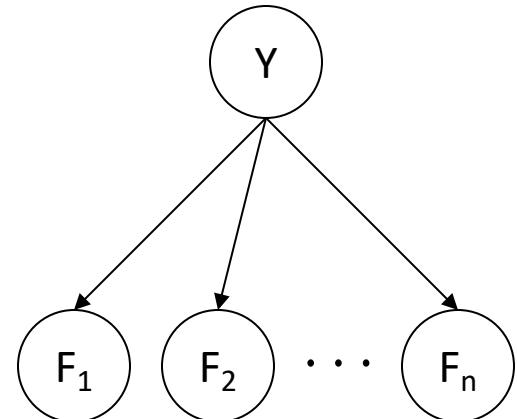
Simple digit recognition version:

- One feature (variable)  $F_{ij}$  for each grid position  $\langle i,j \rangle$
- Feature values are on / off, based on whether intensity is more or less than 0.5 in underlying image
- Each input maps to a feature vector, e.g.



$\rightarrow \langle F_{0,0} = 0 \ F_{0,1} = 0 \ F_{0,2} = 1 \ F_{0,3} = 1 \ F_{0,4} = 0 \ \dots \ F_{15,15} = 0 \rangle$

- Here: lots of features, each is binary valued



Naïve Bayes model:

What do we need to learn?  $P(Y|F_{0,0} \dots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$

# General Naïve Bayes

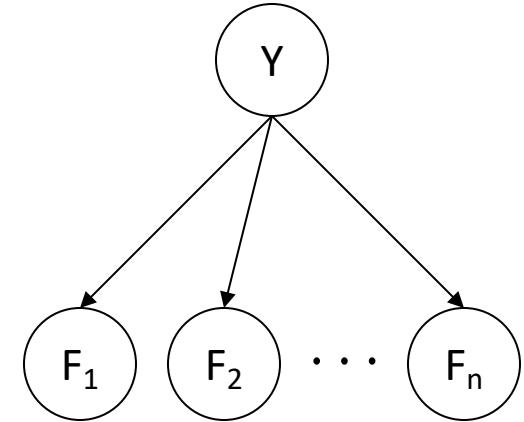
A general Naive Bayes model:

$|Y|$  labels

$$P(Y, F_1 \dots F_n) = P(Y) \prod_i P(F_i | Y)$$

$|Y| \times |F|^n$  values

$n \times |F| \times |Y|$   
parameters



We only have to specify how each feature depends on the class

Total number of parameters is *linear* in number of features

Model is very simplistic, but often works anyway

# Inference for Naïve Bayes

Goal: compute posterior distribution over label variable  $Y$

- Step 1: get joint probability of label and evidence for each label

$$P(Y, f_1 \dots f_n) = \begin{bmatrix} P(y_1, f_1 \dots f_n) \\ P(y_2, f_1 \dots f_n) \\ \vdots \\ P(y_k, f_1 \dots f_n) \end{bmatrix} \rightarrow \frac{\begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix}}{P(f_1 \dots f_n)}$$

- Step 2: sum to get probability of evidence
- Step 3: normalize by dividing Step 1 by Step 2

$$P(Y|f_1 \dots f_n)$$

# General Naïve Bayes

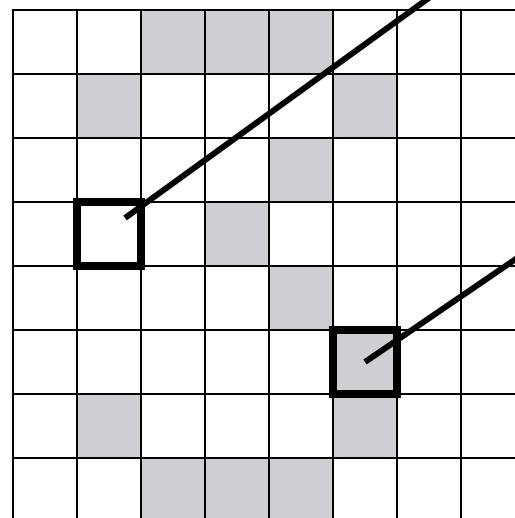
What do we need in order to use Naïve Bayes?

- Inference method (we just saw this part)
  - Start with a bunch of probabilities:  $P(Y)$  and the  $P(F_i | Y)$  tables
  - Use standard inference to compute  $P(Y | F_1 \dots F_n)$
  - Nothing new here
- Estimates of local conditional probability tables
  - $P(Y)$ , the prior over labels
  - $P(F_i | Y)$  for each feature (evidence variable)
  - These probabilities are collectively called the *parameters* of the model and denoted by  $\theta$
  - Up until now, we assumed these appeared by magic, but...
  - ...they typically come from training data counts: we'll look at this soon

# Example: Conditional Probabilities

$P(Y)$

1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1



$P(F_{3,1} = on|Y) \quad P(F_{5,5} = on|Y)$

1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

# Naïve Bayes for Text

Bag-of-words Naïve Bayes:

- Features:  $W_i$  is the word at position i
- As before: predict label conditioned on feature variables (spam vs. ham)
- As before: assume features are conditionally independent given label
- New: each  $W_i$  is identically distributed

Generative model:

$$P(Y, W_1 \dots W_n) = P(Y) \prod_i P(W_i|Y)$$

*Word at position  
i, not  $i^{\text{th}}$  word in  
the dictionary!*

“Tied” distributions and bag-of-words

- Usually, each variable gets its own conditional probability distribution  $P(F|Y)$
- In a bag-of-words model
  - Each position is identically distributed
  - All positions share the same conditional probs  $P(W|Y)$
  - Why make this assumption?
- Called “bag-of-words” because model is insensitive to word order or reordering

# Example: Spam Filtering

Model:  $P(Y, W_1 \dots W_n) = P(Y) \prod_i P(W_i|Y)$

What are the parameters?

$P(Y)$

ham : 0.66
spam: 0.33

$P(W|\text{spam})$

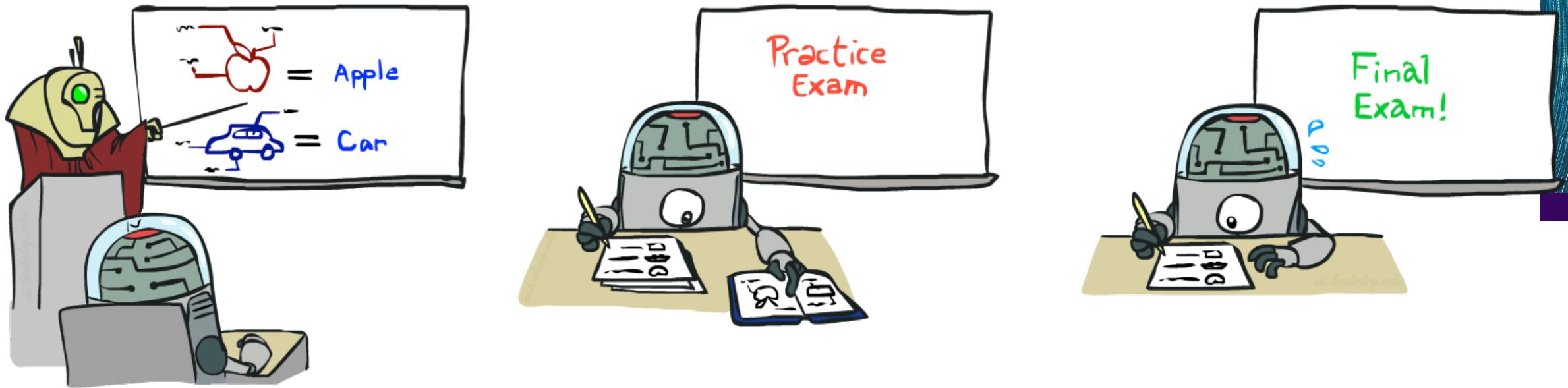
the : 0.0156
to : 0.0153
and : 0.0115
of : 0.0095
you : 0.0093
a : 0.0086
with: 0.0080
from: 0.0075
...

$P(W|\text{ham})$

the : 0.0210
to : 0.0133
of : 0.0119
2002: 0.0110
with: 0.0108
from: 0.0107
and : 0.0105
a : 0.0100
...

Where do these tables come from?

# Training and Testing



# Important Concepts

Data: labeled instances, e.g. emails marked spam/ham

- Training set
- Held out set
- Test set

Features: attribute-value pairs which characterize each  $x$

Experimentation cycle

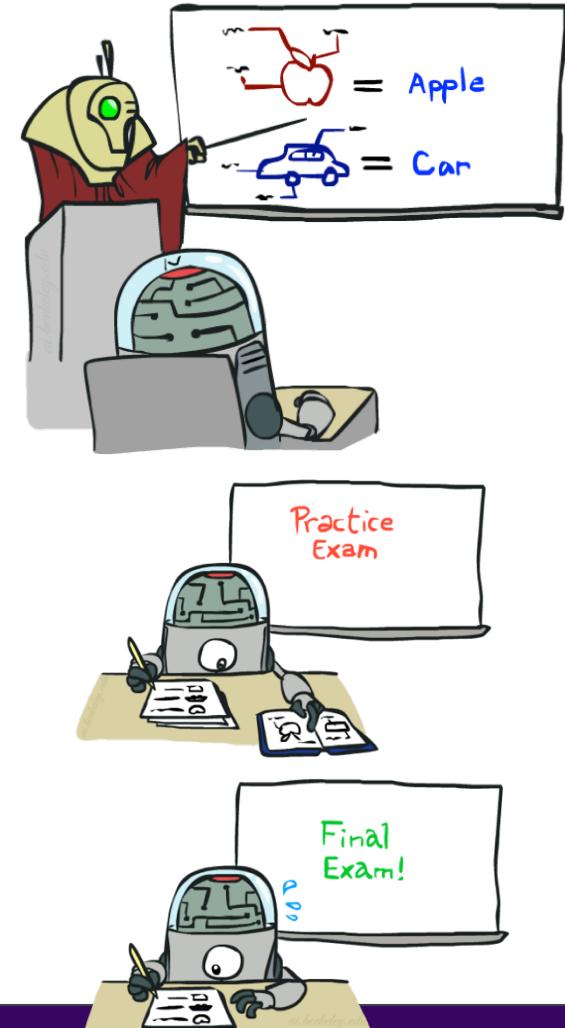
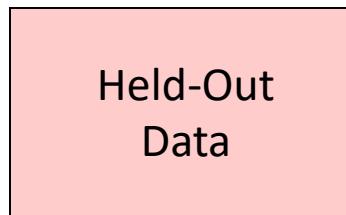
- Learn parameters (e.g. model probabilities) on training set
- (Tune hyperparameters on held-out set)
- Compute accuracy of test set
- Very important: never “peek” at the test set!

Evaluation

- Accuracy: fraction of instances predicted correctly

Overfitting and generalization

- Want a classifier which does well on *test* data
- Overfitting: fitting the training data very closely, but not generalizing well
- We'll investigate overfitting and generalization formally in a few lectures



# Example: Overfitting

$P(\text{features}, C = 2)$

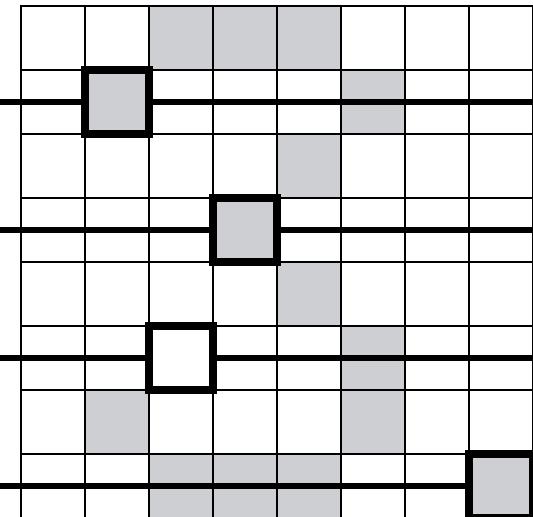
$P(C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.8$

$P(\text{on}|C = 2) = 0.1$

$P(\text{off}|C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.01$



$P(\text{features}, C = 3)$

$P(C = 3) = 0.1$

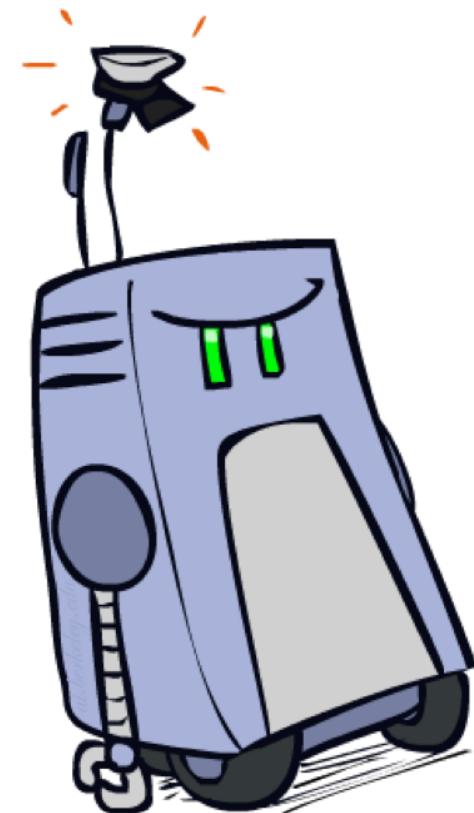
$P(\text{on}|C = 3) = 0.8$

$P(\text{on}|C = 3) = 0.9$

$P(\text{off}|C = 3) = 0.7$

$P(\text{on}|C = 3) = 0.0$

*2 wins!!*



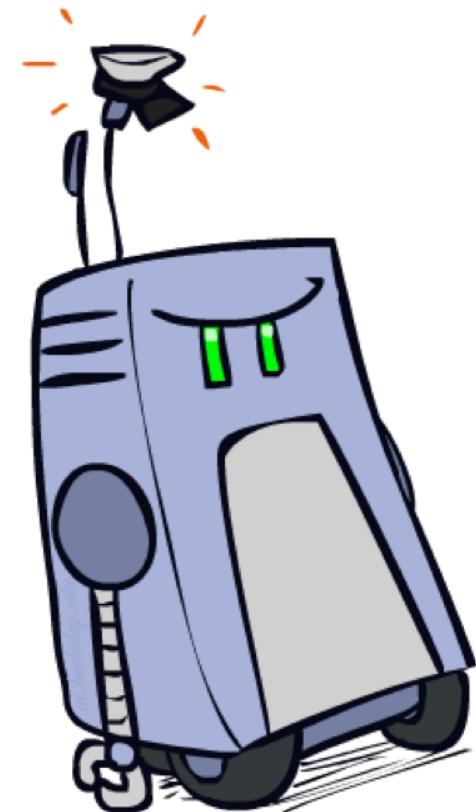
# Example: Overfitting

Posteriors determined by *relative* probabilities (odds ratios):

$$\frac{P(W|\text{spam})}{P(W|\text{ham})}$$

```
screens      : inf
minute       : inf
guaranteed   : inf
$205.00     : inf
delivery     : inf
signature    : inf
...
```

*What went wrong here?*



# Generalization and Overfitting

Relative frequency parameters will **overfit** the training data!

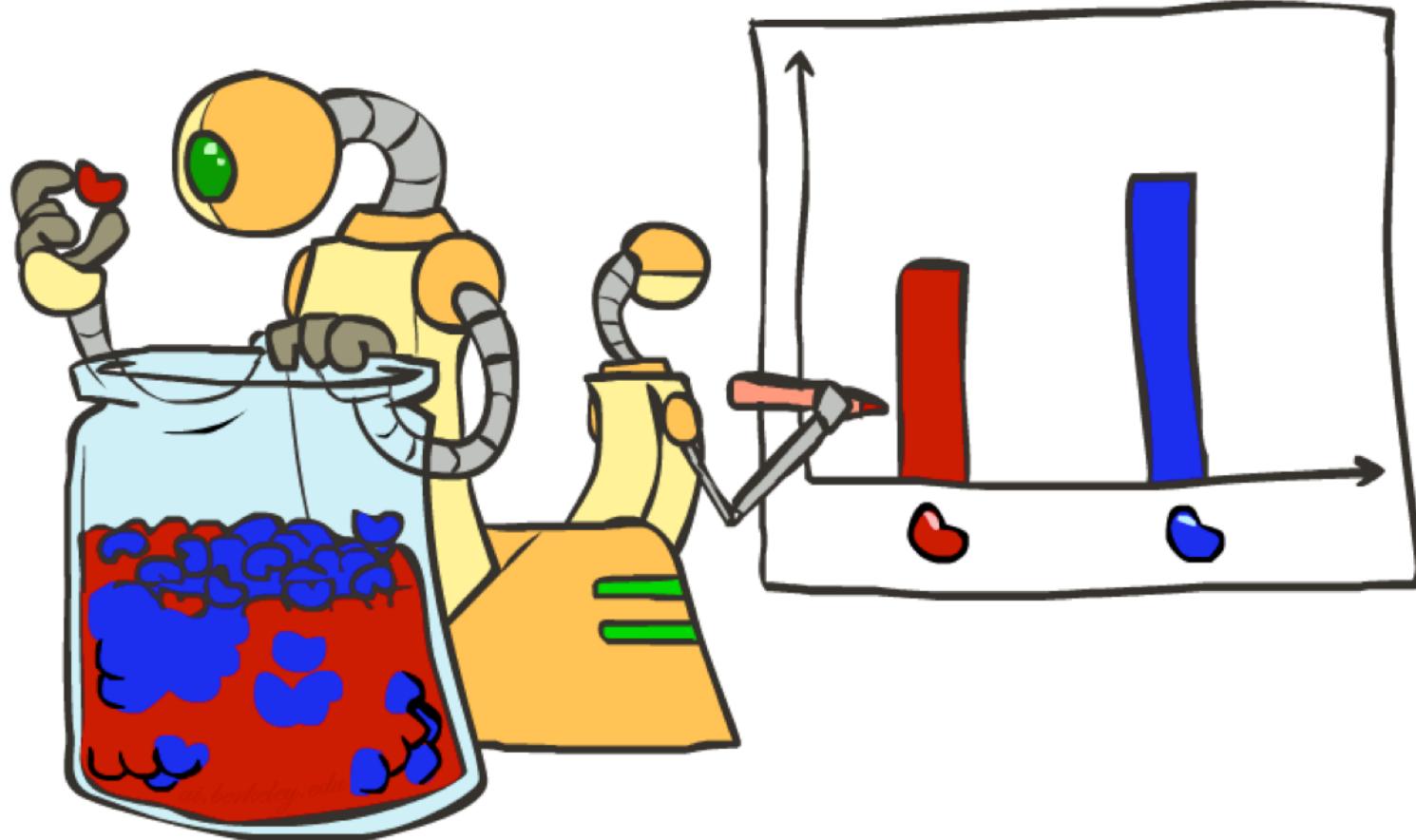
- Just because we never saw a 3 with pixel (15,15) on during training doesn't mean we won't see it at test time
- Unlikely that every occurrence of "minute" is 100% spam
- Unlikely that every occurrence of "seriously" is 100% ham
- What about all the words that don't occur in the training set at all?
- In general, we can't go around giving unseen events zero probability

As an extreme case, imagine using the entire email as the only feature

- Would get the training data perfect (if deterministic labeling)
- Wouldn't *generalize* at all
- Just making the bag-of-words assumption gives us some generalization, but isn't enough

To generalize better: we need to **smooth** or **regularize** the estimates

# Parameter Estimation



# Parameter Estimation

Estimating the distribution of a random variable

*Elicitation:* ask a human (why is this hard?)

*Empirically:* use training data (learning!)

- E.g.: for each outcome  $x$ , look at the *empirical rate* of that val

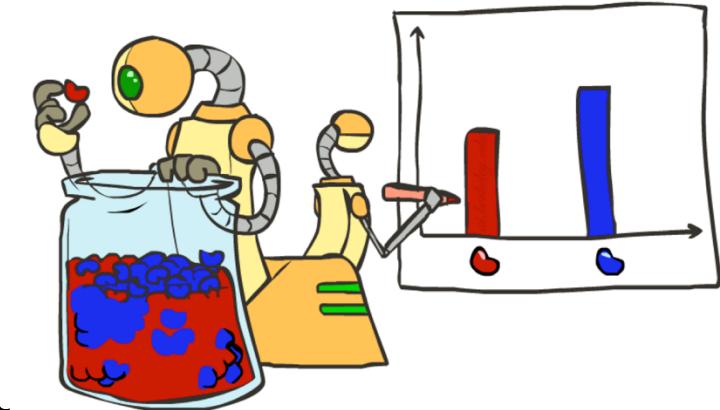
$$P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

r      r      b

$$P_{\text{ML}}(\text{r}) = 2/3$$

- This is the estimate that maximizes the *likelihood of the data*

$$L(x, \theta) = \prod_i P_\theta(x_i)$$



# Maximum Likelihood

Relative frequencies are the maximum likelihood estimates

$$\begin{aligned}\theta_{ML} &= \arg \max_{\theta} P(\mathbf{X}|\theta) \\ &= \arg \max_{\theta} \prod_i P_{\theta}(X_i)\end{aligned}\quad \rightarrow \quad P_{ML}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

# Laplace Smoothing

Laplace's estimate:

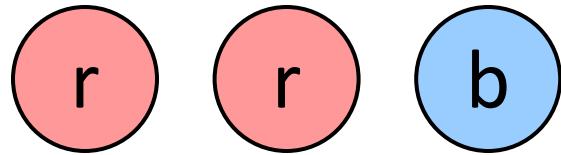
- Pretend you saw every outcome once more than you actually did

$$P_{LAP}(x) = \frac{c(x) + 1}{\sum_x [c(x) + 1]}$$

$$= \frac{c(x) + 1}{N + |X|}$$

$$P_{ML}(X) =$$

$$P_{LAP}(X) =$$



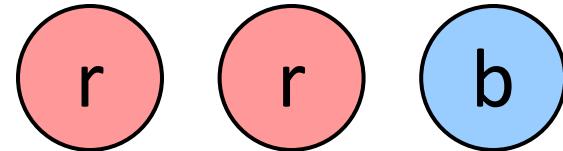
# Laplace Smoothing

Laplace's estimate (extended):

- Pretend you saw every outcome k extra times

$$P_{LAP,k}(x) = \frac{c(x) + k}{N + k|X|}$$

- What's Laplace with k = 0?
- k is the **strength** of the prior



$$P_{LAP,0}(X) =$$

$$P_{LAP,1}(X) =$$

Laplace for conditionals:

- Smooth each condition independently:

$$P_{LAP,k}(x|y) = \frac{c(x,y) + k}{c(y) + k|X|}$$

$$P_{LAP,100}(X) =$$

# Estimation: Linear Interpolation

Another option: linear interpolation

- Also get the empirical  $P(X)$  from the data
- Make sure the estimate of  $P(X | Y)$  isn't too different from the empirical  $P(X)$

$$P_{LIN}(x|y) = \alpha \hat{P}(x|y) + (1.0 - \alpha) \hat{P}(x)$$

- What if  $\alpha$  is 0? 1?

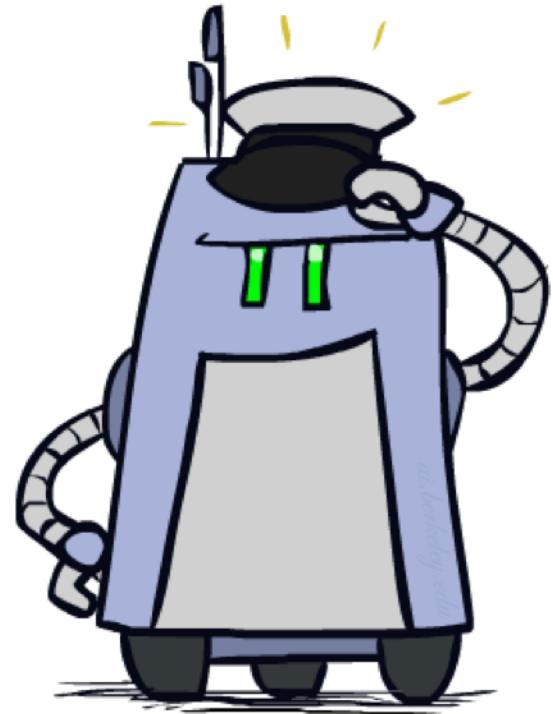
# Real NB: Smoothing

For real classification problems, smoothing is critical

New odds ratios:

$$\frac{P(W|\text{spam})}{P(W|\text{ham})}$$

Credit	:	28.4
ORDER	:	27.2
<FONT>	:	26.9
money	:	26.5
...		



*Do these make more sense?*

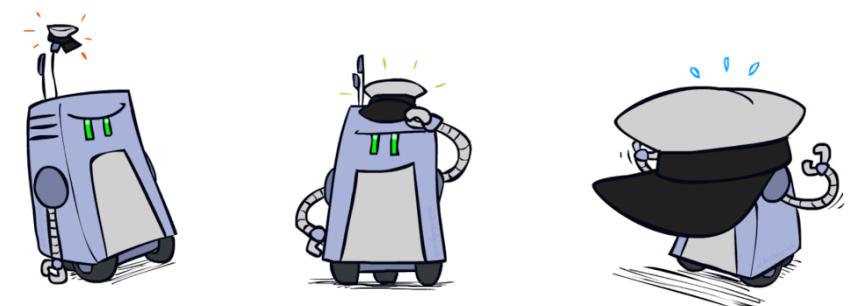
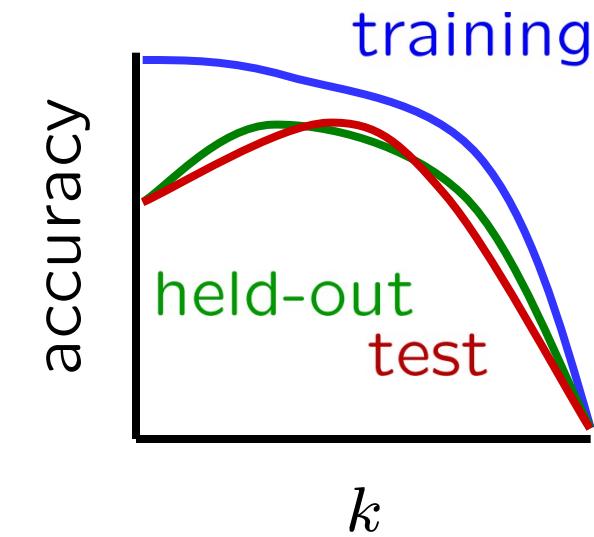
# Tuning on Held-Out Data

Now we've got two kinds of unknowns

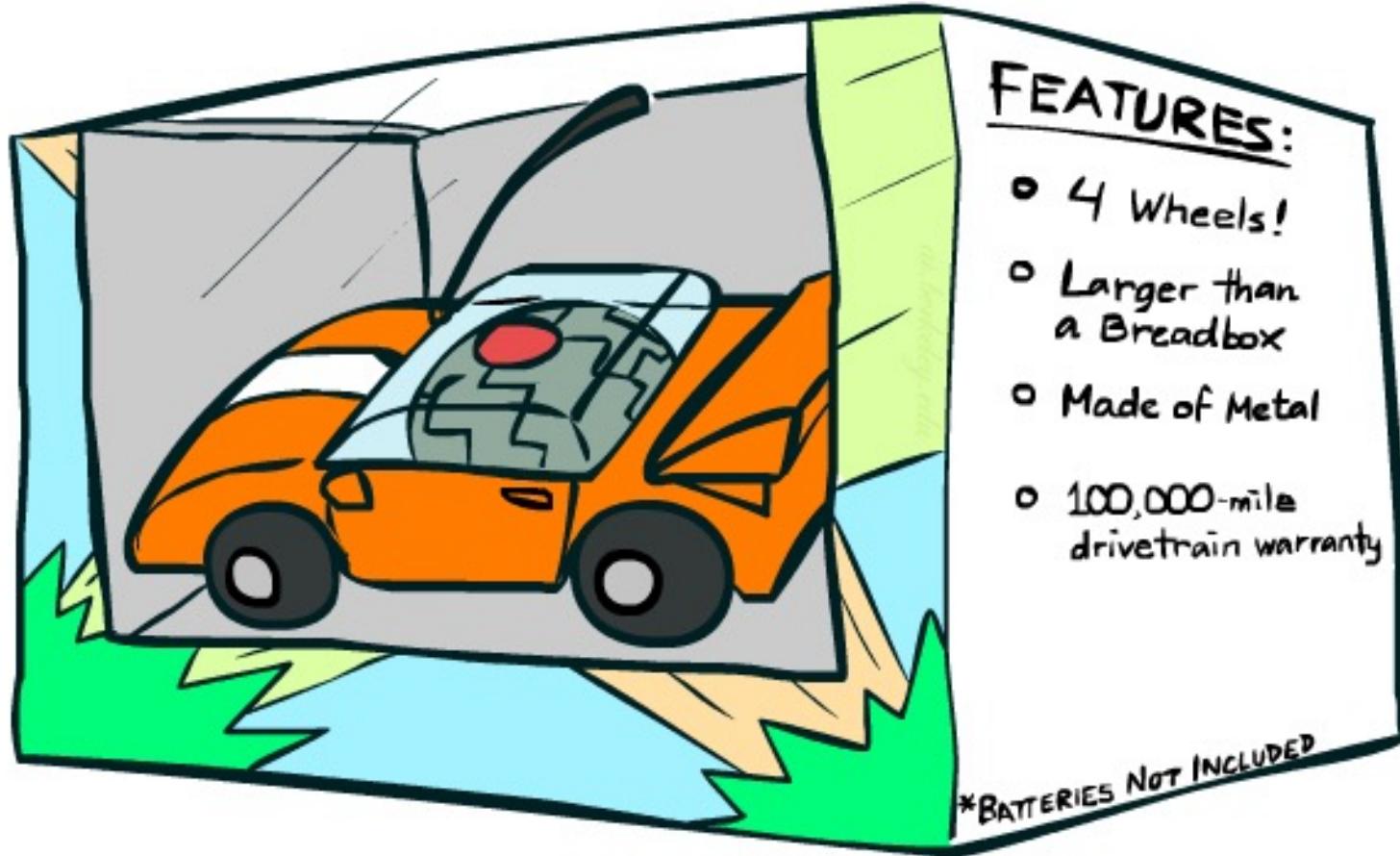
- Parameters: the probabilities  $P(X|Y)$ ,  $P(Y)$
- Hyperparameters: e.g. the amount / type of smoothing to do,  $k$ ,  $\alpha$

What should we learn where?

- Learn parameters from training data
- Tune hyperparameters on different data
  - Why?
- For each value of the hyperparameters, train and test on the held-out data
- Choose the best value and do a final test on the test data



# Features



# Errors, and What to Do

## Examples of errors

Dear GlobalSCAPE Customer,

GlobalSCAPE has partnered with ScanSoft to offer you the latest version of OmniPage Pro, for just \$99.99\* - the regular list price is \$499! The most common question we've received about this offer is - Is this genuine? We would like to assure you that this offer is authorized by ScanSoft, is genuine and valid. You can get the . . .

. . . To receive your \$30 Amazon.com promotional certificate, click through to

<http://www.amazon.com/apparel>

and see the prominent link for the \$30 offer. All details are there. We hope you enjoyed receiving this message. However, if you'd rather not receive future e-mails announcing new store launches, please click . . .

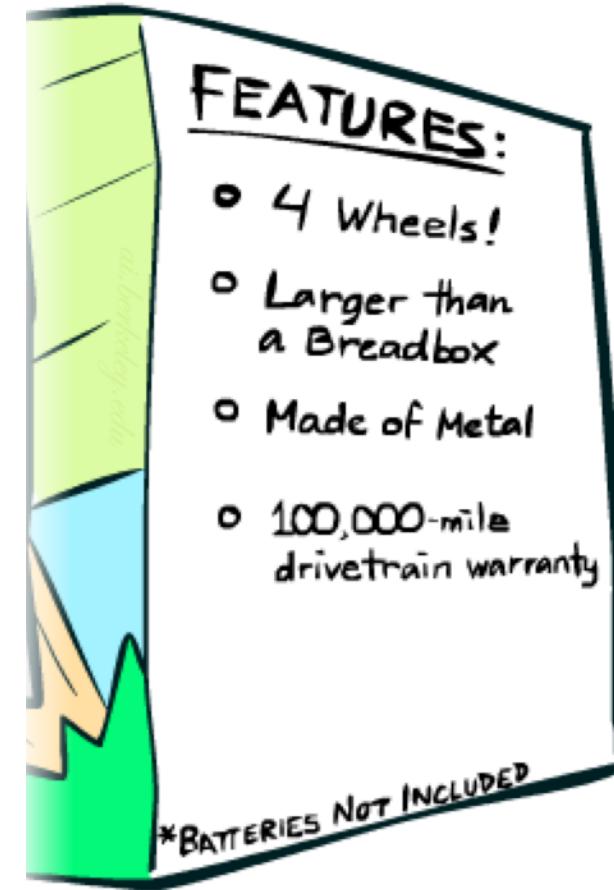
# What to Do About Errors?

Need more features— words aren't enough!

- Have you emailed the sender before?
- Have 1K other people just gotten the same email?
- Is the sending information consistent?
- Is the email in ALL CAPS?
- Do inline URLs point where they say they point?
- Does the email address you by (your) name?

Can add these information sources as new variables in the NB model

Next class we'll talk about classifiers which let you easily add arbitrary features more easily



# Baselines

## First step: get a **baseline**

- Baselines are very simple “straw man” procedures
- Help determine how hard the task is
- Help know what a “good” accuracy is

### Weak baseline: most frequent label classifier

- Gives all test instances whatever label was most common in the training set
- E.g. for spam filtering, might label everything as ham
- Accuracy might be very high if the problem is skewed
- E.g. calling everything “ham” gets 66%, so a classifier that gets 70% isn’t very good...

For real research, usually use previous work as a (strong) baseline

# Confidences from a Classifier

The **confidence** of a probabilistic classifier:

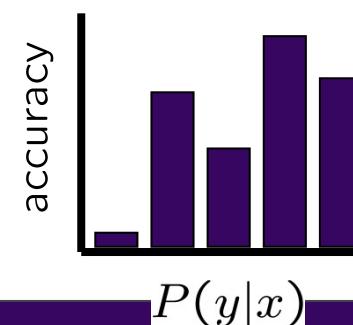
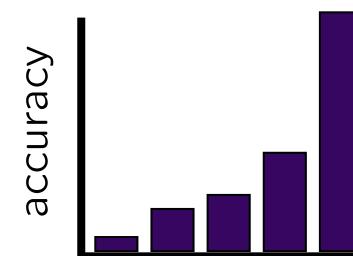
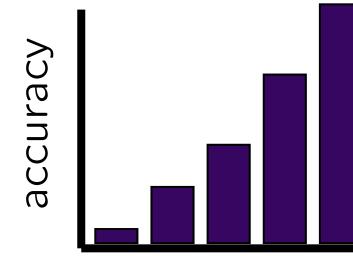
- Posterior over the top label

$$\text{confidence}(x) = \max_y P(y|x)$$

- Represents how sure the classifier is of the classification
- Any probabilistic model will have confidences
- No guarantee confidence is correct

## Calibration

- Weak calibration: higher confidences mean higher accuracy
- Strong calibration: confidence predicts accuracy rate
- What's the value of calibration?



# Summary

Bayes rule lets us do diagnostic queries with causal probabilities

The naïve Bayes assumption takes all features to be independent given the class label

We can build classifiers out of a naïve Bayes model using training data

Smoothing estimates is important in real systems

Classifier confidences are useful, when you can get them

# What to Do About Errors

Problem: there's still spam in your inbox

Need more **features** – words aren't enough!

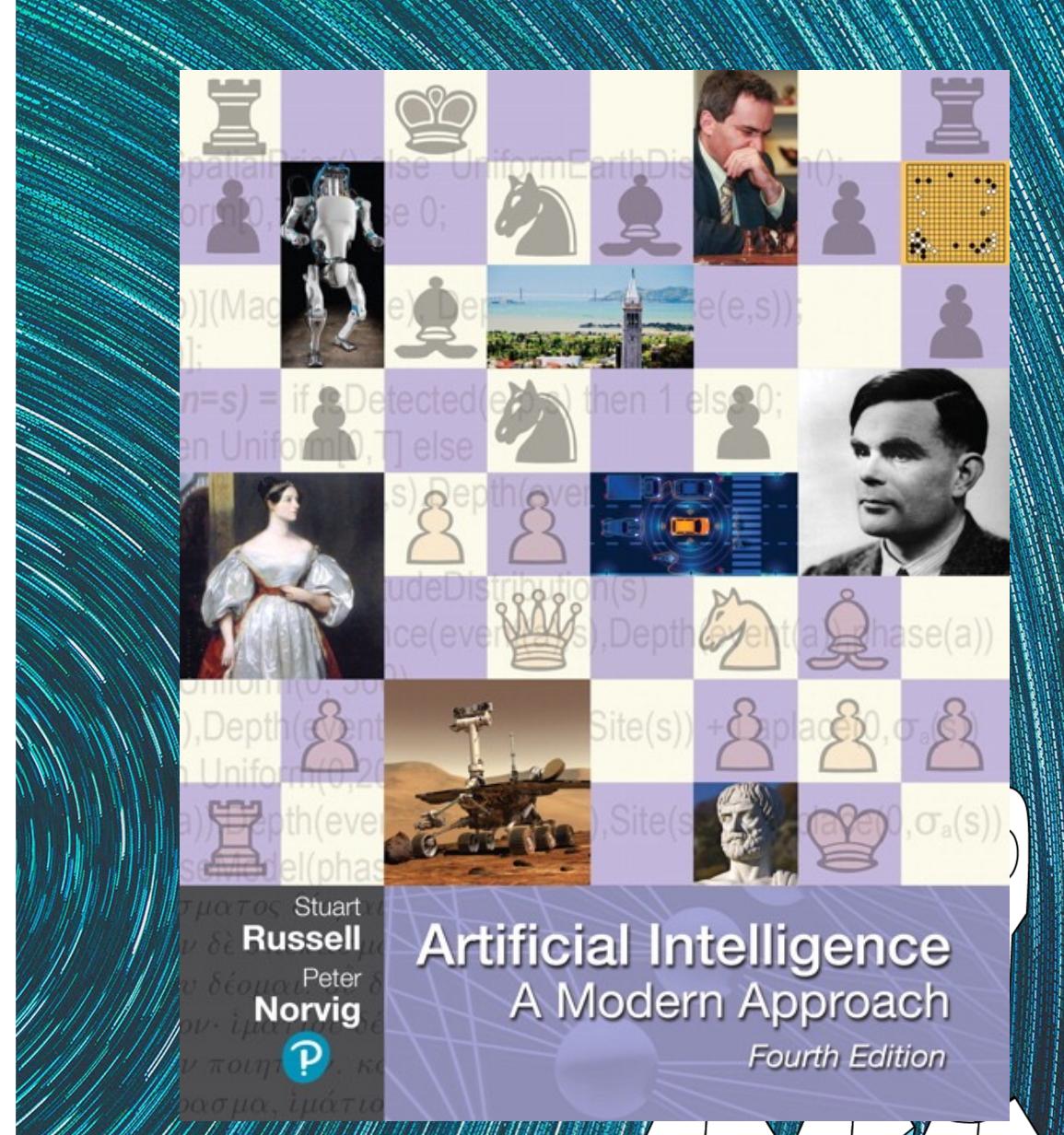
- Have you emailed the sender before?
- Have 1M other people just gotten the same email?
- Is the sending information consistent?
- Is the email in ALL CAPS?
- Do inline URLs point where they say they point?
- Does the email address you by (your) name?

Naïve Bayes models can incorporate a variety of features, but tend to do best in homogeneous cases (e.g. all features are word occurrences)

# CIS 421/521: ARTIFICIAL INTELLIGENCE

# Perceptrons

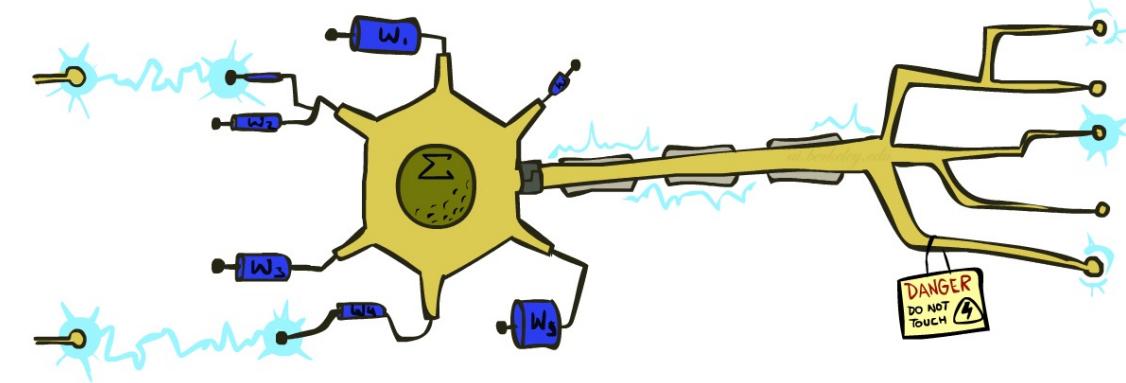
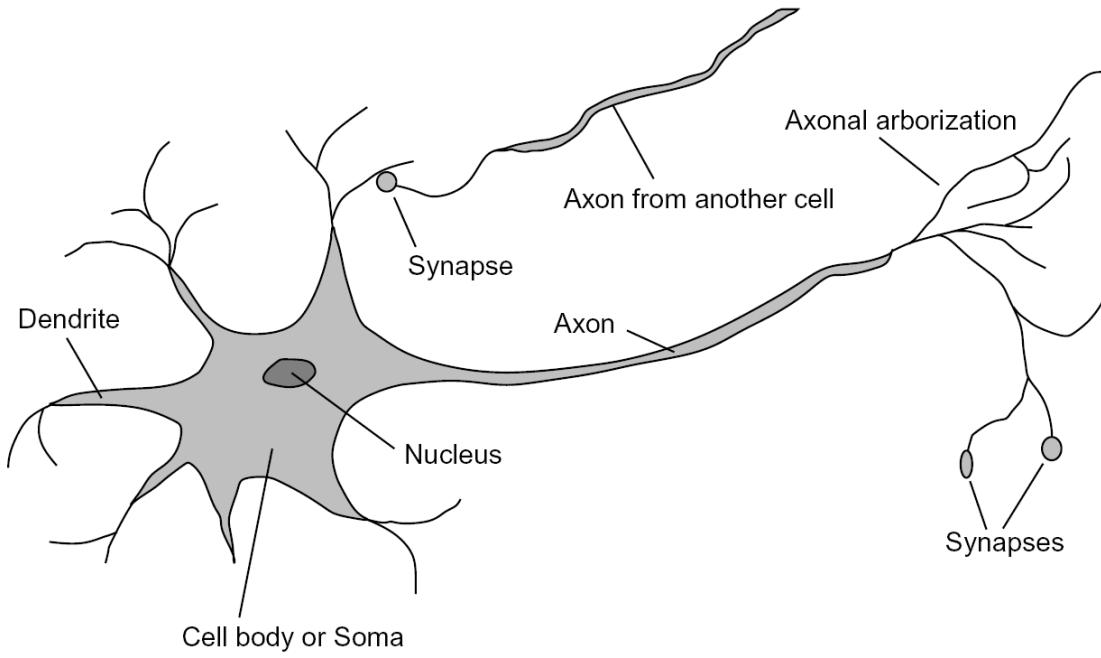
# Optional Reading: Chapter 1 Of Nielsen's "Neural Networks and Deep Learning"



Sides Courtesy of Dan Klein and Pieter Abbeel --- University of California, Berkeley  
Engineering  
The slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at <http://ai.berkeley.edu>.

# Some (Simplified) Biology

Very loose inspiration: human neurons

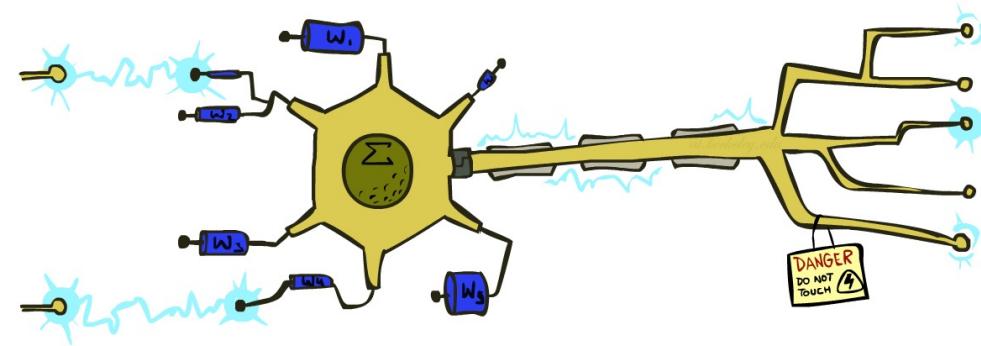


# Linear Classifiers

Inputs are **feature values**

Each feature has a **weight**

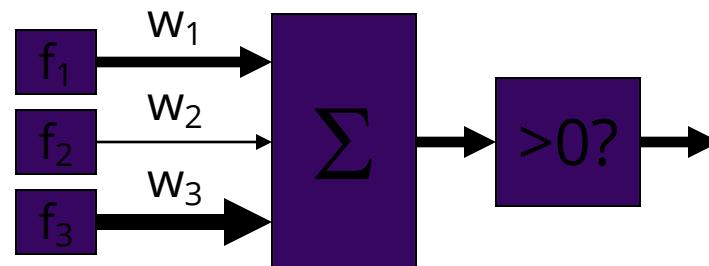
Sum is the **activation**



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

If the activation is:

- Positive, output +1
- Negative, output -1



# Feature Vectors

$x$

```
Hello,  
  
Do you want free printr  
cartridges? Why pay more  
when you can get them  
ABSOLUTELY FREE! Just
```

$f(x)$

$y$

SPAM  
or  
+

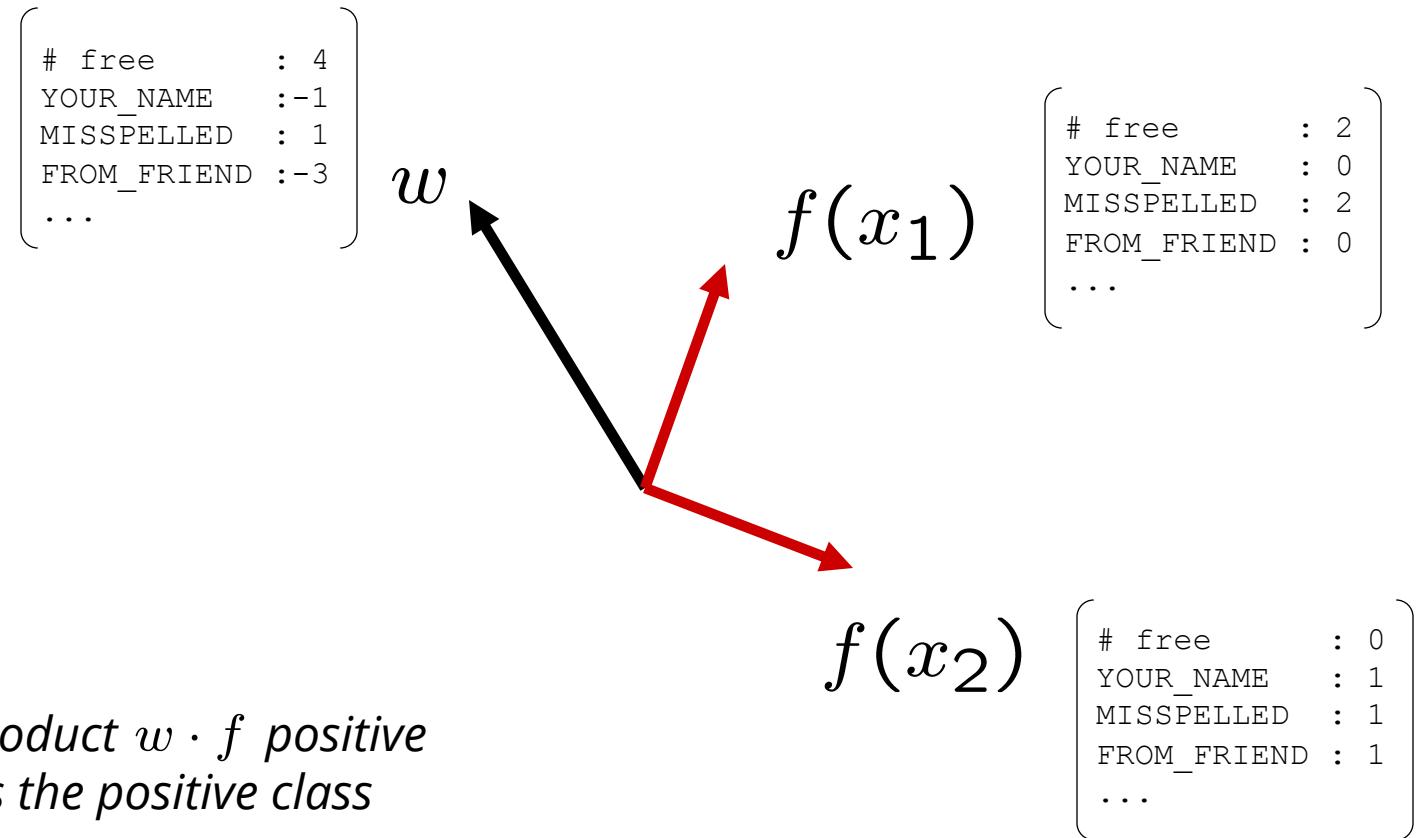

$$\begin{cases} \text{PIXEL-7,12} : 1 \\ \text{PIXEL-7,13} : 0 \\ \dots \\ \text{NUM_LOOPS} : 1 \\ \dots \end{cases}$$

“2”

# Weights

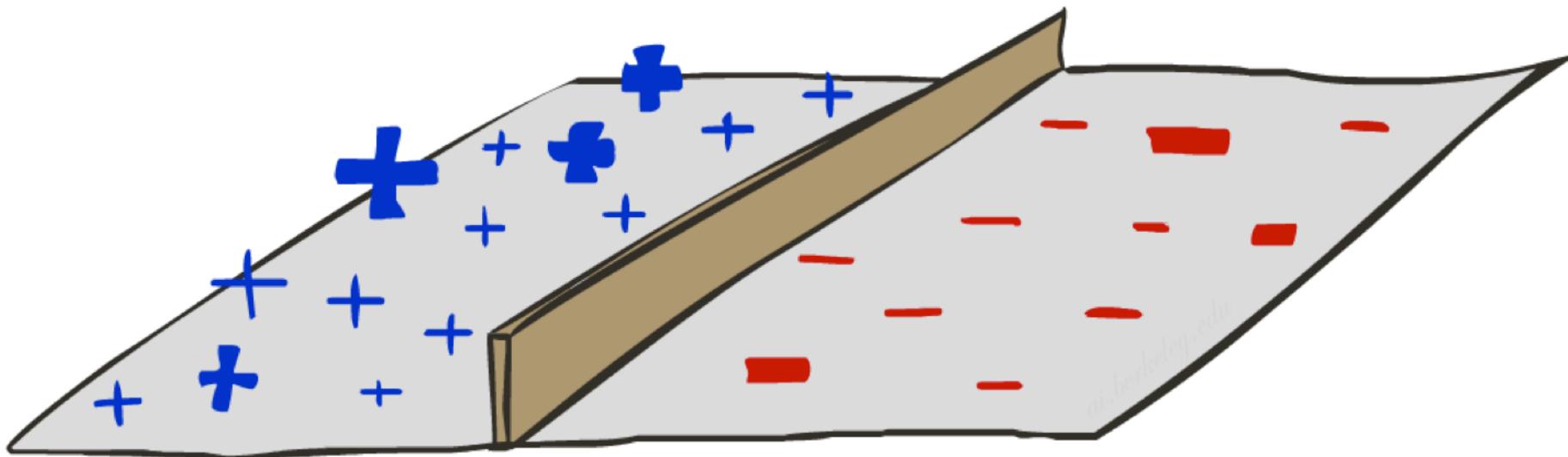
Binary case: compare features to a weight vector

Learning: figure out the weight vector from examples



*Dot product  $w \cdot f$  positive  
means the positive class*

# Decision Rules

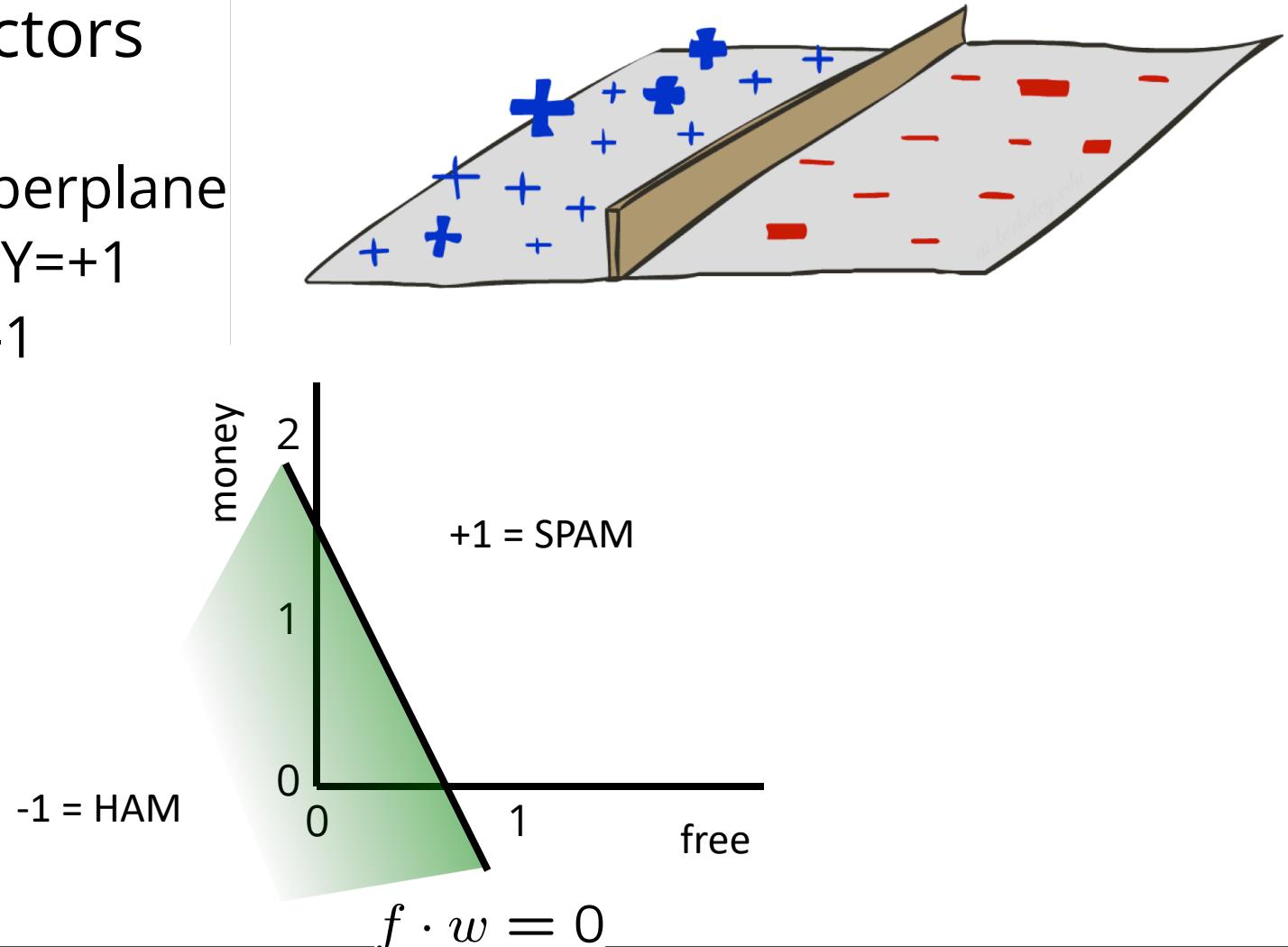


# Binary Decision Rule

In the space of feature vectors

- Examples are points
- Any weight vector is a hyperplane
- One side corresponds to  $Y=+1$
- Other corresponds to  $Y=-1$

$w$	
BIAS	: -3
free	: 4
money	: 2
...	



# Weight Updates

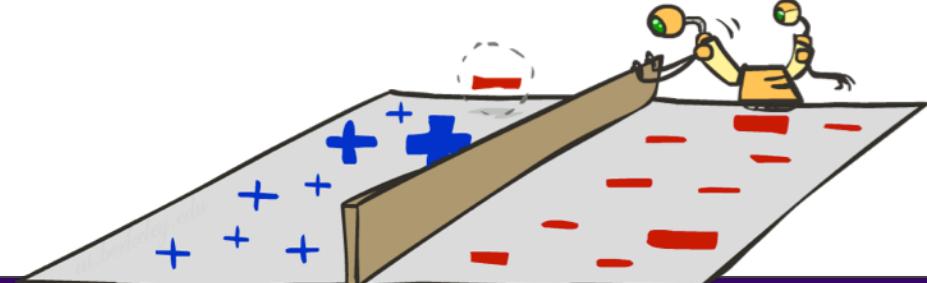
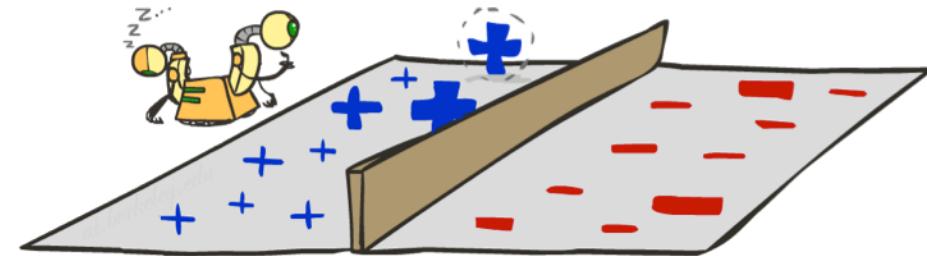
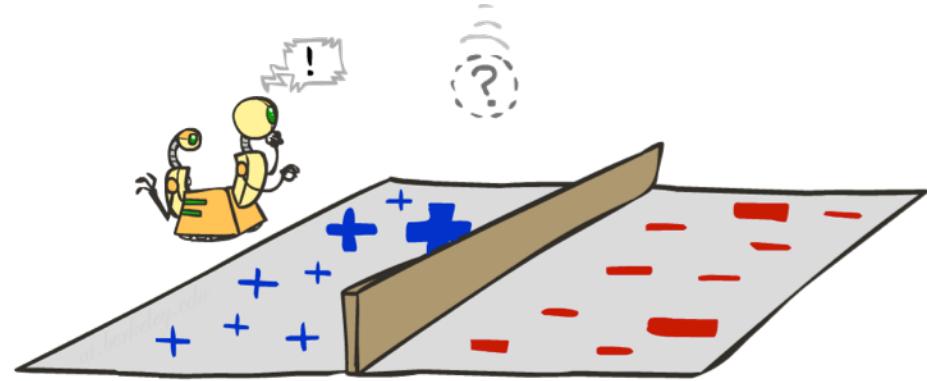


# Learning: Binary Perceptron

Start with weights = 0

For each training instance:

- Classify with current weights
- If correct (i.e.,  $y=y^*$ ), no change!
- If wrong: adjust the weight vector



# Learning: Binary Perceptron

Start with weights = 0

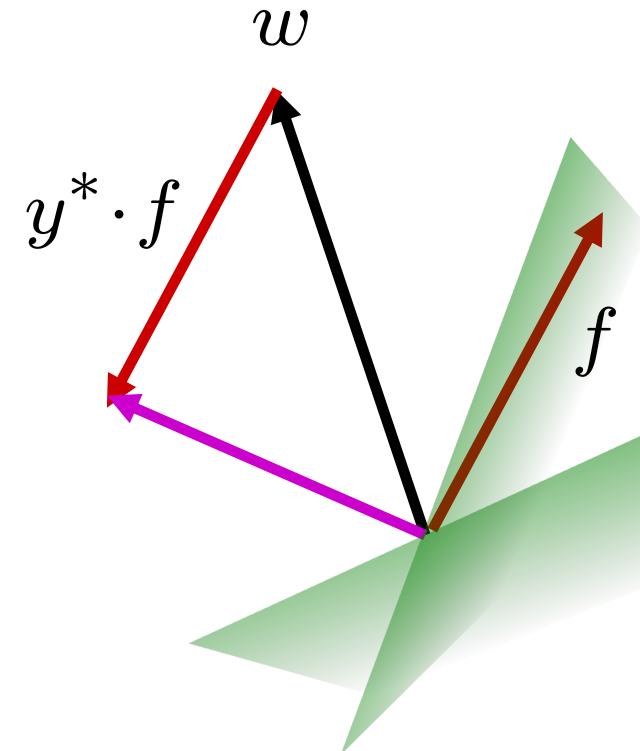
For each training instance:

- Classify with current weights

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

- If correct (i.e.,  $y=y^*$ ), no change!
- If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if  $y^*$  is -1.

$$w = w + y^* \cdot f$$



# 3Blue1Brown

Need a refresher on linear algebra topics like vector addition and matrix multiplication?

I recommend the wonderful YouTube series by Grant Sanderson. His channel is called 3Blue1Brown.

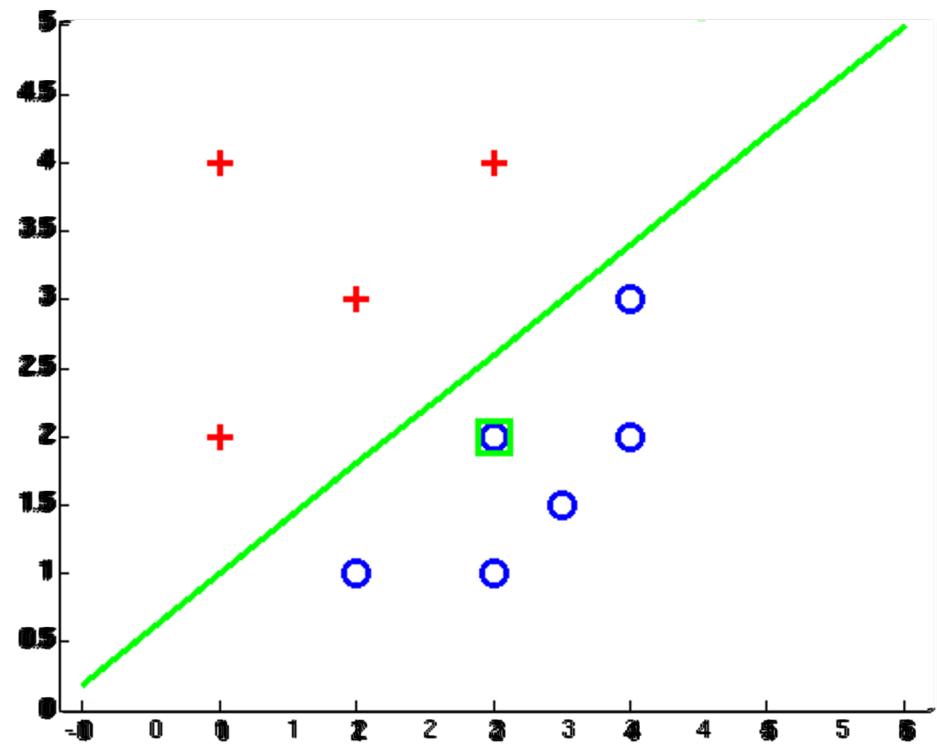
Grant gives intuitive visual tutorials to a ton of math concepts

Here is his “Essence of Linear Algebra” series:

- [https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE\\_ab](https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab)

# Examples: Perceptron

Separable Case



# Multiclass Decision Rule

If we have multiple classes:

- A weight vector for each class:

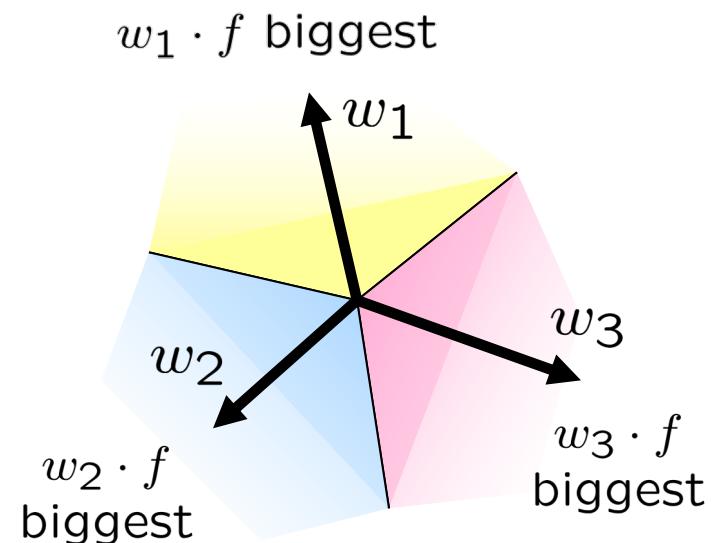
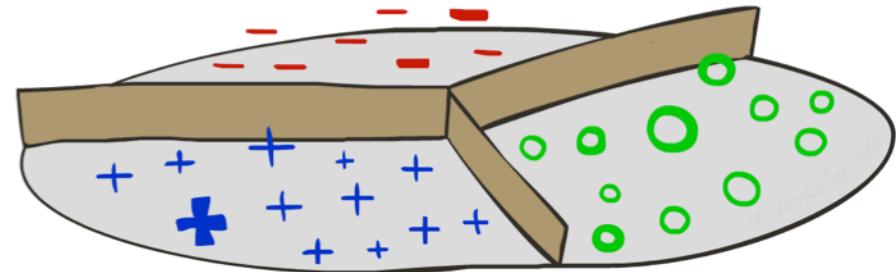
$$w_y$$

- Score (activation) of a class  $y$ :

$$w_y \cdot f(x)$$

- Prediction highest score wins

$$y = \arg \max_y w_y \cdot f(x)$$



*Binary = multiclass where the negative class has weight zero*

# Learning: Multiclass Perceptron

Start with all weights = 0

Pick up training examples one by one

Predict with current weights

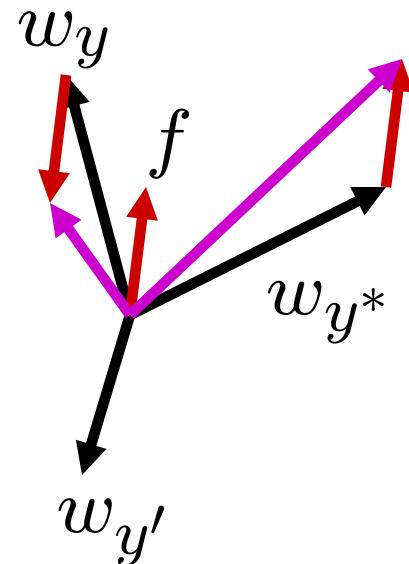
$$y = \arg \max_y w_y \cdot f(x)$$

If correct, no change!

If wrong: lower score of wrong answer. raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$



# Example: Multiclass Perceptron

“win the vote”

“win the election”

“win the game”

$w_{SPORTS}$

BIAS	:	1
win	:	0
game	:	0
vote	:	0
the	:	0
...		

$w_{POLITICS}$

BIAS	:	0
win	:	0
game	:	0
vote	:	0
the	:	0
...		

$w_{TECH}$

BIAS	:	0
win	:	0
game	:	0
vote	:	0
the	:	0
...		

# Properties of Perceptrons

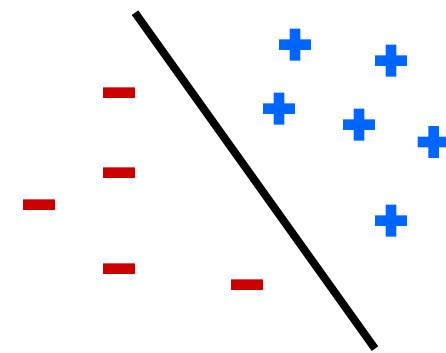
Separability: true if some parameters get the training set perfectly correct

Convergence: if the training is separable, perceptron will eventually converge (binary case)

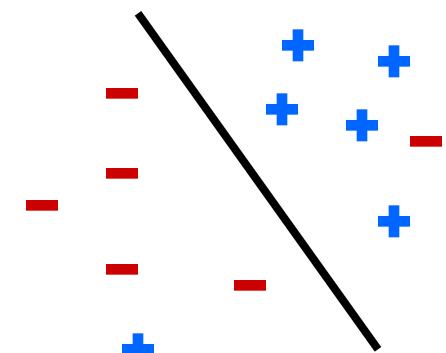
Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

Separable

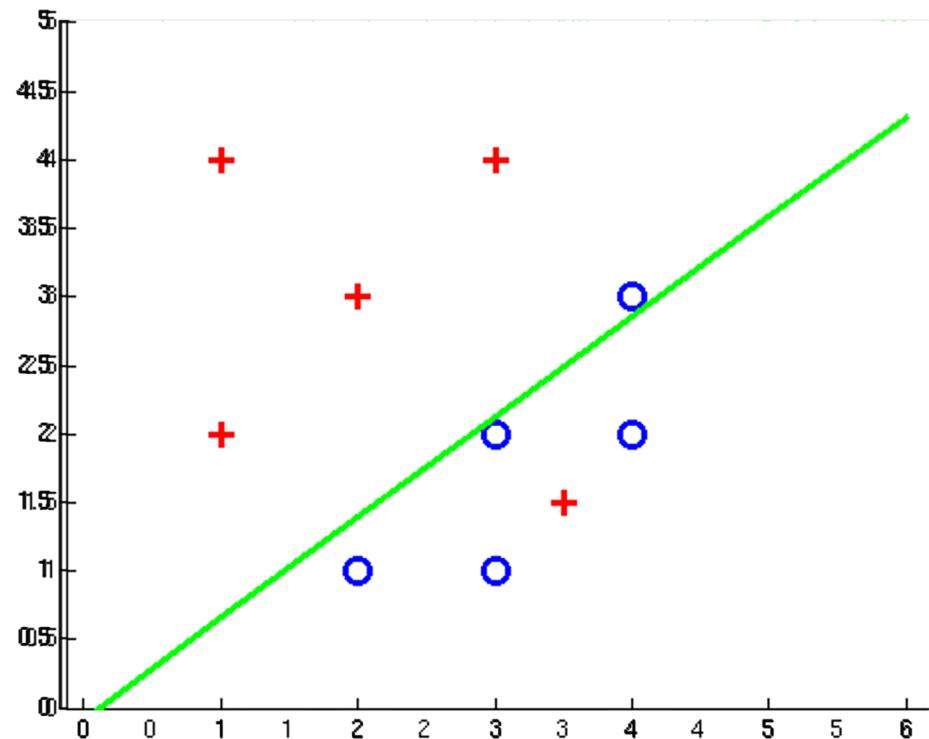


Non-Separable

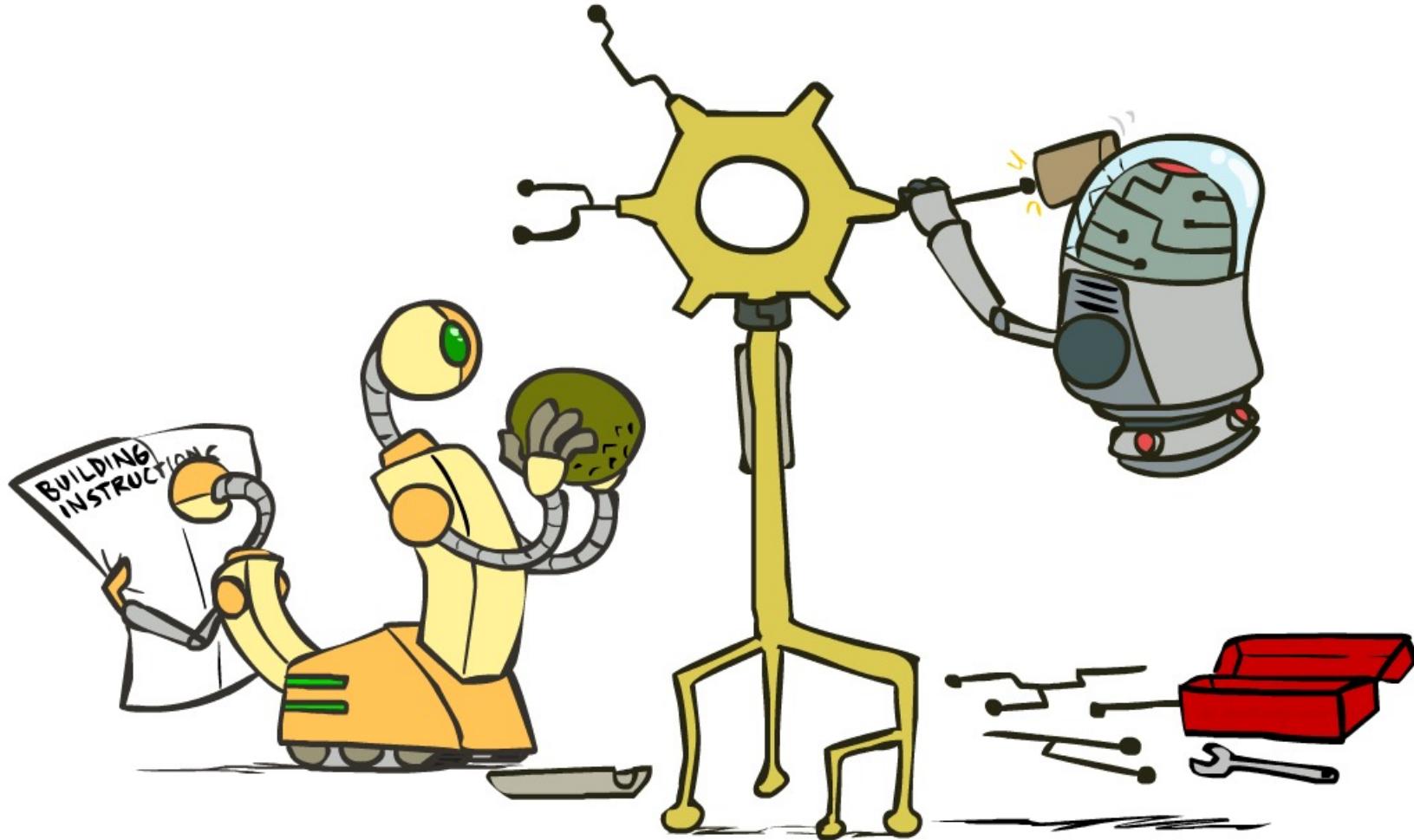


# Examples: Perceptron

Non-Separable Case



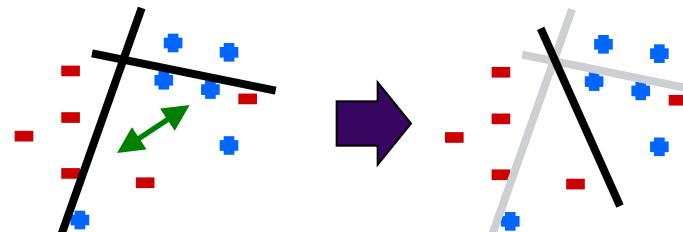
# Improving the Perceptron



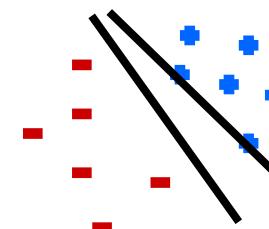
# Problems with the Perceptron

Noise: if the data isn't separable, weights might thrash

- Averaging weight vectors over time can help (averaged perceptron)

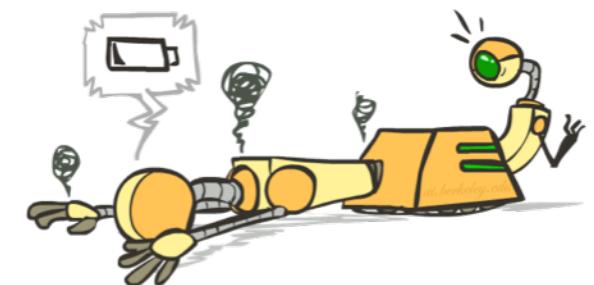
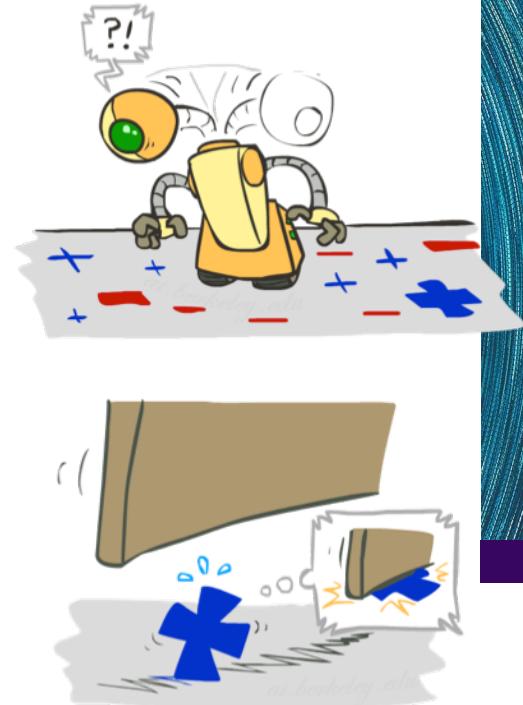
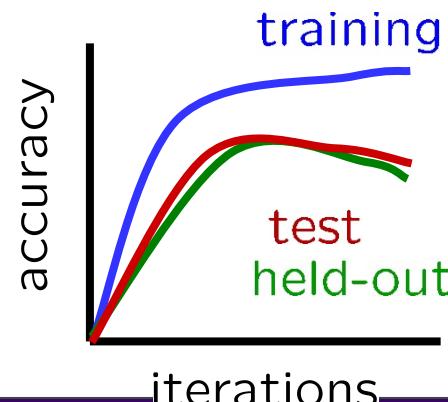


Mediocre generalization: finds a "barely" separating solution



Overtraining: test / held-out accuracy usually rises, then falls

- Overtraining is a kind of overfitting



# Fixing the Perceptron

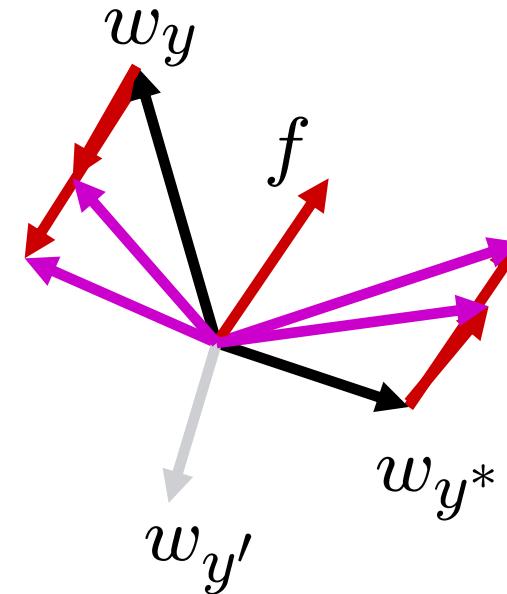
Idea: adjust the weight update to mitigate these effects

MIRA\*: choose an update size that fixes the current mistake...  
... but, minimizes the change to  $w$

$$\min_w \frac{1}{2} \sum_y \|w_y - w'_y\|^2$$

$$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$$

The +1 helps to generalize



Guessed  $y$  instead of  $y^*$  on example  $x$  with features  $f(x)$

$$w_y = w'_y - \tau f(x)$$

$$w_{y^*} = w'_{y^*} + \tau f(x)$$

# Minimum Correcting Update

$$\min_w \frac{1}{2} \sum_y \|w_y - w'_y\|^2$$
$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$



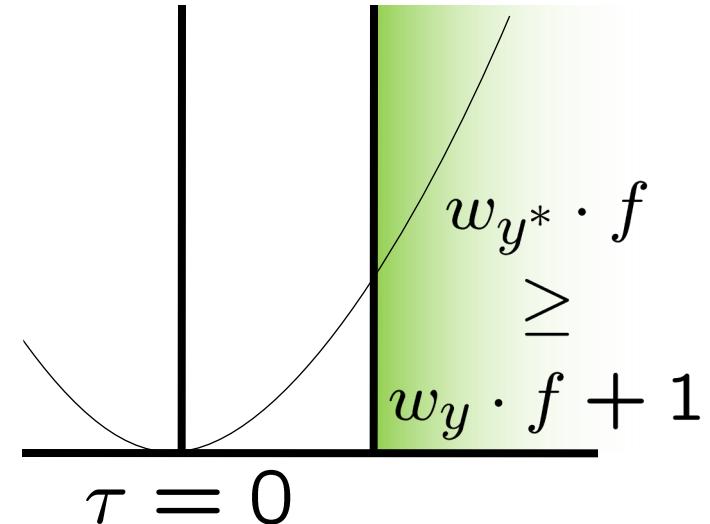
$$\min_{\tau} \|\tau f\|^2$$
$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$



$$(w'_{y^*} + \tau f) \cdot f = (w'_y - \tau f) \cdot f + 1$$

$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}$$

$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$



min not  $\tau=0$ , or would not have made an error, so min will be where equality holds

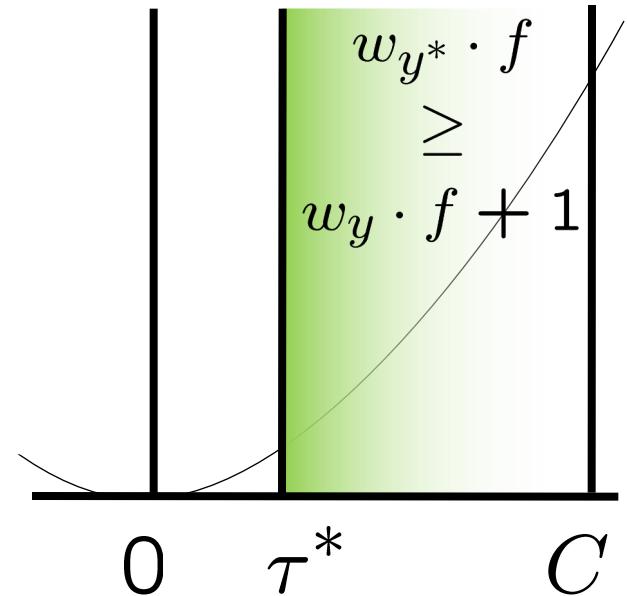
# Maximum Step Size

In practice, it's also bad to make updates that are too large

- Example may be labeled incorrectly
- You may not have enough features
- Solution: cap the maximum possible value of  $\tau$  with some constant  $C$

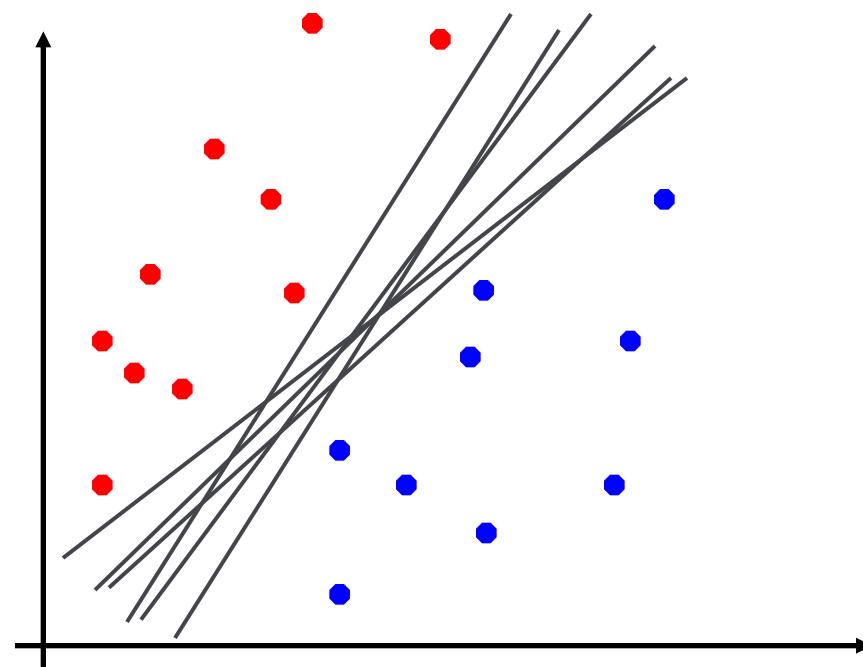
$$\tau^* = \min \left( \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}, C \right)$$

- Corresponds to an optimization that assumes non-separable data
- Usually converges faster than perceptron
- Usually better, especially on noisy data



# Linear Separators

Which of these linear separators is optimal?



# Support Vector Machines

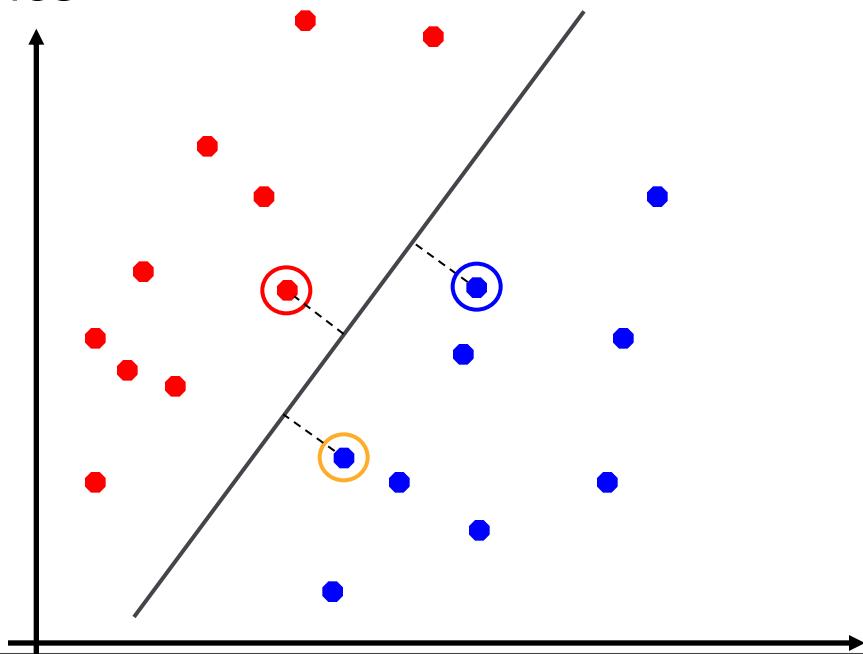
Maximizing the margin: good according to intuition, theory, practice

Only support vectors matter; other training examples are ignorable

Support vector machines (SVMs) find the separator with max margin

Basically, SVMs are MIRA where you optimize over all examples at once

$$\min_w \frac{1}{2} \|w - w'\|^2$$
$$w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$



$$\min_w \frac{1}{2} \|w\|^2$$
$$\forall i, y \quad w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

# Classification: Comparison

## Naïve Bayes

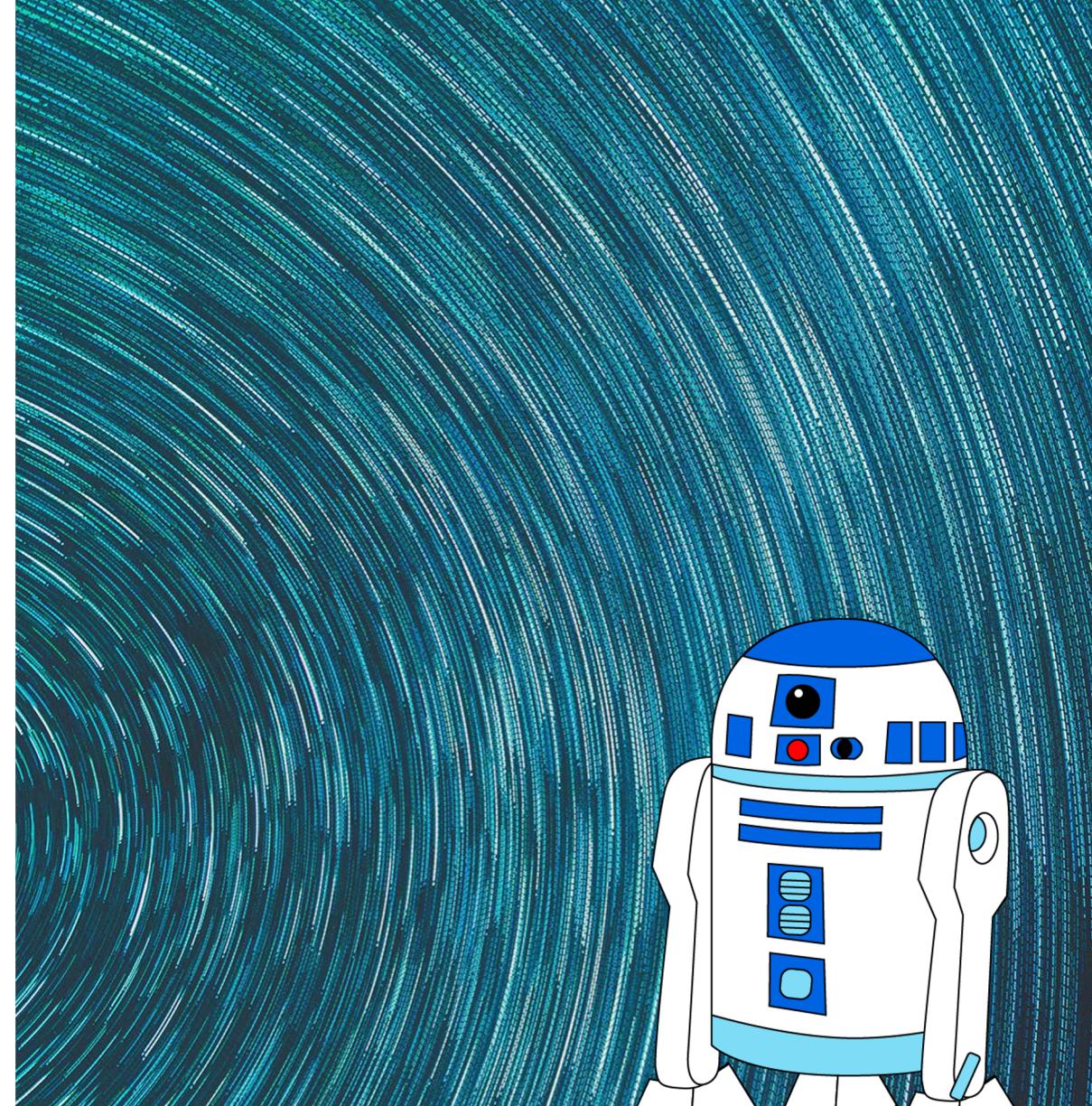
- Builds a model training data
- Gives prediction probabilities
- Strong assumptions about feature independence
- One pass through data (counting)

## Perceptrons / MIRA:

- Makes less assumptions about data
- Mistake-driven learning
- Multiple passes through data (prediction)
- Often more accurate

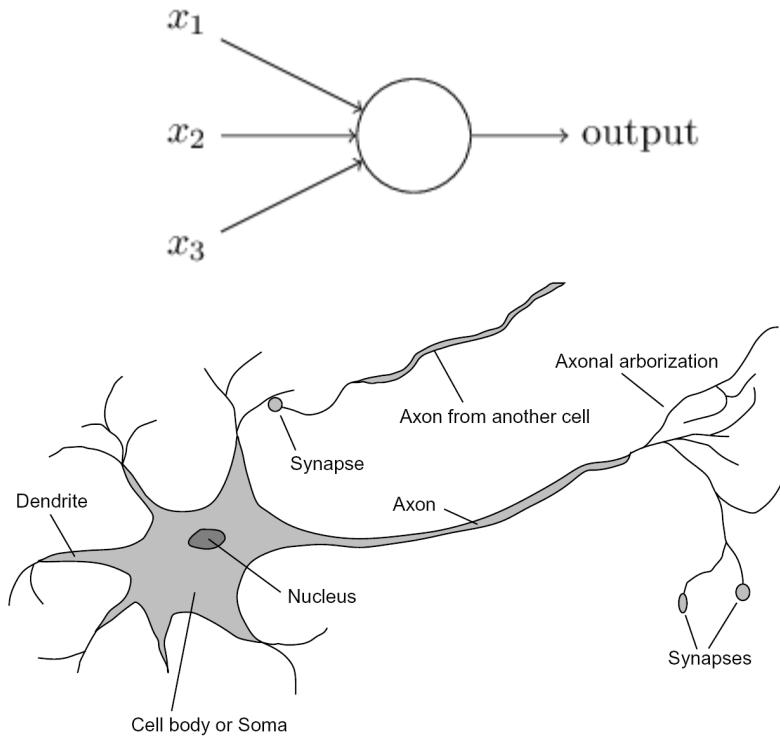
CIS 421/521:  
ARTIFICIAL INTELLIGENCE

# Perceptrons



# Perceptrons

Perceptrons were developed in the 1950s and 1960s loosely inspired by the neuron.



# Electronic 'Brain' Teaches Itself

The Navy last week demonstrated the embryo of an electronic computer named the Perceptron which, when completed in about a year, is expected to be the first non-living mechanism able to "perceive, recognize and identify its surroundings without human training or control." Navy officers demonstrating a preliminary form of the device in Washington said they hesitated to call it a machine because it is so much like a "human being without life."

Dr. Frank Rosenblatt, research psychologist at the Cornell Aeronautical Laboratory, Inc., Buffalo, N. Y., designer of the Perceptron, conducted the demonstration. The machine, he said, would be the first electronic device to think as the human brain. Like humans, Perceptron will make mistakes at first, "but it will grow wiser as it gains experience," he said.

The first Perceptron, to cost about \$100,000, will have about 1,000 electronic "association cells" receiving electrical impulses from an eyelike scanning device with 400 photocells. The human brain has ten billion responsive cells, including 100,000,000 connections with the eye.

## Difference Recognized

recognize the difference between right and left, almost the way a child learns.

When fully developed, the Perceptron will be designed to remember images and information it has perceived itself, whereas ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons, Dr. Rosenblatt said, will be able to recognize people and call out their names. Printed pages, longhand letters and even speech commands are within its reach. Only one more step of development, a difficult step, he said, is needed for the device to hear speech in one language and instantly translate it to speech or writing in another language.

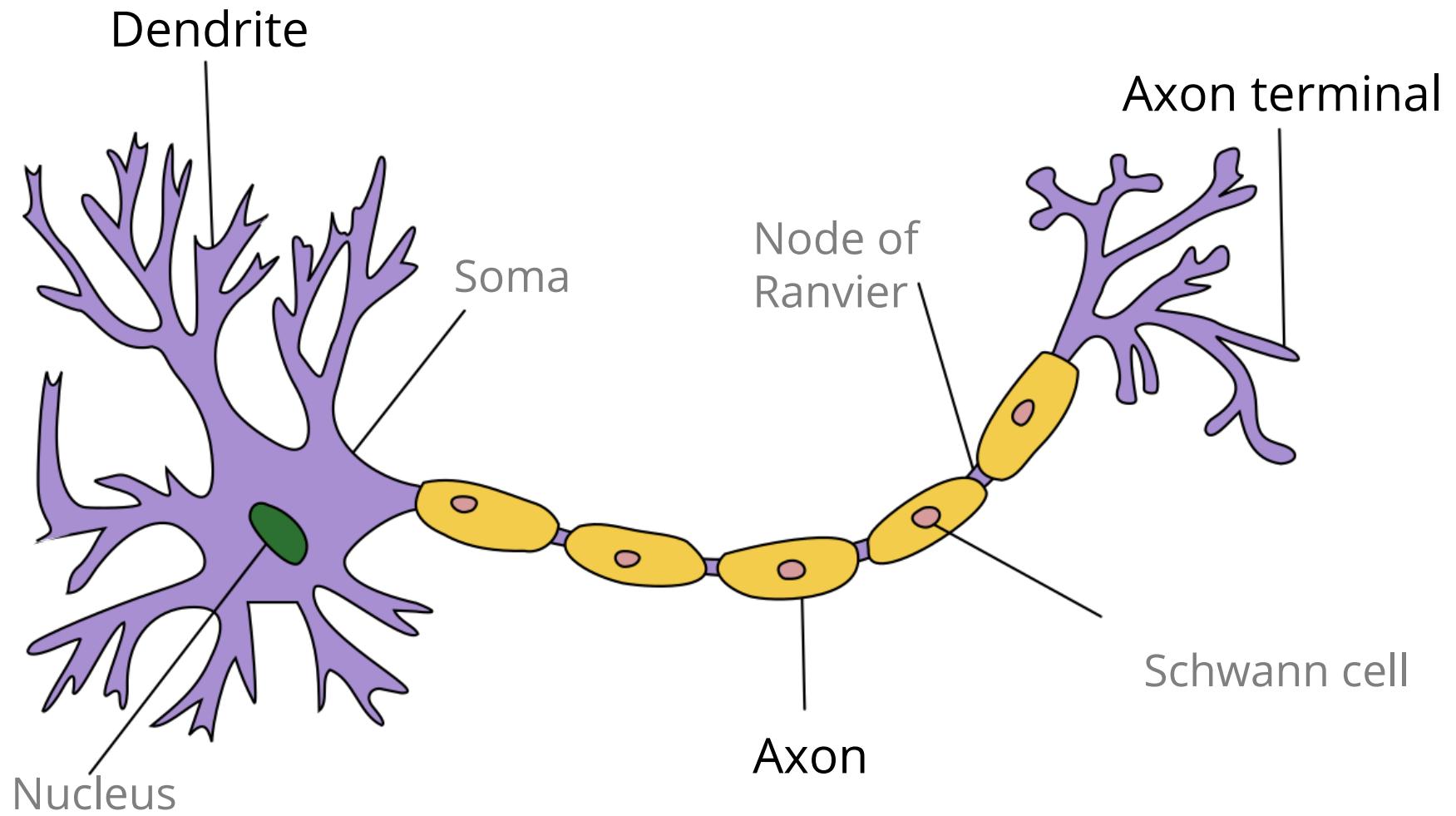
## Self-Reproduction

In principle, Dr. Rosenblatt said, it would be possible to build Perceptrons that could reproduce themselves on an assembly line and which would be "conscious" of their existence.

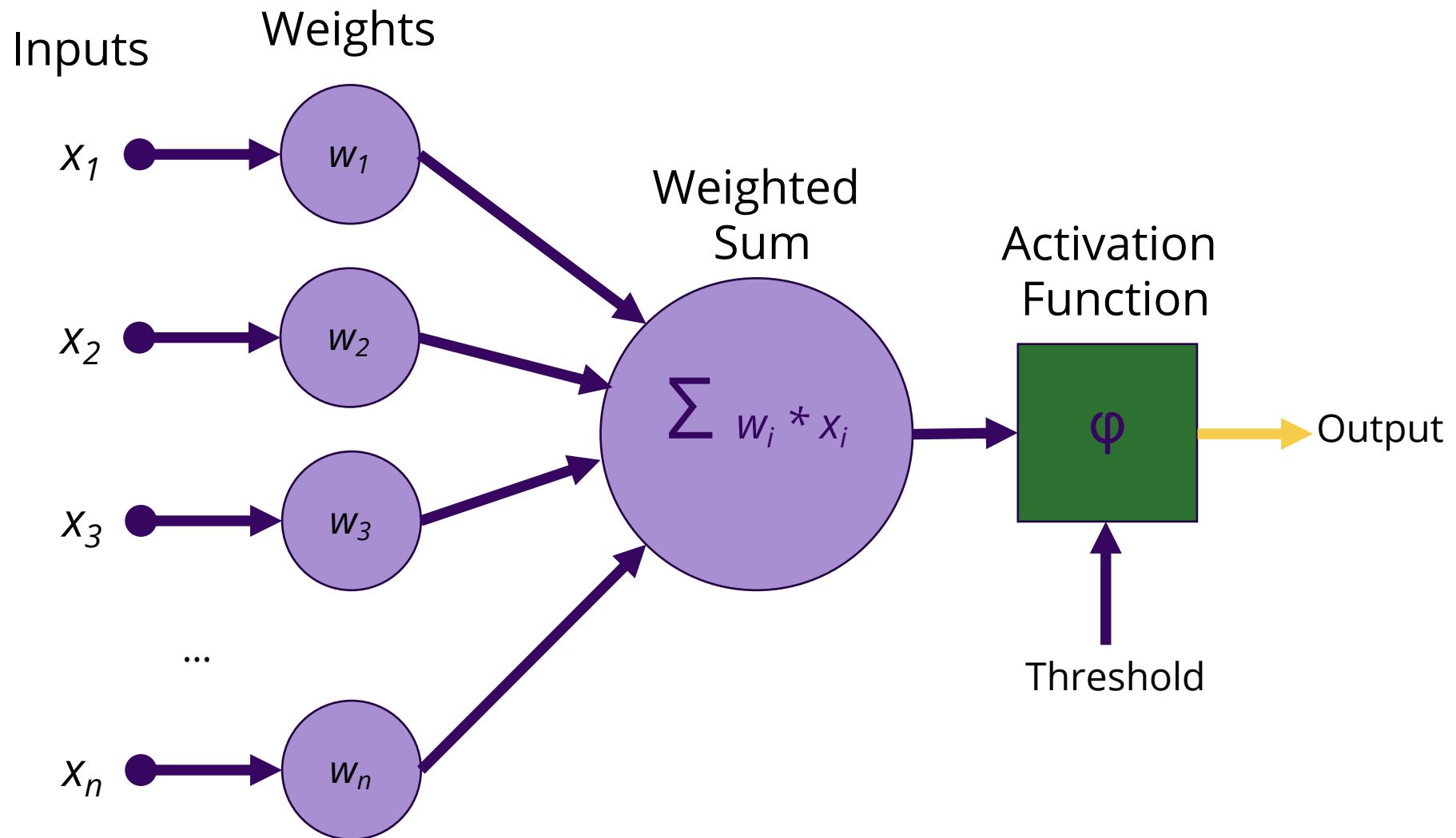
Perceptron, it was pointed out, needs no "priming." It is not necessary to introduce it to surroundings and circumstances, record the data involved and then store them for future comparison as is the case



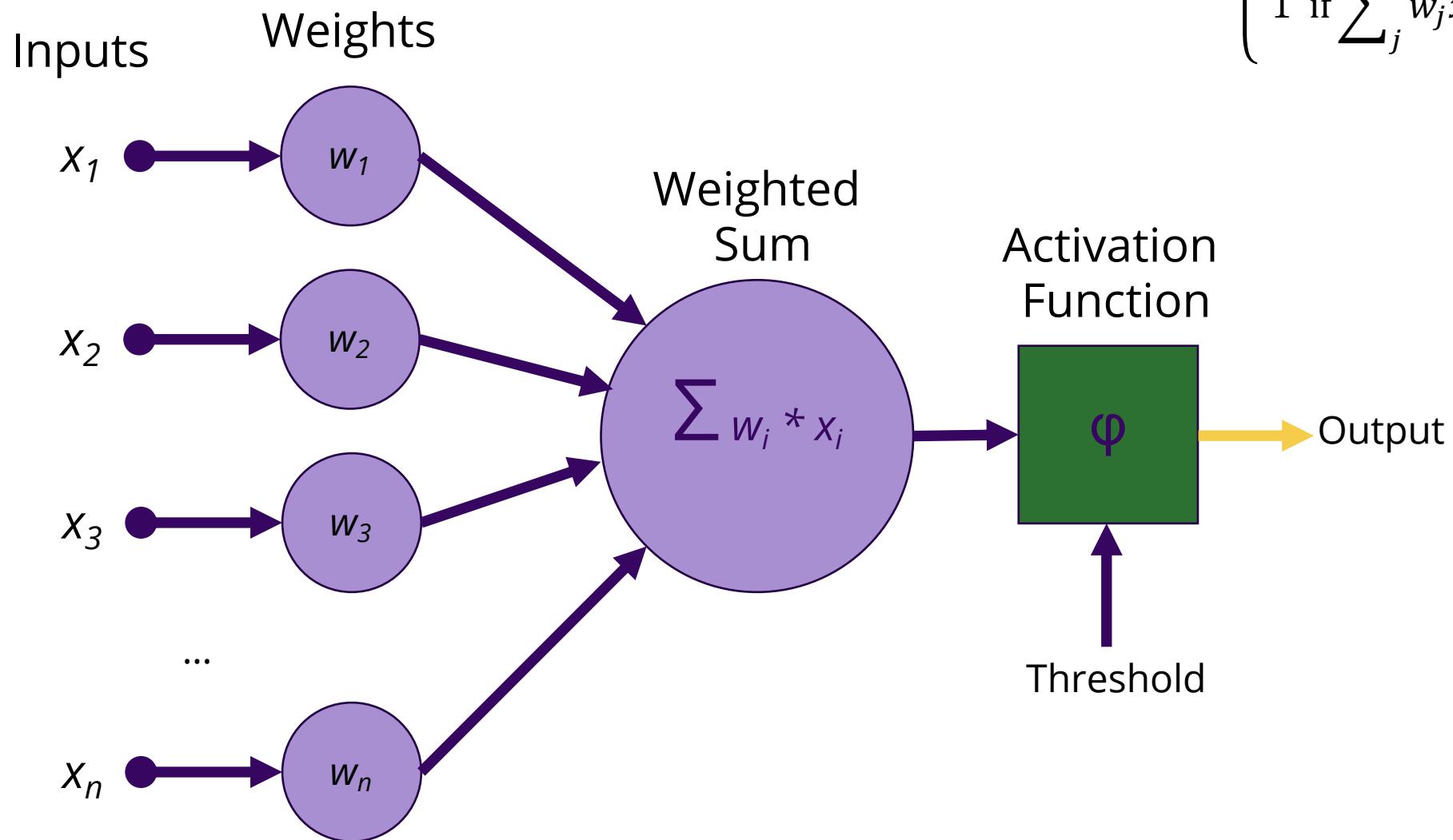
# Neuron



# Perceptron



# Perceptron

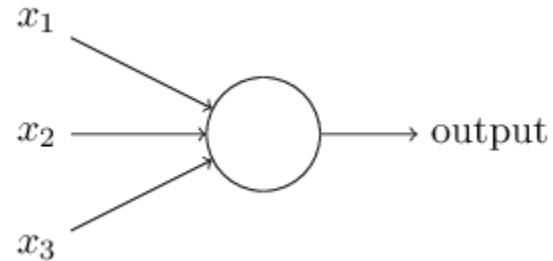


$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

# Perceptrons for decision making

We can think about the perceptron or the sigmoid neuron as a device that makes decisions by weighing up evidence.

Example: Suppose there's a cheese festival in your town. You like cheese.



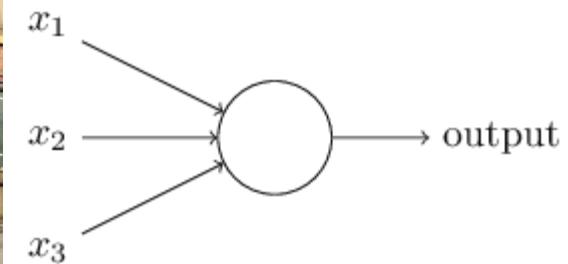
Example from Michael Nielsen's book [Neural Networks and Deep Learning](#)

# Perceptrons for decision making

You might use 3 factors to decide whether to go.

1. Is the weather good?
2. Can your loyal companion come with you?
3. Is the festival near public transit?

These can be the binary input values to a perceptron

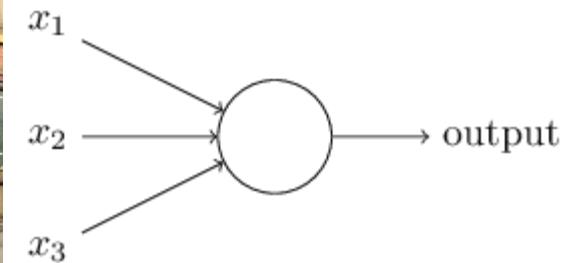


# Perceptrons for decision making

By varying weights and the threshold we get different models of decision making

Example 1:  $w_1 = 6 \quad w_2 = 2 \quad w_3 = 2$ , threshold = 5

Example 2:  $w_1 = 6 \quad w_2 = 2 \quad w_3 = 2$ , threshold = 3



# Notational changes

Change 1: We can write  $\sum_j w_j x_j$  as a dot product of the input vector and the weight vector:

$$\sum_j w_j x_j \equiv \mathbf{w} \cdot \mathbf{x}$$

Change 2: We can move the threshold to other other side of the inequality. We define a perceptron's "bias" as the -1 \* its threshold:

$$b \equiv -\text{threshold}$$

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

rewrites to

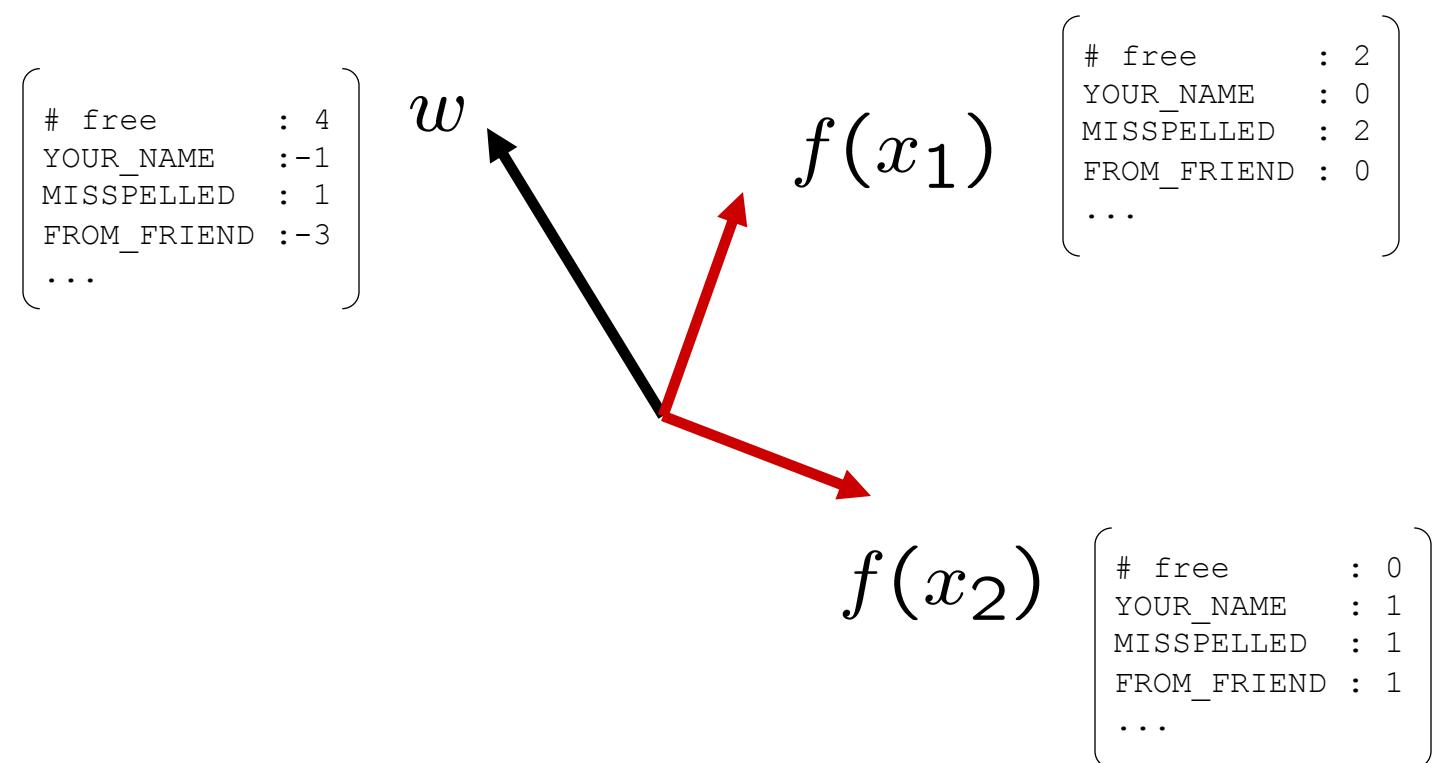
$$\text{output} = \begin{cases} 0 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

# Learning weights from examples

Perceptions can be used for all kinds of classification problems.

Think of the inputs as *features* representing something we want to classify.

The feature values for inputs are fixed, but we can choose different weight vectors. Depending on the weight vector that we pick, we will get a different classifier.

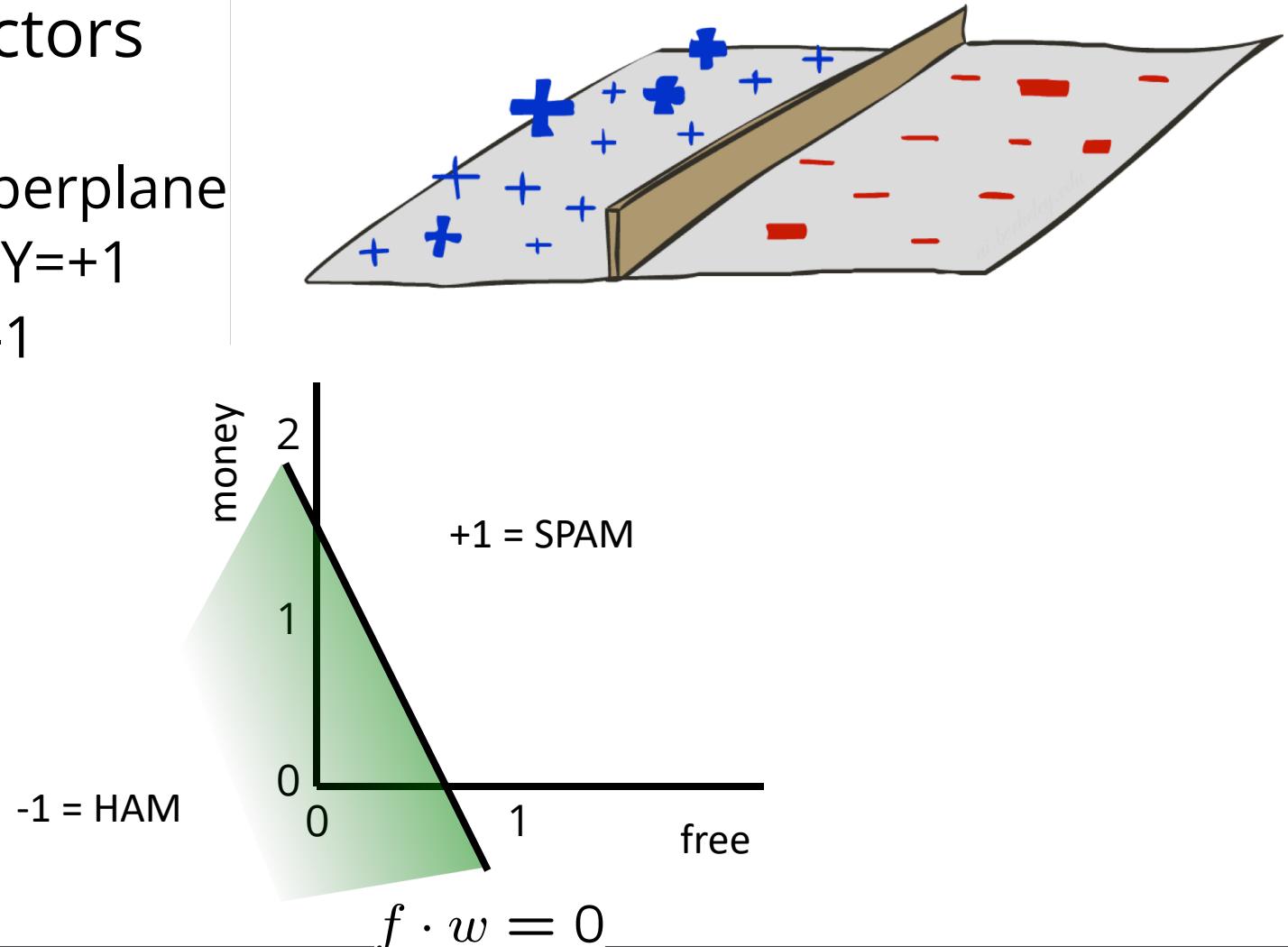


# Binary Decision Rule

In the space of feature vectors

- Examples are points
- Any weight vector is a hyperplane
- One side corresponds to  $Y=+1$
- Other corresponds to  $Y=-1$

$w$	
BIAS	: -3
free	: 4
money	: 2
...	



# Weight Updates

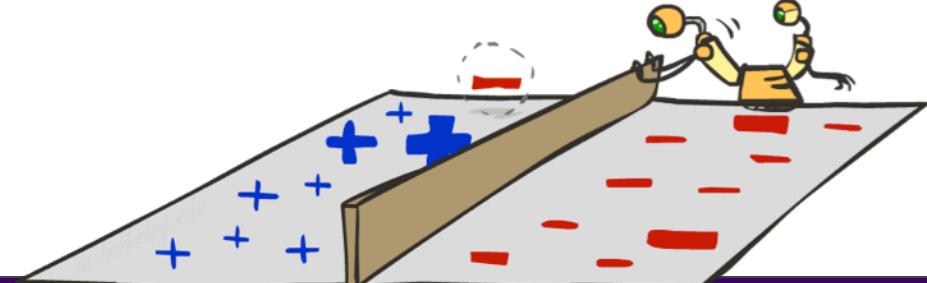
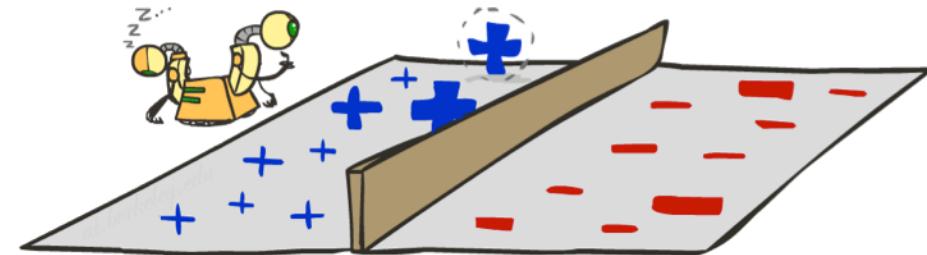
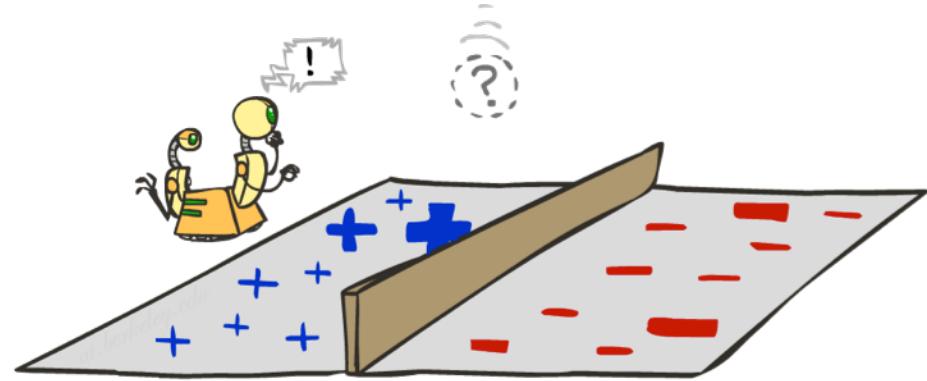


# Learning: Binary Perceptron

Start with weights = 0

For each training instance:

- Classify with current weights
- If correct (i.e.,  $y=y^*$ ), no change!
- If wrong: adjust the weight vector

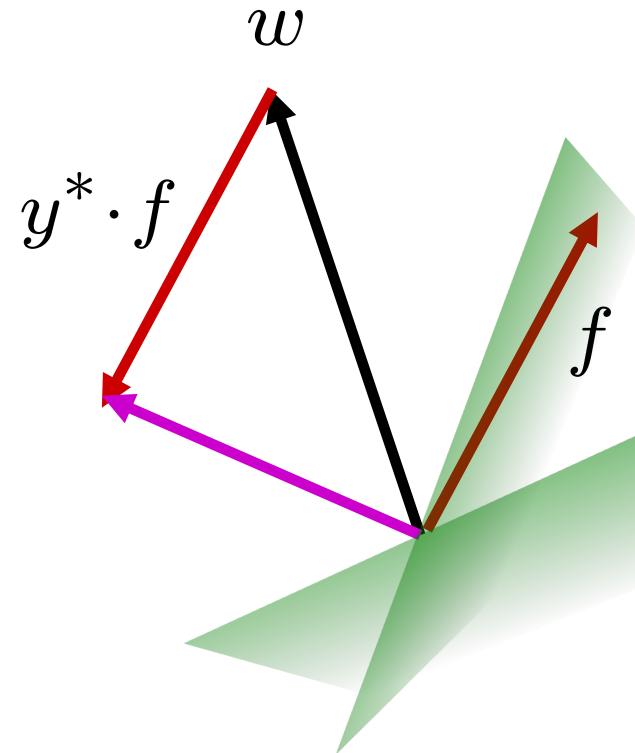


# Learning a Binary Perceptron

Start with weights = 0

For each training instance:

- Classify with current weights
- $y = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{f}(x) \leq 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{f}(x) > 0 \end{cases}$
- If correct (i.e.,  $y=y^*$ ), no change!
- If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if  $y^*$  is -1.
- $\mathbf{w} = \mathbf{w} + y^* \cdot \mathbf{f}$



# Multiclass Decision Rule

If we have multiple classes:

- A weight vector for each class:

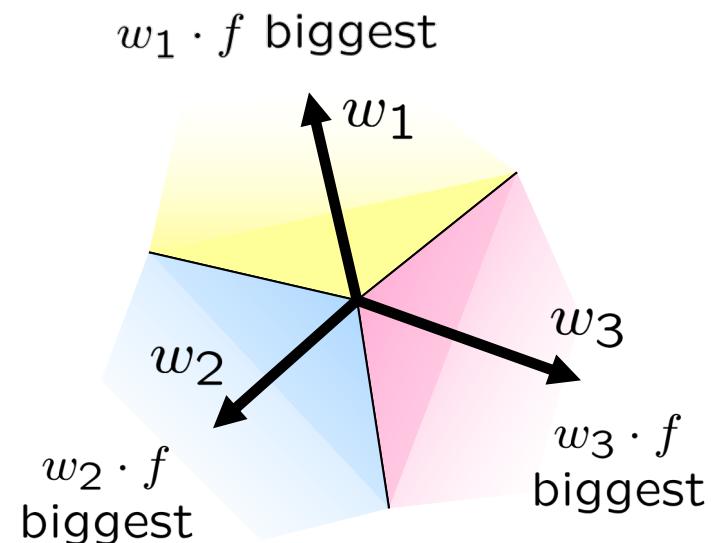
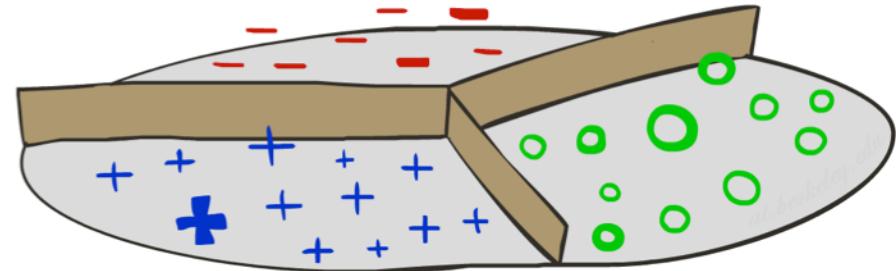
$$w_y$$

- Score (activation) of a class  $y$ :

$$w_y \cdot f(x)$$

- Prediction highest score wins

$$y = \arg \max_y w_y \cdot f(x)$$



*Binary = multiclass where the negative class has weight zero*

# Learning: Multiclass Perceptron

Start with all weights = 0

Pick up training examples one by one

Predict with current weights

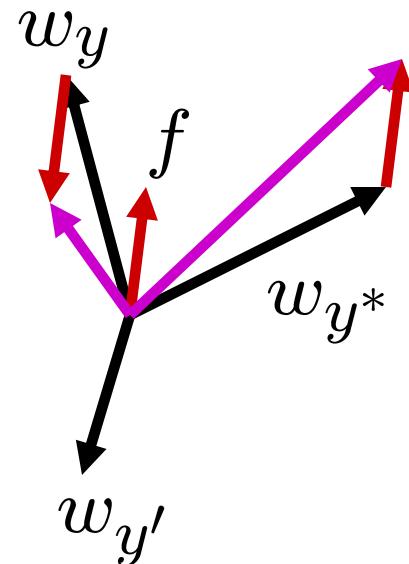
$$y = \arg \max_y w_y \cdot f(x)$$

If correct, no change!

If wrong: lower score of wrong answer. raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$



# Example: Multiclass Perceptron

“win the vote”

“win the election”

“win the game”

$w_{SPORTS}$

BIAS	:	1
win	:	0
game	:	0
vote	:	0
the	:	0
...		

$w_{POLITICS}$

BIAS	:	0
win	:	0
game	:	0
vote	:	0
the	:	0
...		

$w_{TECH}$

BIAS	:	0
win	:	0
game	:	0
vote	:	0
the	:	0
...		

# Properties of Perceptrons

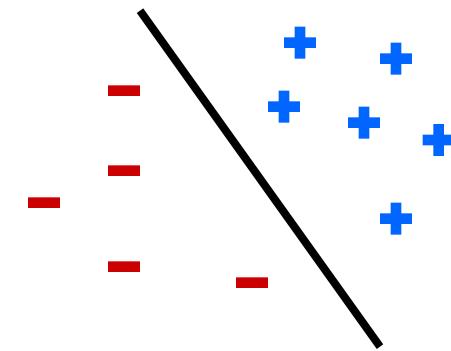
Separability: true if some parameters get the training set perfectly correct

Convergence: if the training is separable, perceptron will eventually converge (binary case)

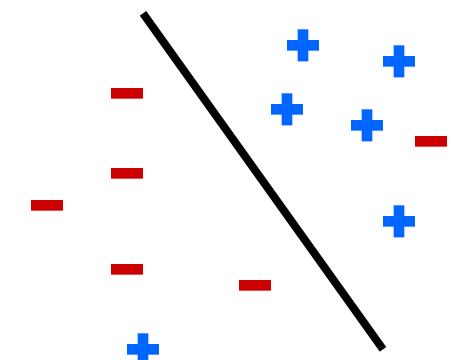
Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

Separable

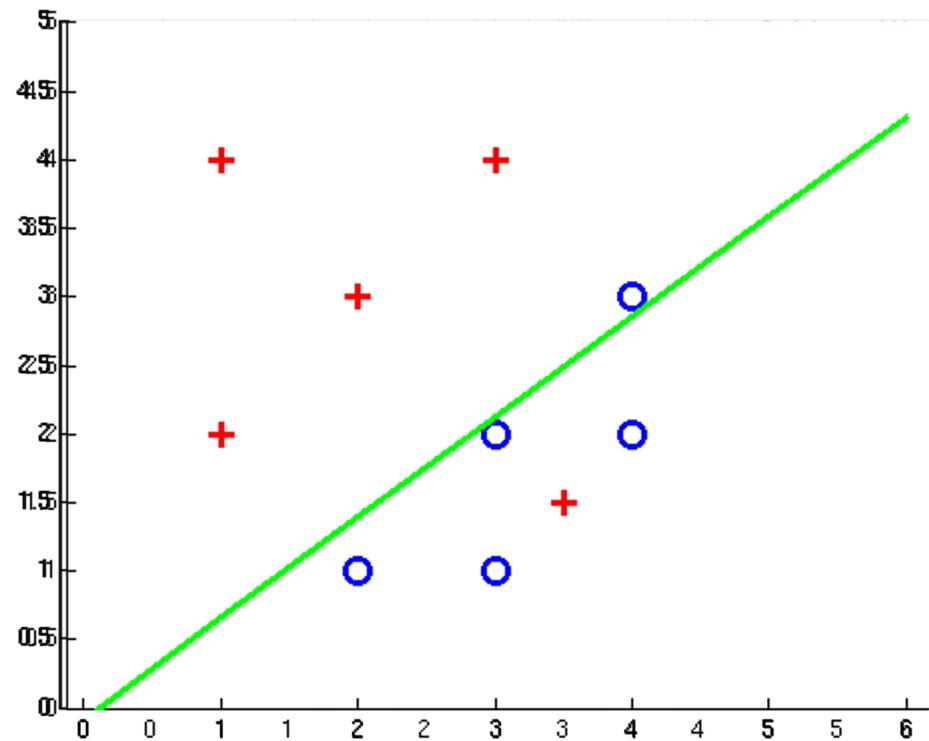


Non-Separable

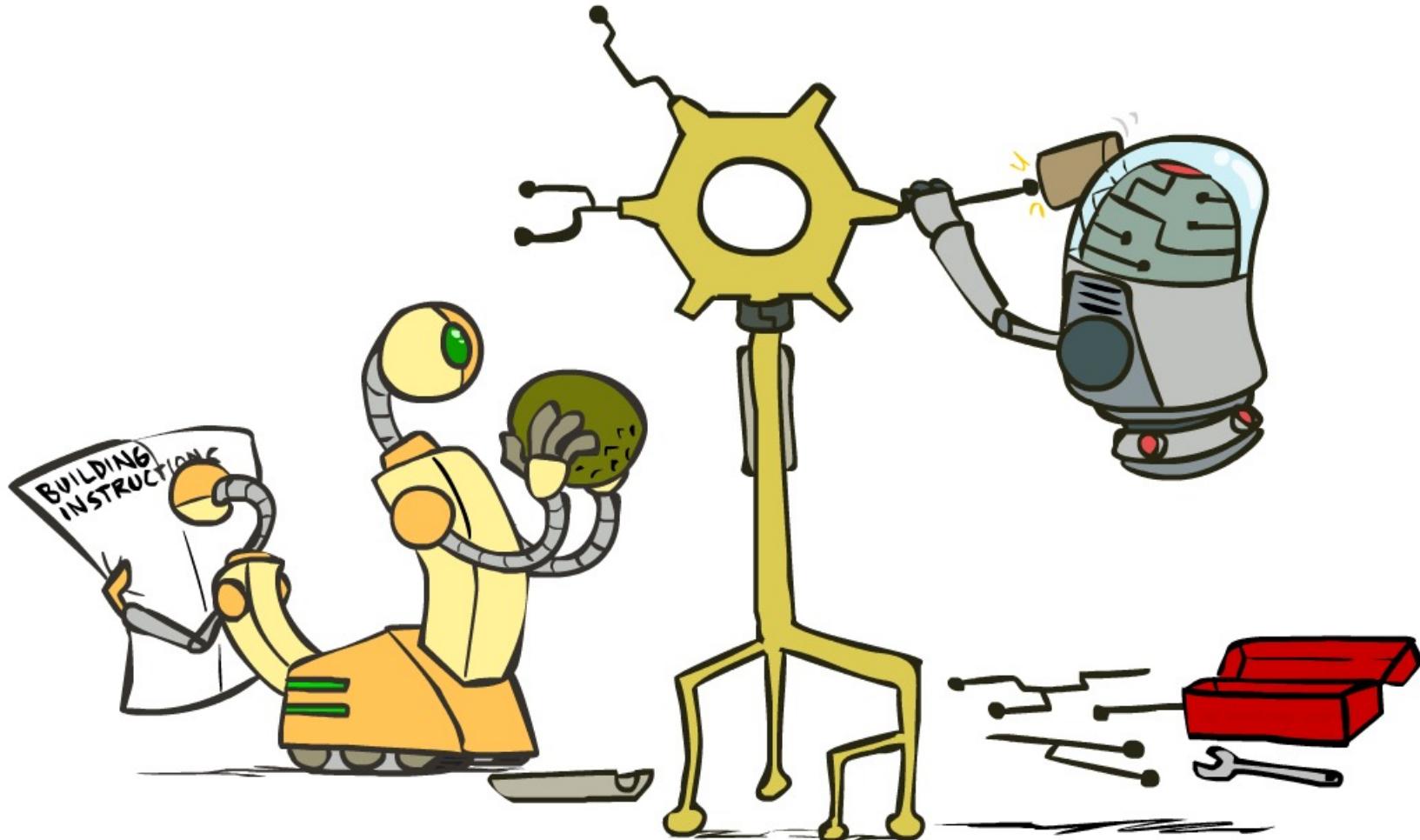


# Examples: Perceptron

Non-Separable Case



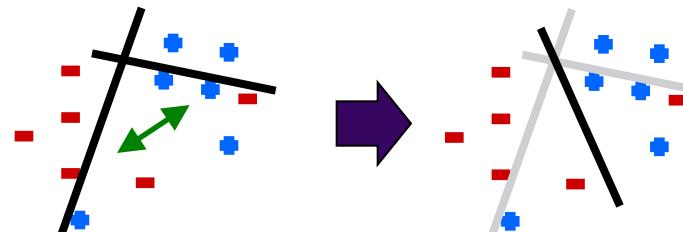
# Improving the Perceptron



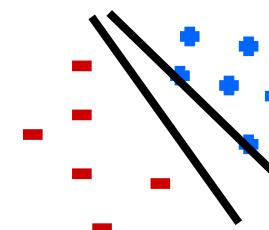
# Problems with the Perceptron

Noise: if the data isn't separable, weights might thrash

- Averaging weight vectors over time can help (averaged perceptron)

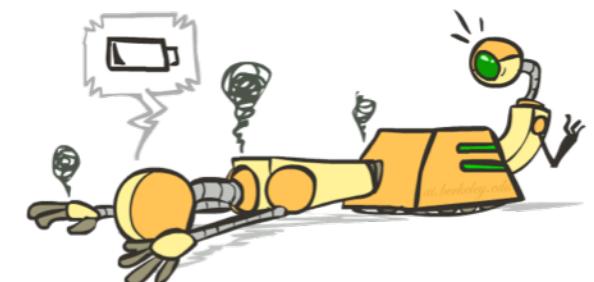
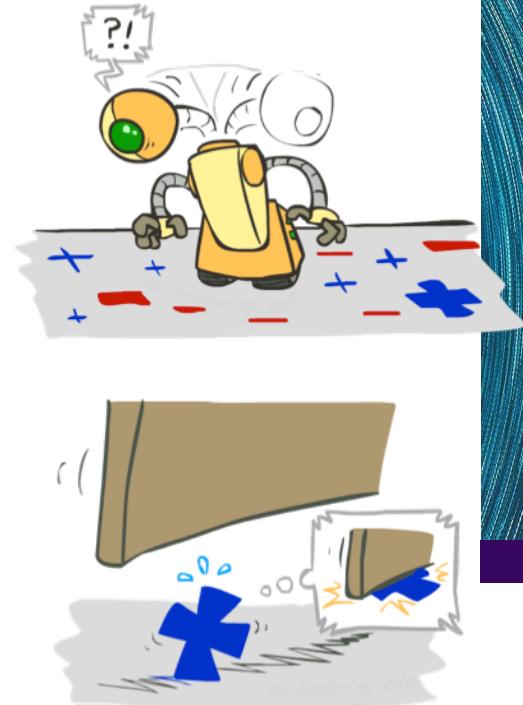
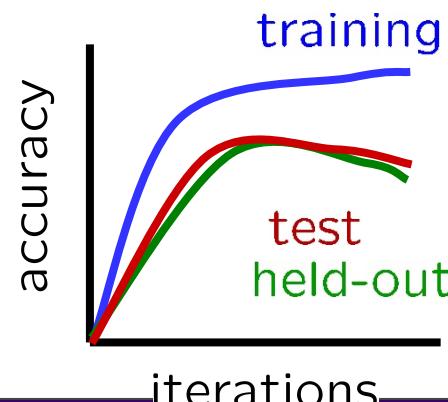


Mediocre generalization: finds a "barely" separating solution



Overtraining: test / held-out accuracy usually rises, then falls

- Overtraining is a kind of overfitting



# Fixing the Perceptron

Idea: adjust the weight update to mitigate these effects

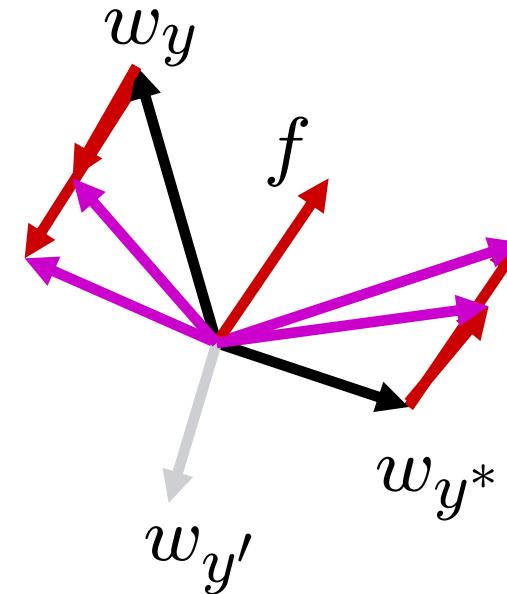
MIRA = Margin Infused Relaxed Algorithm

Choose an update size that fixes the current mistake...

... but, minimizes the change to  $w$

$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}$$

The  $+1$  helps to generalize



Guessed  $y$  instead of  $y^*$  on example  $x$  with features  $f(x)$

$$w_y = w'_y - \tau f(x)$$

$$w_{y^*} = w'_{y^*} + \tau f(x)$$

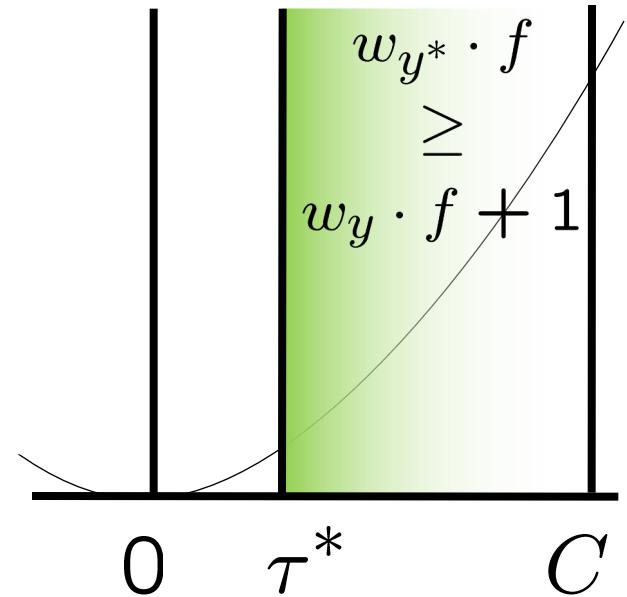
# Maximum Step Size

In practice, it's also bad to make updates that are too large

- Example may be labeled incorrectly
- You may not have enough features
- Solution: cap the maximum possible value of  $\tau$  with some constant  $C$

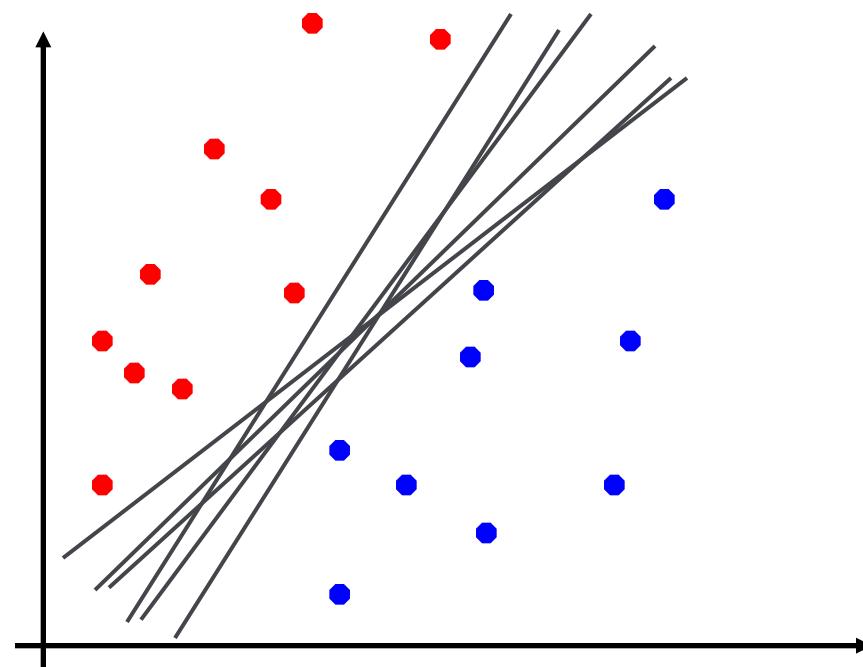
$$\tau^* = \min \left( \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}, C \right)$$

- Corresponds to an optimization that assumes non-separable data
- Usually converges faster than perceptron
- Usually better, especially on noisy data



# Linear Separators

Which of these linear separators is optimal?



# Support Vector Machines

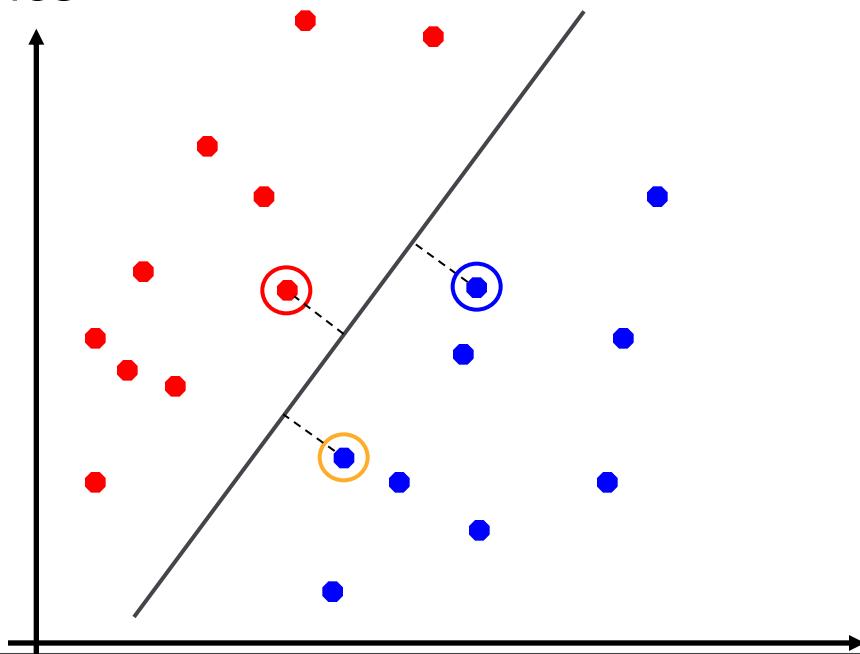
Maximizing the margin: good according to intuition, theory, practice

Only support vectors matter; other training examples are ignorable

Support vector machines (SVMs) find the separator with max margin

Basically, SVMs are MIRA where you optimize over all examples at once

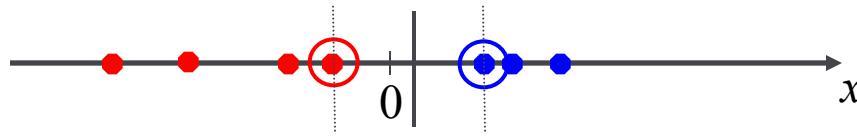
$$\min_w \frac{1}{2} \|w - w'\|^2$$
$$w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$



$$\min_w \frac{1}{2} \|w\|^2$$
$$\forall i, y \quad w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

# Non-Linear Separators

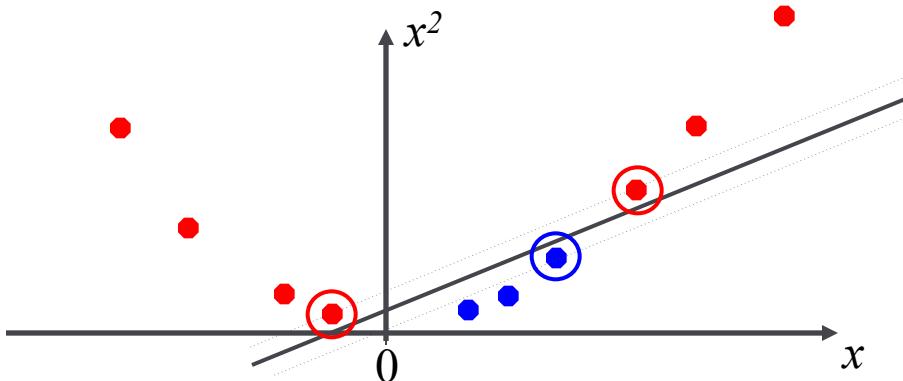
Data that is linearly separable works out great for linear decision rules:



But what are we going to do if the dataset is just too hard?

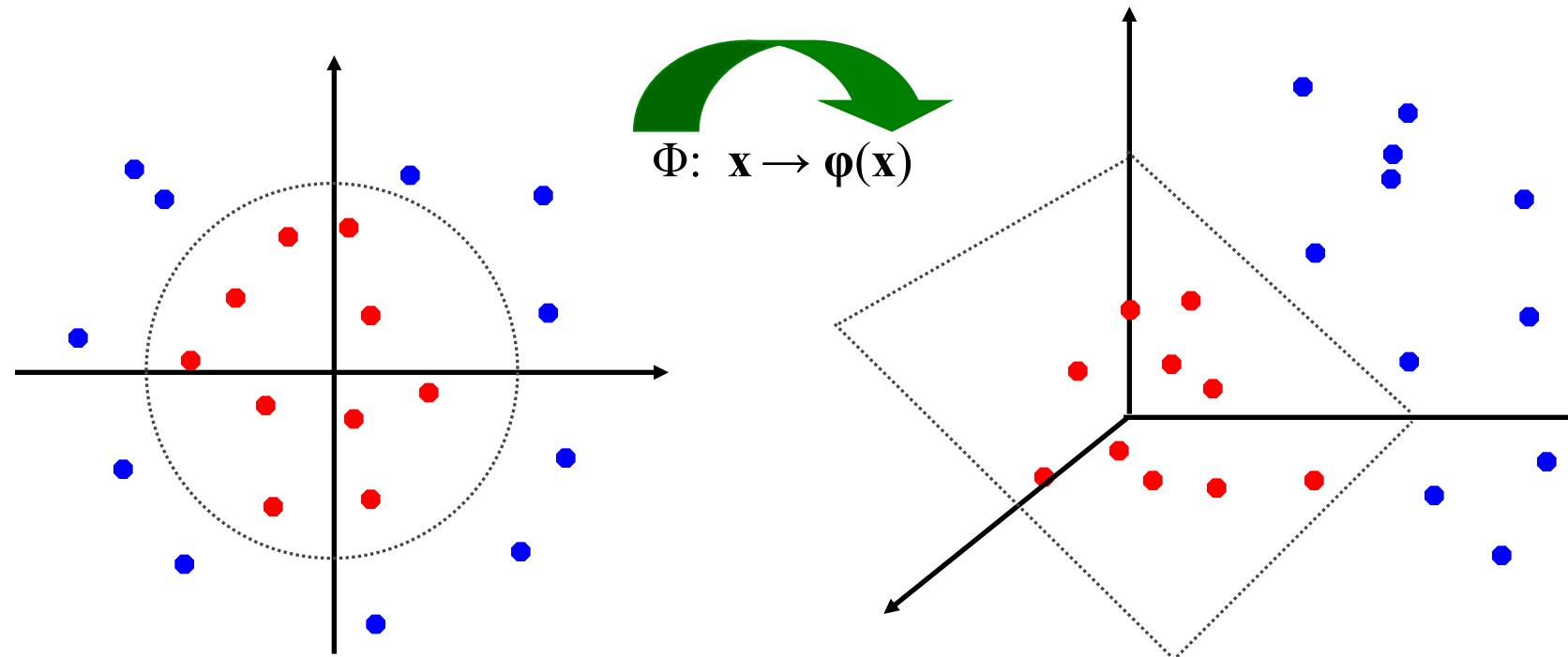


How about... mapping data to a higher-dimensional space:



# Non-Linear Separators

General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



# Classification: Comparison

## Naïve Bayes

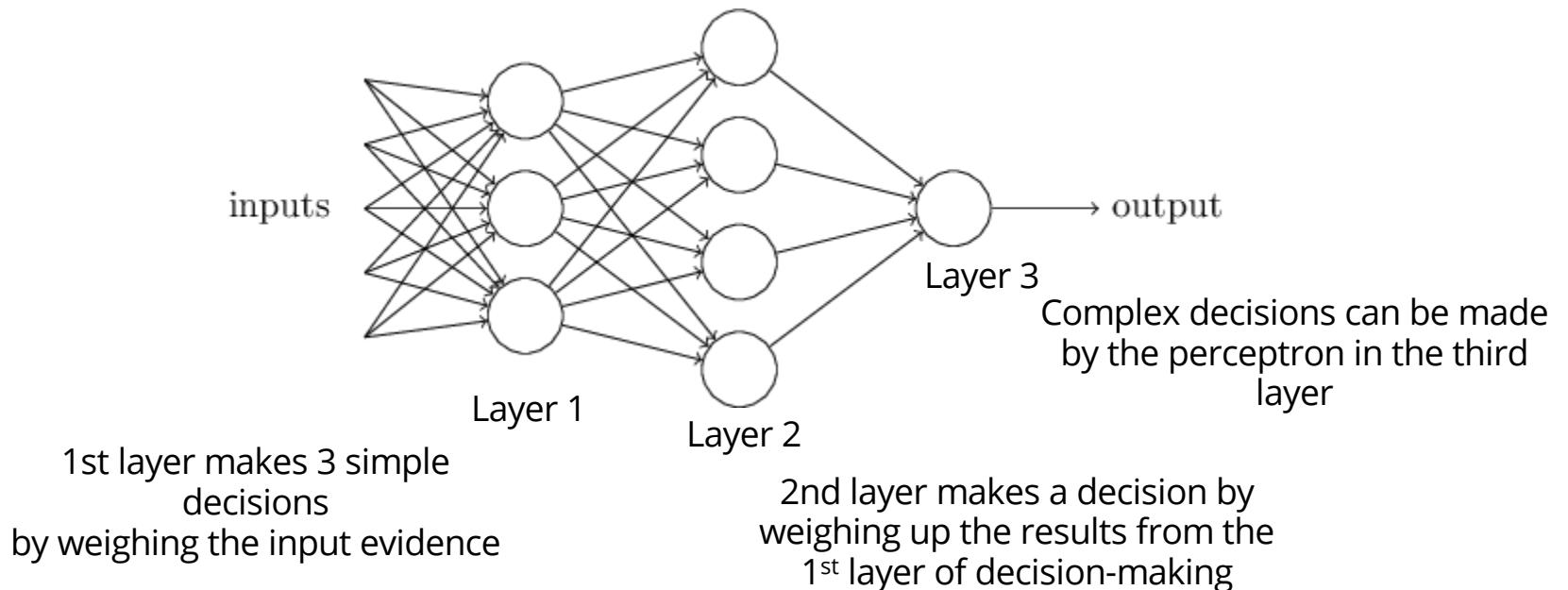
- Builds a model training data
- Gives prediction probabilities
- Strong assumptions about feature independence
- One pass through data (counting)

## Perceptrons / MIRA:

- Makes less assumptions about data
- Mistake-driven learning
- Multiple passes through data (prediction)
- Often more accurate

# Perceptrons for decision making

- A complex network of perceptrons could make quite subtle decisions:



# Weights, bias and dot products

- Two notational changes simplify the way that perceptrons are described.
- The first change is to replace the weighted sum as a dot product

$$w \cdot x \equiv \sum_j w_j x_j$$

- The second change is to move the threshold to the other side of the inequality, and to replace it by a *bias*,  $b$ -threshold

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

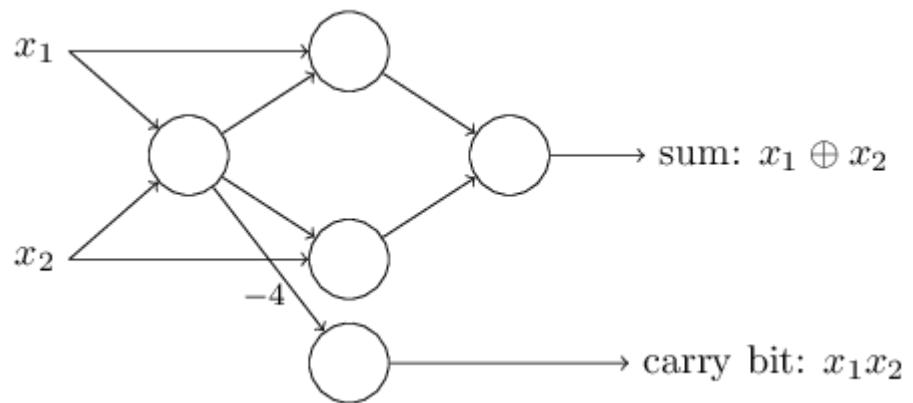
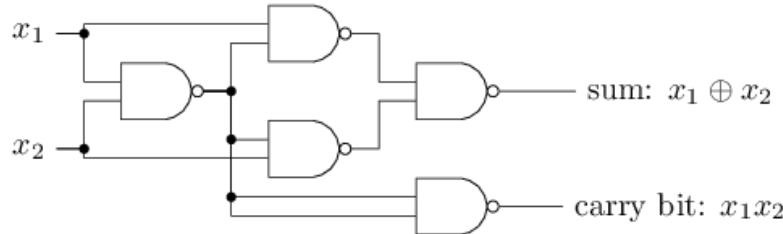
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

# Logical functions

Networks of perceptrons to compute *any* logical function

We can build any computation up out of NAND gates.

For example, a circuit which adds two bits  $x_1$  and  $x_2$



All unlabeled weights are -2, all biases =3.

# The XOR problem

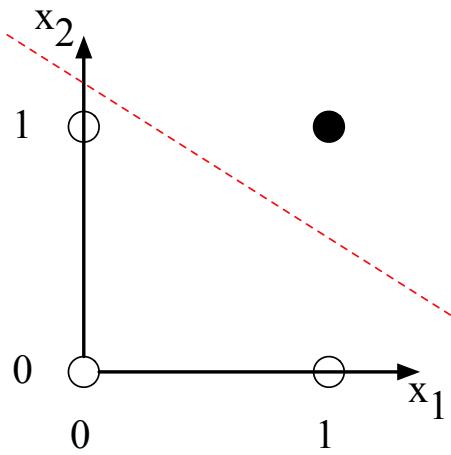
A single neural unit cannot be used to compute the XOR function

AND		
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

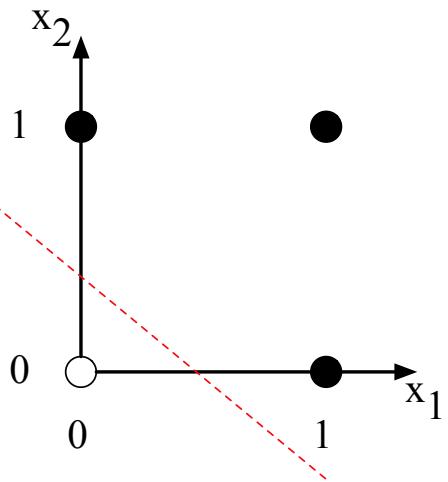
OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

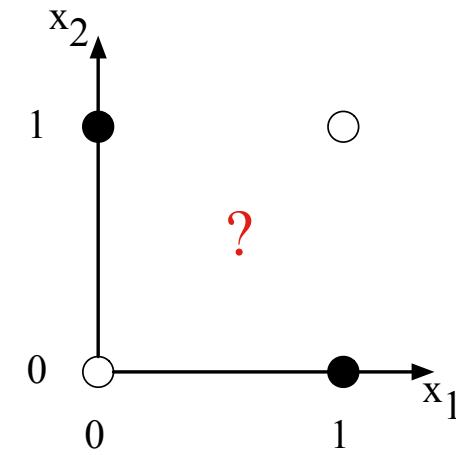
# The XOR Problem



a)  $x_1$  AND  $x_2$

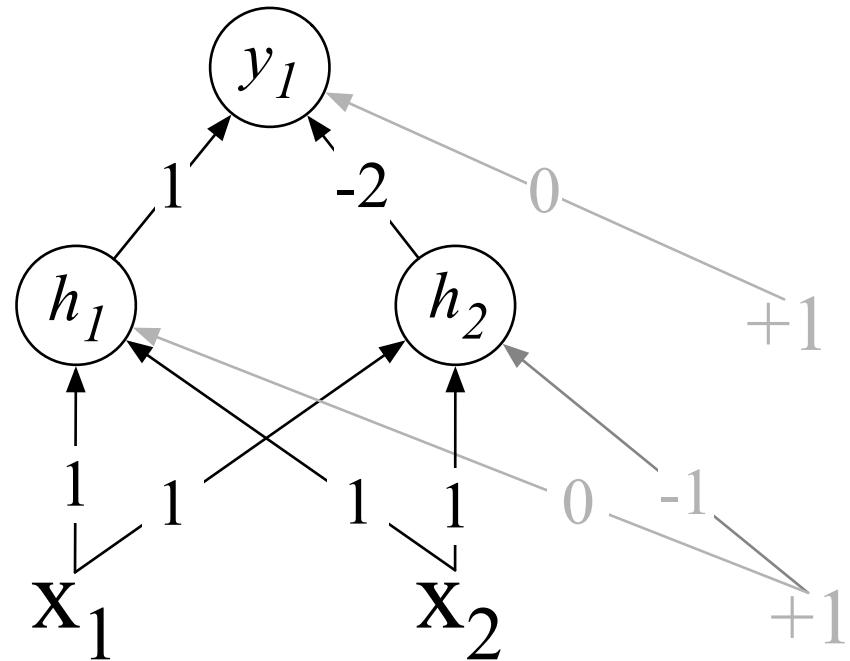


b)  $x_1$  OR  $x_2$

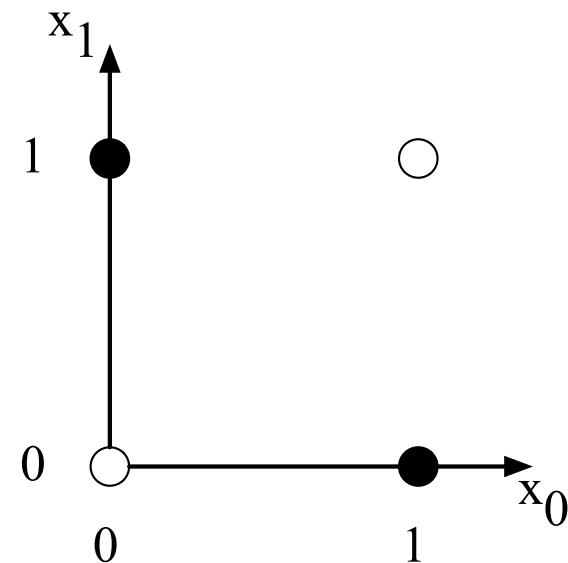


c)  $x_1$  XOR  $x_2$

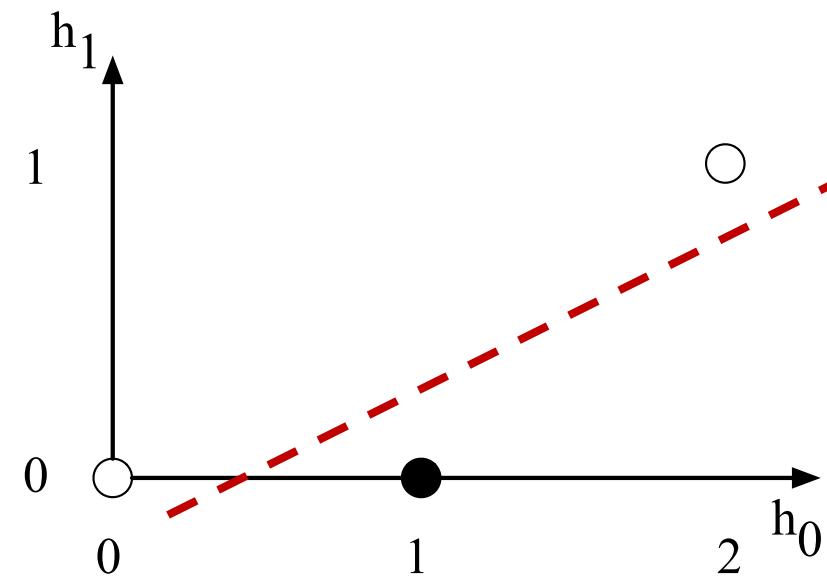
# The XOR Solution



# The XOR Solution



a) The original  $x$  space



b) The new  $h$  space

# Activation Functions

Instead of directly outputting  $z = w \cdot x + b$ , which is a linear function of  $x$ , neuron units apply a non-linear function  $f$  to  $z$ .

The output of this function is called the **activation value** for the unit, represented by the variable **a**. The output of a neural network is called **y**, so if the activation of a node is the final output of a network then

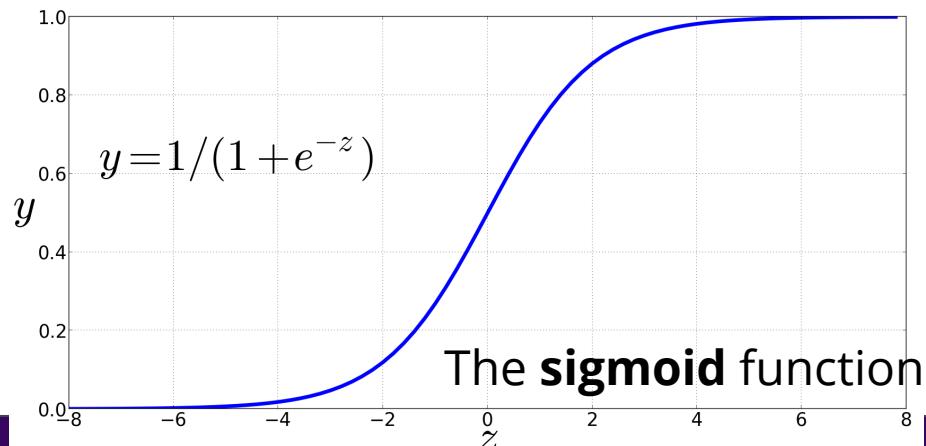
$$y=a=f(z)$$

There are 3 commonly used non-linear functions used for  $f$ :

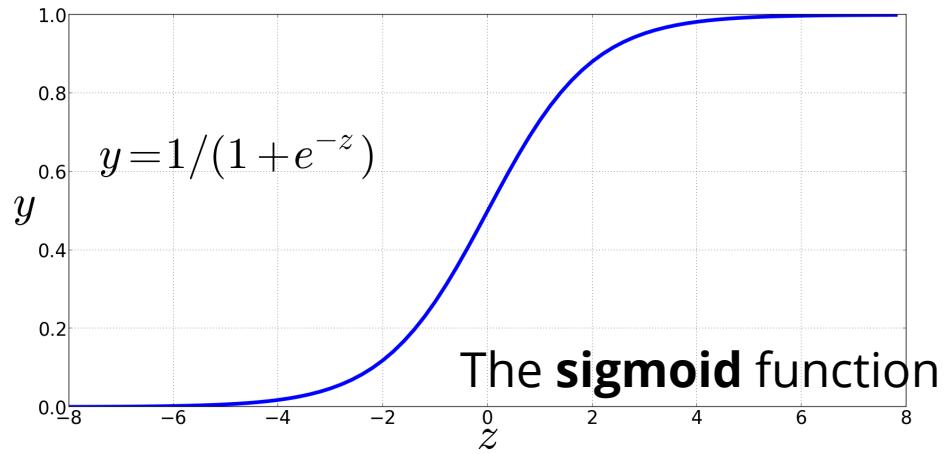
The **sigmoid** function

The **tanh** function

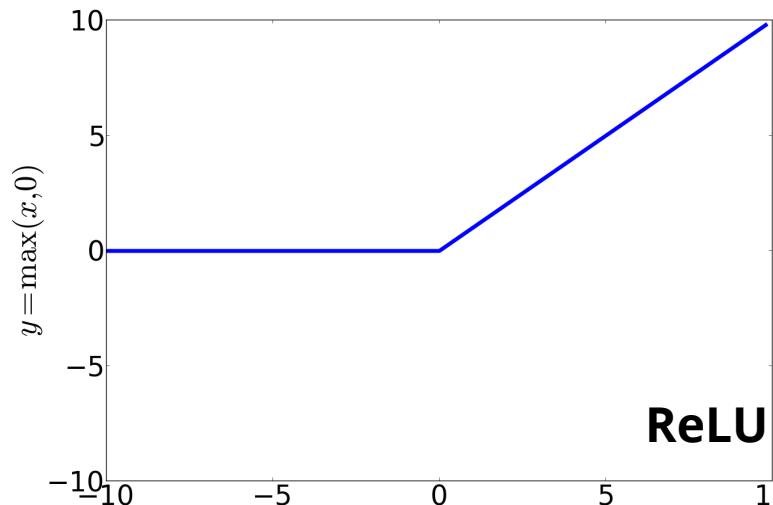
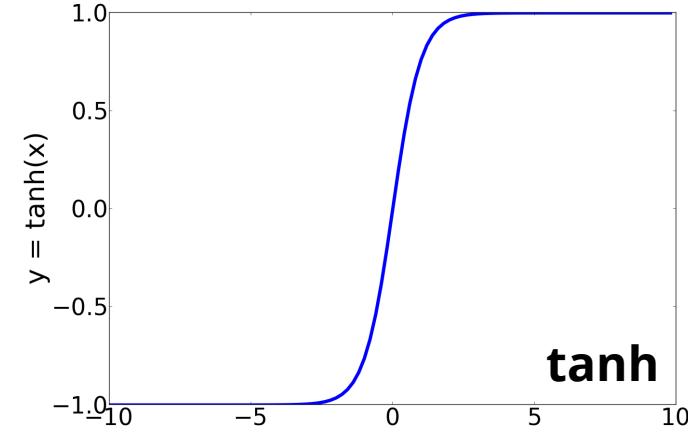
The **rectified linear unit ReLU**



# Activation Functions



The sigmoid function



ReLU