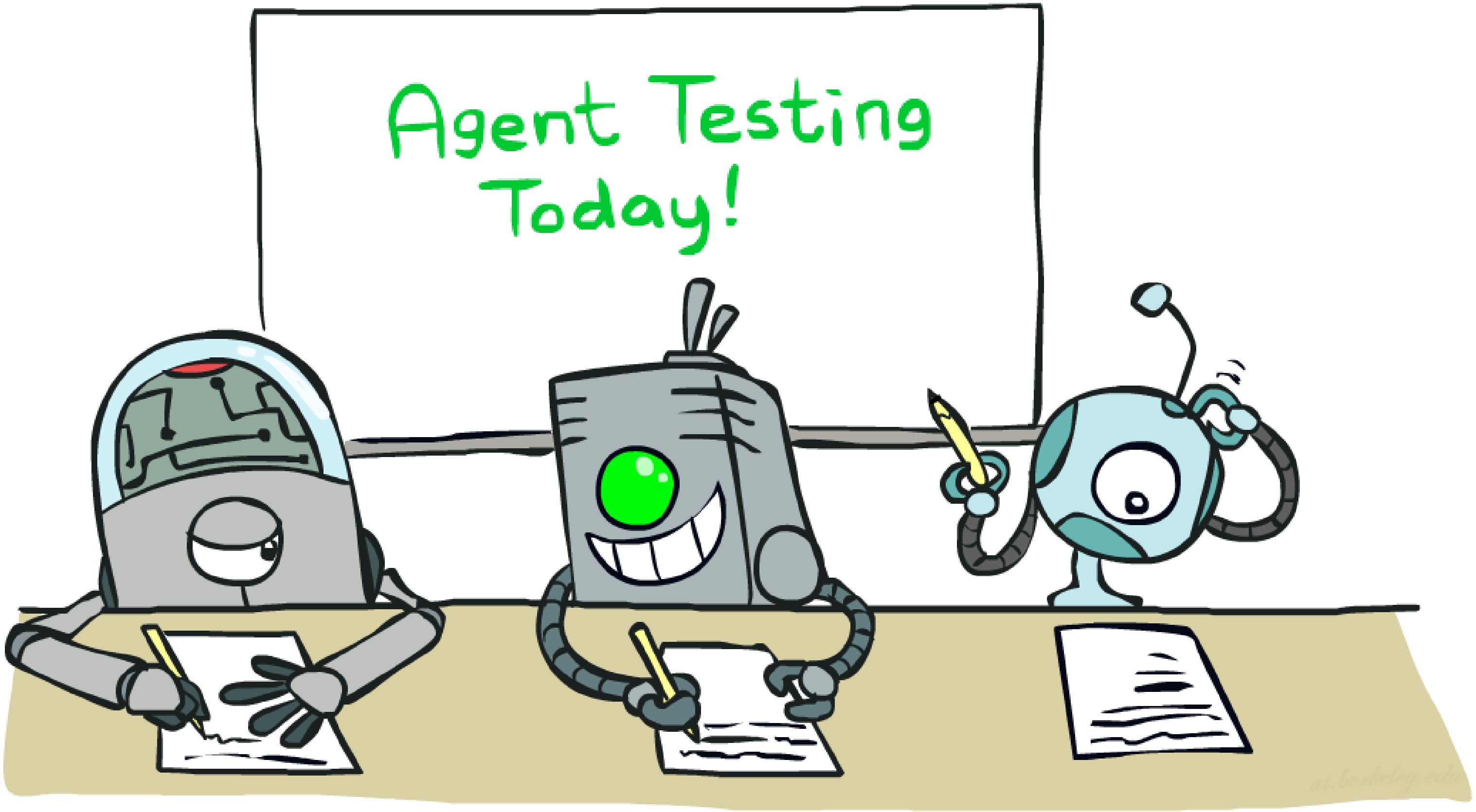
Announcement

- Programming Assignment 5
  - Due: May. 31, 11:59pm
- Homework 5
  - Due: May. 24, 11:59pm

Supervised Machine Learning



AIMA Chapter 18, 20

[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley]

Machine Learning

- Up until now: how to use a model to make optimal decisions
  - Except reinforcement learning
- Machine learning: how to acquire a model from data / experience
- Related courses
  - SI151 Optimization and Machine Learning
  - CS282 Machine Learning
  - CS280 Deep Learning

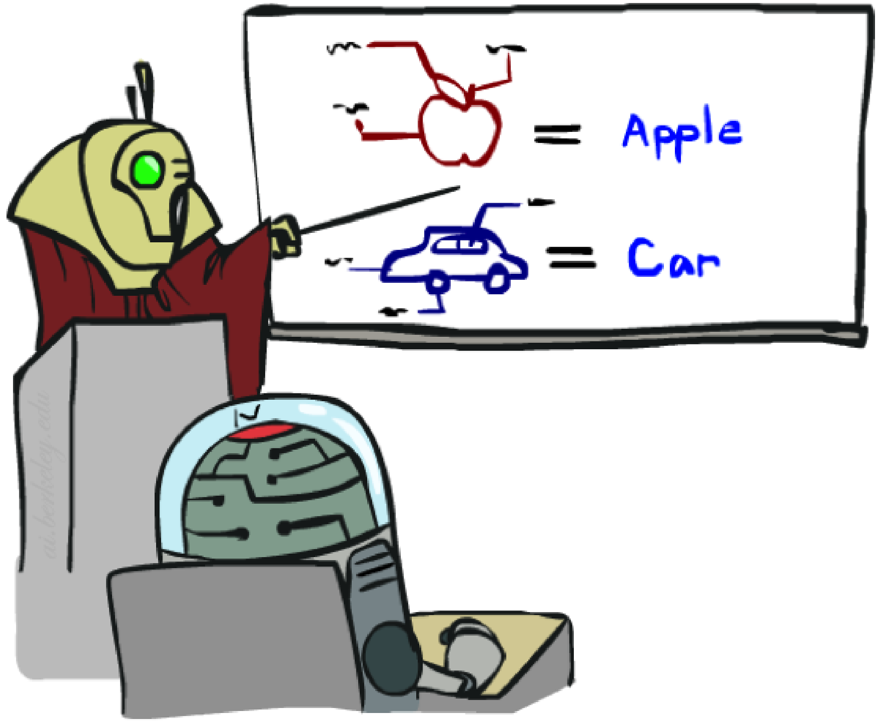
Types of Learning

- Supervised learning
  - Training data includes desired outputs
- Unsupervised learning
  - Training data does not include desired outputs
- Semi-supervised learning
  - Training data includes a few desired outputs
- Reinforcement learning
  - Rewards from sequence of actions

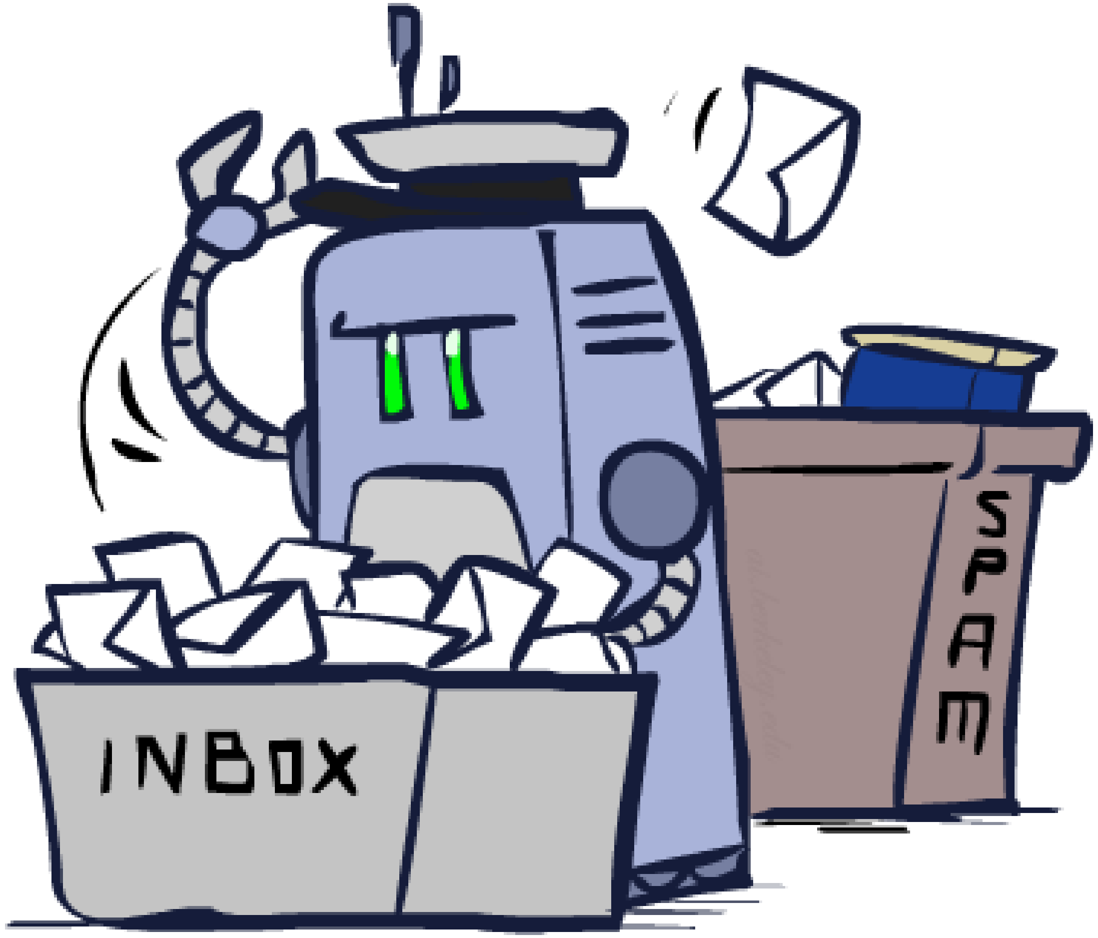
Supervised learning

- To learn an unknown *target function* f
- Input: a *training set* of *labeled examples* (x_j, y_j) where $y_j = f(x_j)$
- Output: *hypothesis* h that is "close" to f

- Types of supervised learning
  - Classification = learning f with discrete output value
  - Regression = learning f with real-valued output value
  - Structured prediction = learning f with structured output



Classification



Example: Spam Filter

- Input: an email
- Output: spam/ham

- Setup:
  - Get a large collection of example emails, each labeled "spam" or "ham" (by hand)
  - 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 virture of its nature as being utterly confidencial 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

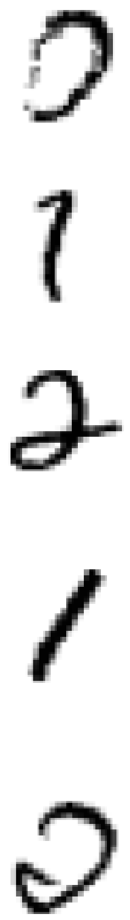
> Ok, Iknow this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

Example: Digit Recognition

- Input: images / pixel grids
- Output: a digit 0-9

- Setup:
  - Get a large collection of example images, each labeled with a digit
  - Want to learn to predict labels of new, future digit images

- Features: The attributes used to make the digit decision
  - Pixels: (6,8)=ON
  - Shape Patterns: NumComponents, AspectRatio, NumLoops
  - …



0

1

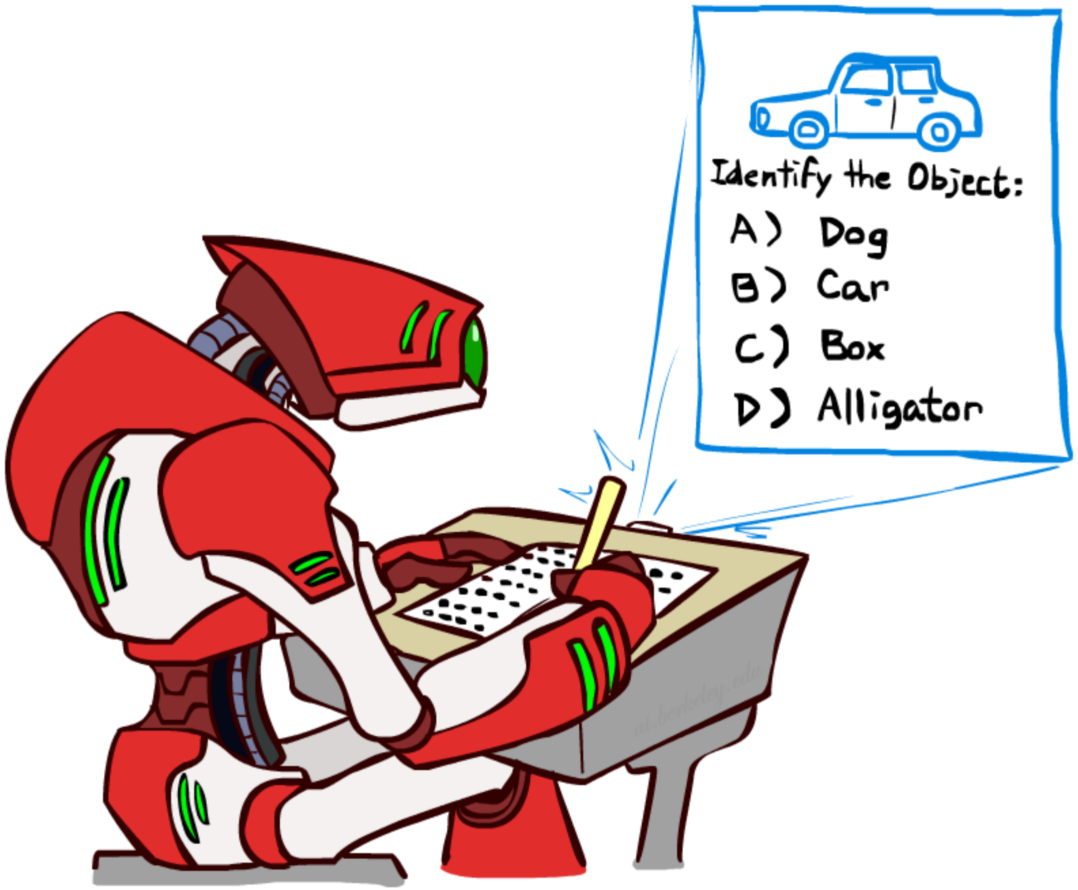
2

1

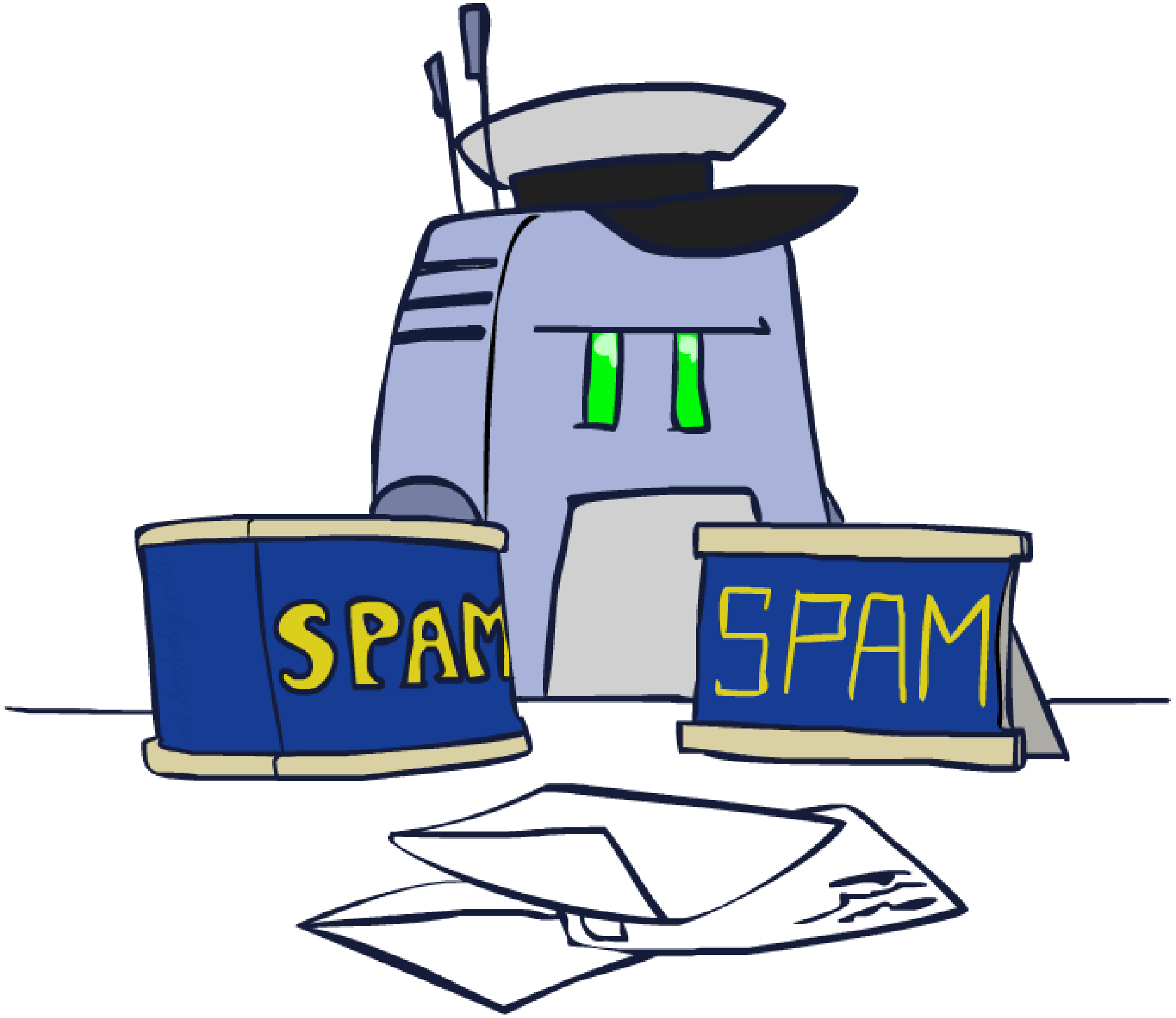
??

Other Classification Tasks

- Medical diagnosis
  - input: symptoms
  - output: disease
- Automatic essay grading
  - input: document
  - output: grades
- Fraud detection
  - input: account activity
  - output: fraud / no fraud
- Email routing
  - input: customer complaint email
  - output: which department needs to ignore this email
- Fruit and vegetable inspection
  - input: image (or gas analysis)
  - output: moldy or OK
- … many more



Naïve Bayes Classifier



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?

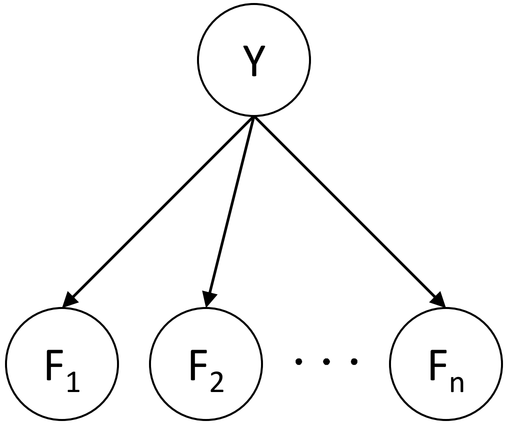
Naïve Bayes for Digits

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

- Simple digit recognition version:
  - One feature (variable) F_{ij} for each grid position <i,j>
  - 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.

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

  - Here: lots of features, each is binary valued



- Naïve Bayes model: $P(Y|F_{0,0} \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$

- What do we need to learn?

General Naïve Bayes

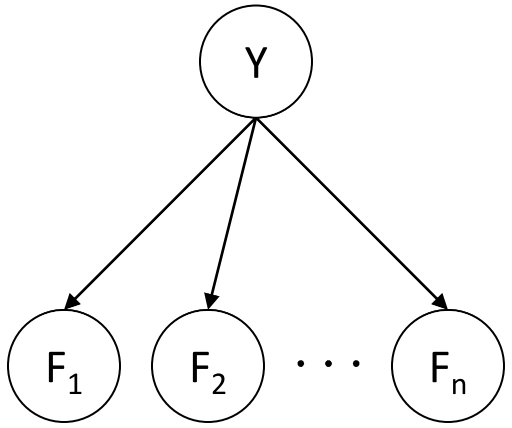
- A general Naive Bayes model:

|Y| parameters

$$P(\mathsf{Y}, \mathsf{F}_1 \dots \mathsf{F}_n) = \quad P(\mathsf{Y}) \prod_i P(\mathsf{F}_i | \mathsf{Y})$$

|Y| x $|F|^n$ values

n x |F| x |Y| parameters



- We only have to specify how each feature depends on the class
- Total number of parameters is *linear* in n
- 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

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

$$\xrightarrow{+} \quad P(f_1 \ldots f_n)$$

  - Step 2: normalization

$$\Downarrow$$

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

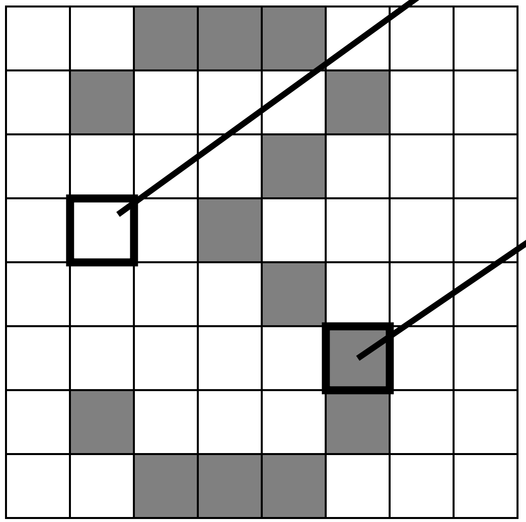
General Naïve Bayes

- 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 $P(Y|F_1 \ldots F_n)$
    - 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 θ
    - 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



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

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

$P(F_{5,5} = on|Y)$

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 positon i

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

Word at position i, not i^{th} word in the dictionary!

  - Usually, each variable gets its own conditional probability distribution P(F|Y)
  - Here
    - Each position is identically distributed
    - All positions share the same conditional probabilities P(W|Y)
  - Called "bag-of-words" because model is insensitive to word order or reordering

Example: Spam Filtering

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

$P(Y)$

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
...	

Spam Example

Log product of probabilities

$P(Y)$

$P(W_1|Y)$

$P(W_2|Y)$

$\vdots$

Word	P(w|spam)	P(w|ham)	Tot Spam	Tot Ham
(prior)	0.33333	0.66666	-1.1	-0.4
Gary	0.00002	0.00021	-11.8	-8.9
would	0.00069	0.00084	-19.1	-16.0
you	0.00881	0.00304	-23.8	-21.8
like	0.00086	0.00083	-30.9	-28.9
to	0.01517	0.01339	-35.1	-33.2
lose	0.00008	0.00002	-44.5	-44.0
weight	0.00016	0.00002	-53.3	-55.0
while	0.00027	0.00027	-61.5	-63.2
you	0.00881	0.00304	-66.2	-69.0
sleep	0.00006	0.00001	-76.0	-80.5

P(spam | w) = 98.9

Training and Testing

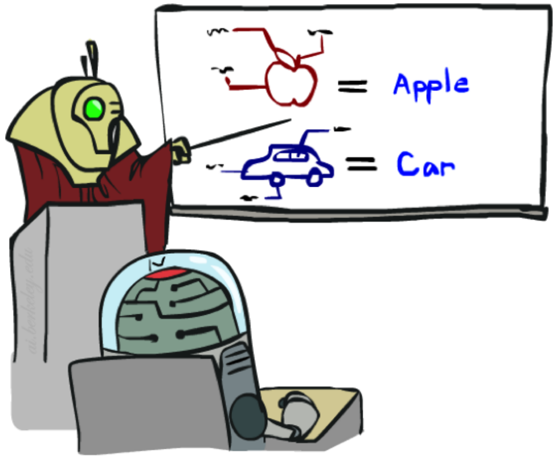
Important Concepts

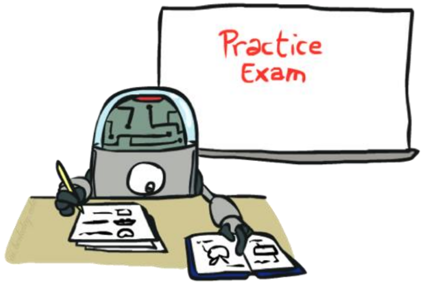
- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - Tune hyperparameters on held-out set
  - Compute accuracy of test set (fraction of instances predicted correctly)
  - Very important: never “peek” at the test set!

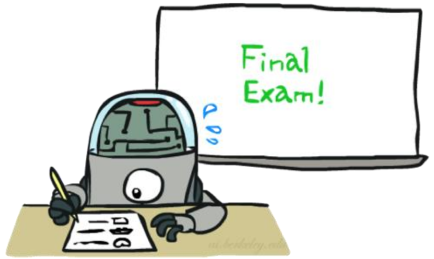
Training Data

Held-Out Data

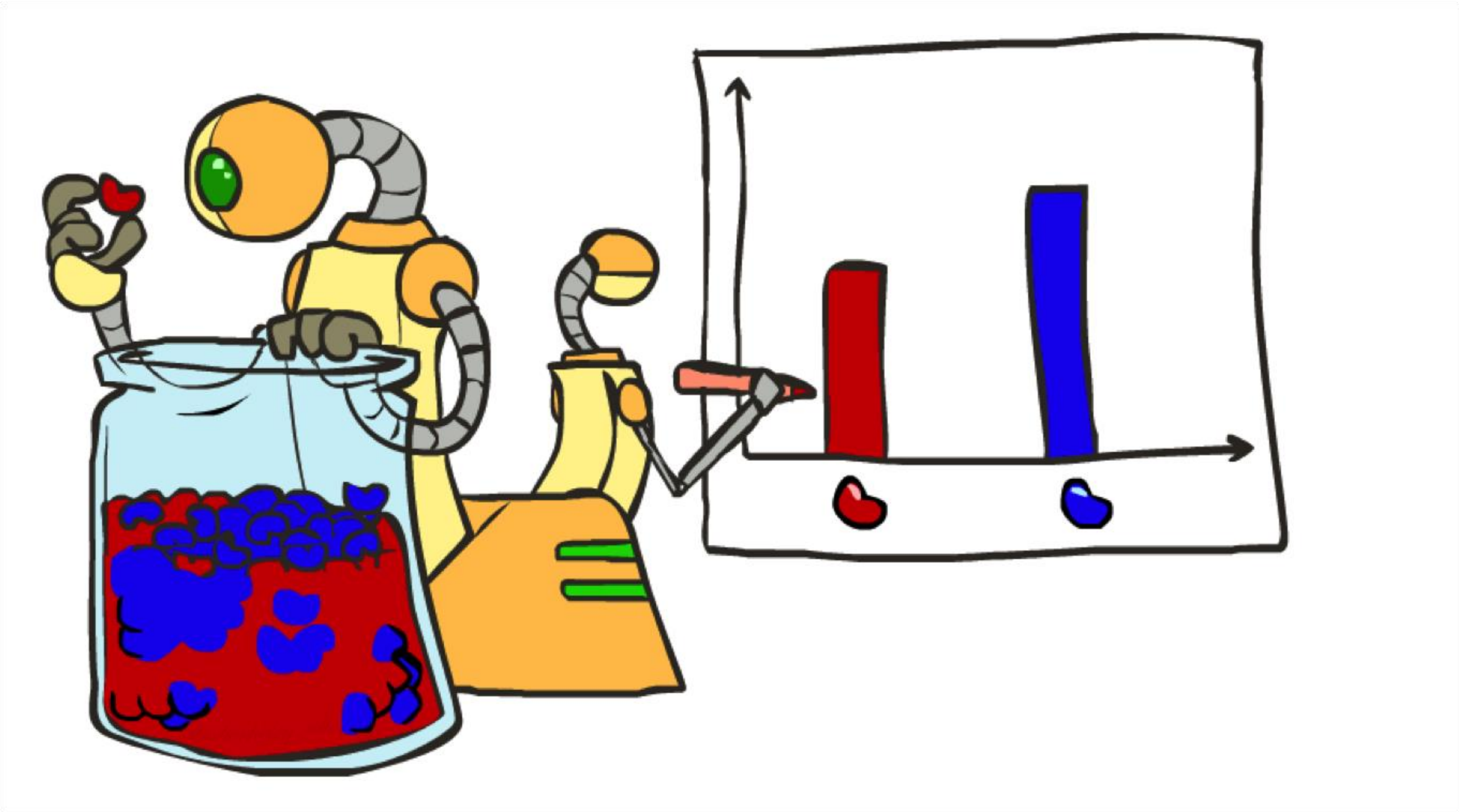
Test Data



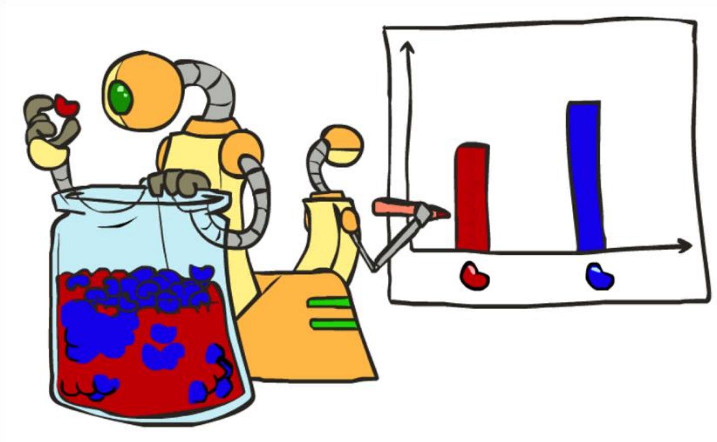




Training



Parameter Estimation



- Estimating the distribution of a random variable
- *Elicitation:* ask a human (this is hard…)
- *Empirically:* use training data (learning!)
  - For each outcome x, look at the *empirical rate* of that value

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

  - Ex:
    - We've seen 1000 words from spam emails, among which we see "money" for 50 times
    - So we set P(money | spam) = 0.05
  - This is the estimate that maximizes the *likelihood of the data*
    - Likelihood: conditional probability of the data given the parameters

Maximum Likelihood Estimation

- Coin flipping:
  - P(Heads) = θ, P(Tails) = 1-θ
- Flips are *i.i.d.*
  - Independent events
  - Identically distributed according to unknown distribution
- Sequence *D* of α_H Heads and α_T Tails

$$P(\mathcal{D} \mid \theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$$

Maximum Likelihood Estimation

- **MLE:** Choose θ to maximize probability of *D*

$$\begin{aligned}\widehat{\theta} &= \arg\max_{\theta} \quad \ln P(\mathcal{D} \mid \theta) \\ &= \arg\max_{\theta} \quad \ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}\end{aligned}$$

- Set derivative to zero, and solve!

$$\frac{d}{d\theta} \ln P(\mathcal{D} \mid \theta) = \frac{d}{d\theta} [\ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}]$$

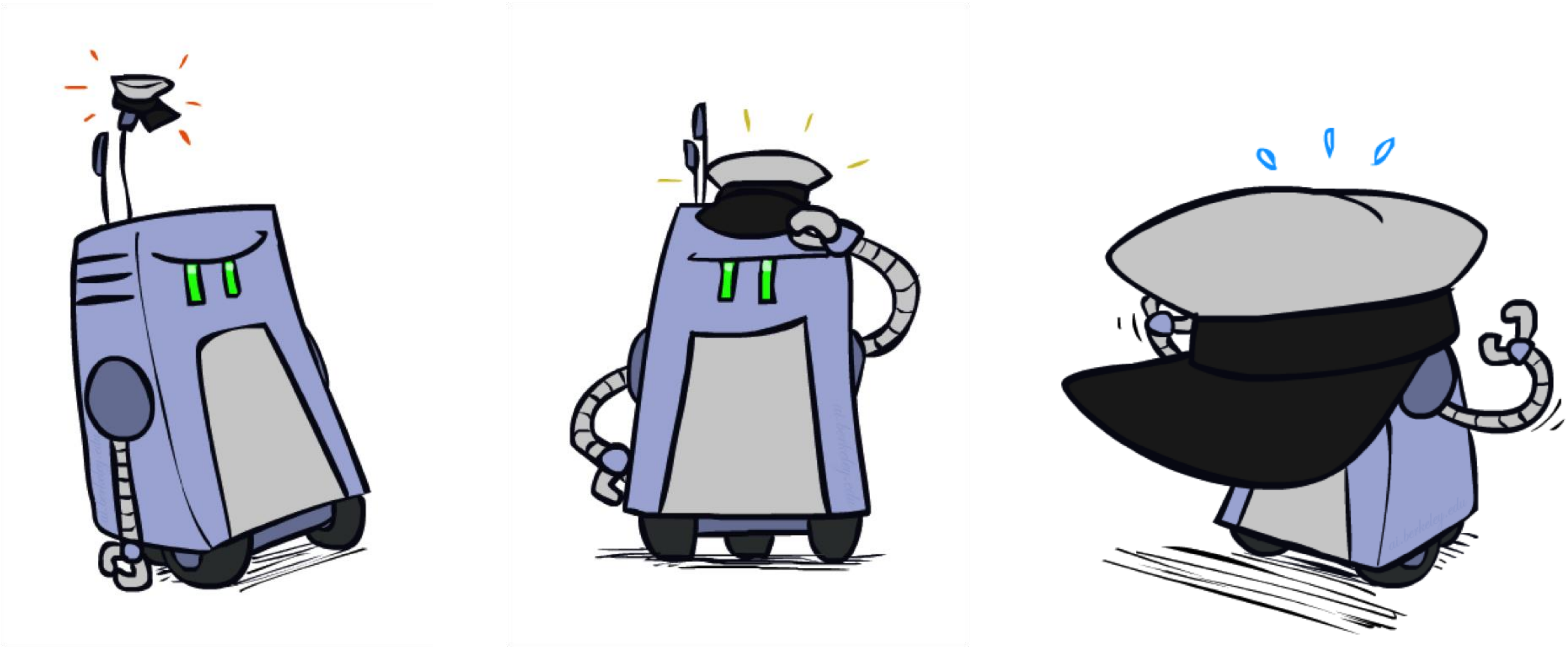
$$= \frac{d}{d\theta} [\alpha_H \ln \theta + \alpha_T \ln(1-\theta)]$$

$$= \alpha_H \frac{d}{d\theta} \ln \theta + \alpha_T \frac{d}{d\theta} \ln(1-\theta)$$

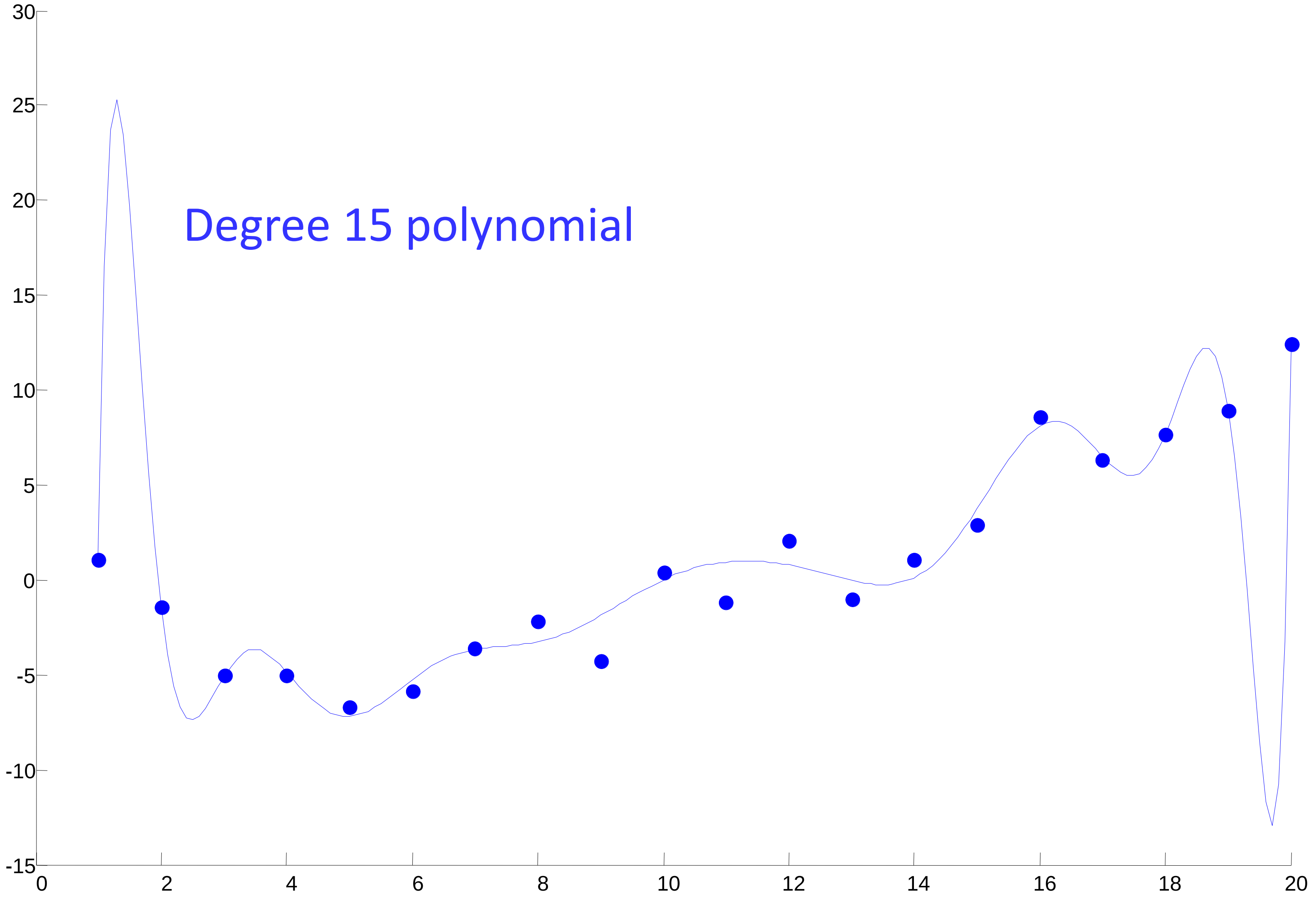
$$= \frac{\alpha_H}{\theta} - \frac{\alpha_T}{1-\theta} = 0$$

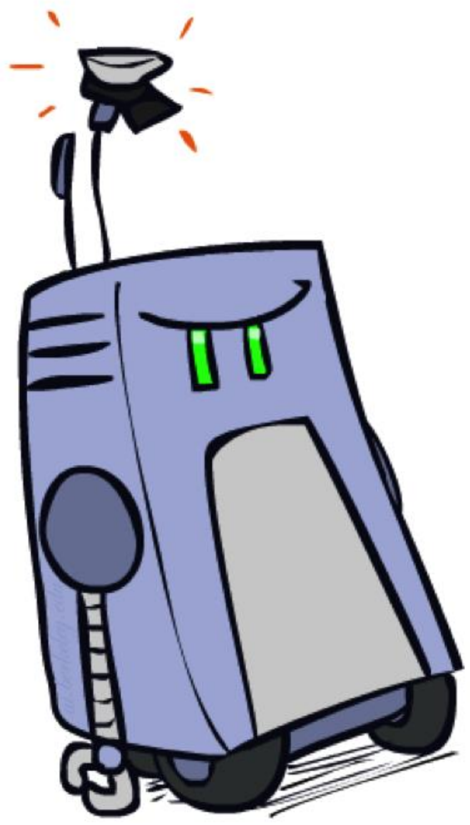
$$\boxed{\widehat{\theta}_{MLE} = \frac{\alpha_H}{\alpha_H + \alpha_T}}$$

Generalization and Overfitting



Overfitting





Example: Overfitting

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

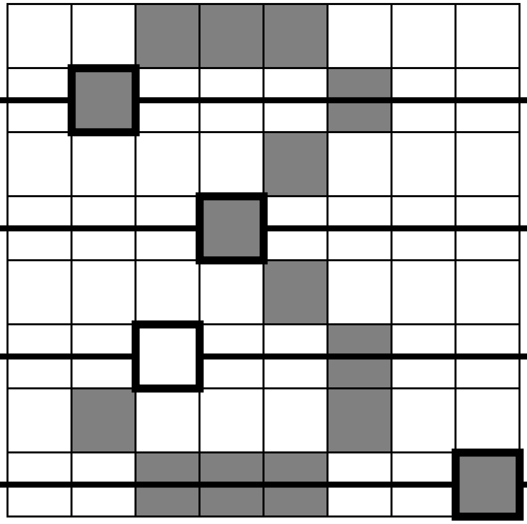
$P(C = 2) = 0.1$

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

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

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

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



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

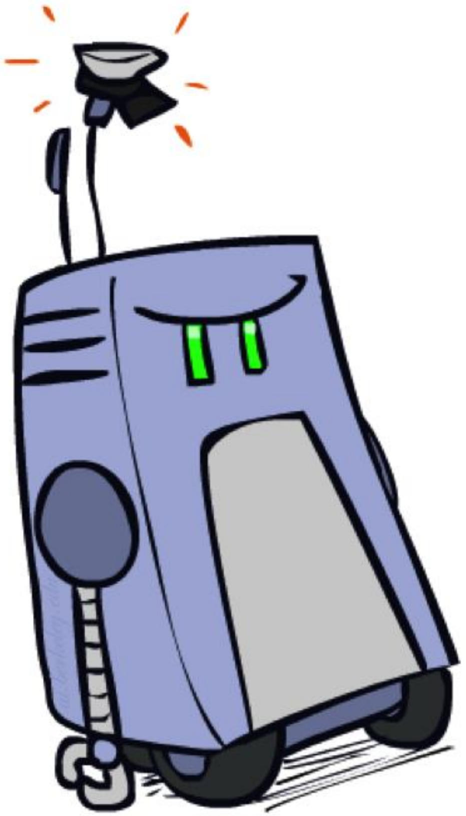
$P(C = 3) = 0.1$

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

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

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

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



2 wins!!

Example: Overfitting

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

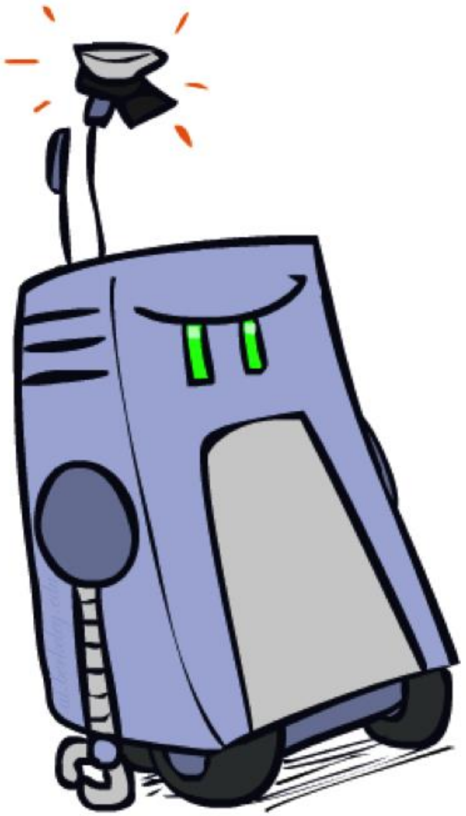
$$\frac{P(W|\text{ham})}{P(W|\text{spam})}$$

```
south-west  : inf
nation      : inf
morally     : inf
nicely      : inf
extent      : inf
seriously   : inf
...
```

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

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

What went wrong here?



Generalization and Overfitting

- Using empirical rate 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
  - Just because we never saw a word in spam emails during training doesn't mean we won't see it at test time
  - Therefore, we can't give unseen events zero probability
  - More generally, rates in the training data may not exactly match rates at test time

Generalization and Overfitting

- Overfitting: learn to fit the training data very closely, but fit the test data poorly
  - Generalization: try to fit the test data as well
- Why does overfitting occur?
  - Training data is not representative of the true data distribution
    - Too few training samples
    - Training data is noisy
  - Too many attributes, some of them irrelevant to the classification task
  - The model is too expressive
    - Ex: the model is capable of memorizing all the spam emails in the training set

Generalization and Overfitting

- Avoid overfitting
  - Acquire more training data (not always possible)
  - Remove irrelevant attributes (not always possible)
  - Limit the model expressiveness by regularization, early stopping, pruning, etc.

- In our previous example, we may smooth the empirical rate to improve generalization

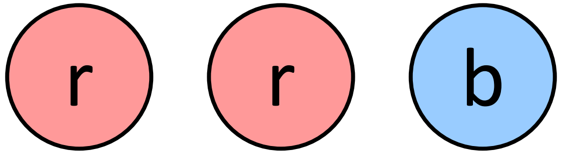
Laplace Smoothing

- Laplace's estimate:
  - Pretend you saw every outcome once more than you actually did

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

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

  - Can derive this estimate with *Dirichlet priors*



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

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

Laplace Smoothing

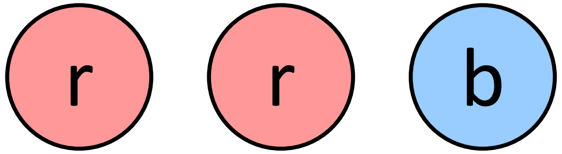
- Laplace's estimate (extended):
  - Pretend you saw every outcome k extra times

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

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

- Laplace for conditionals:
  - Smooth each condition independently:

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



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

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

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

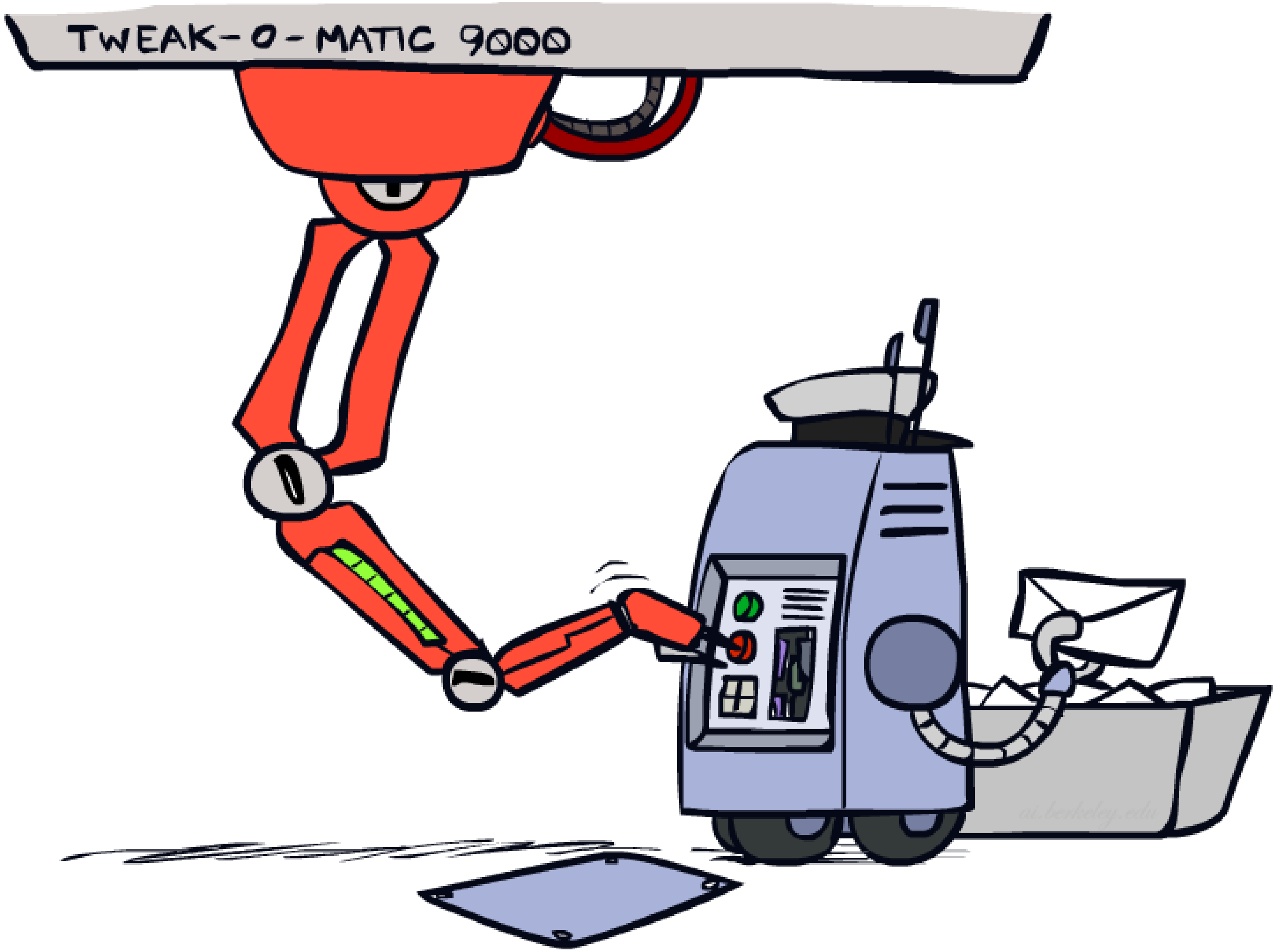
Linear Interpolation

- In practice, Laplace often performs poorly for P(X|Y):
  - When |X| is very large
  - When |Y| is very large

- 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)

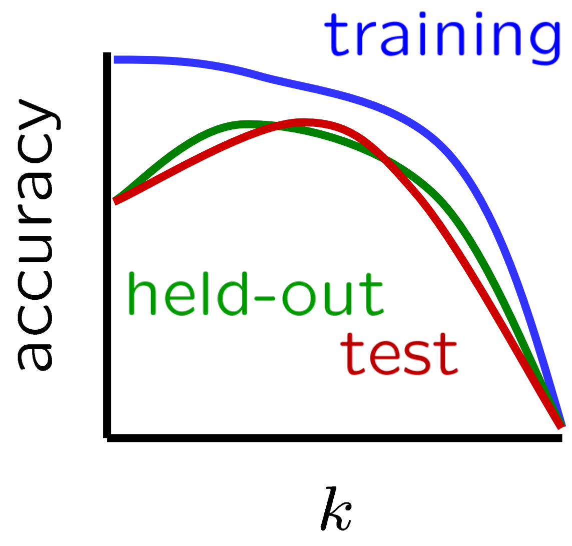
$$P_{LIN}(x|y) = \alpha \hat{P}(x|y) + (1.0 - \alpha)\hat{P}(x)$$

Tuning

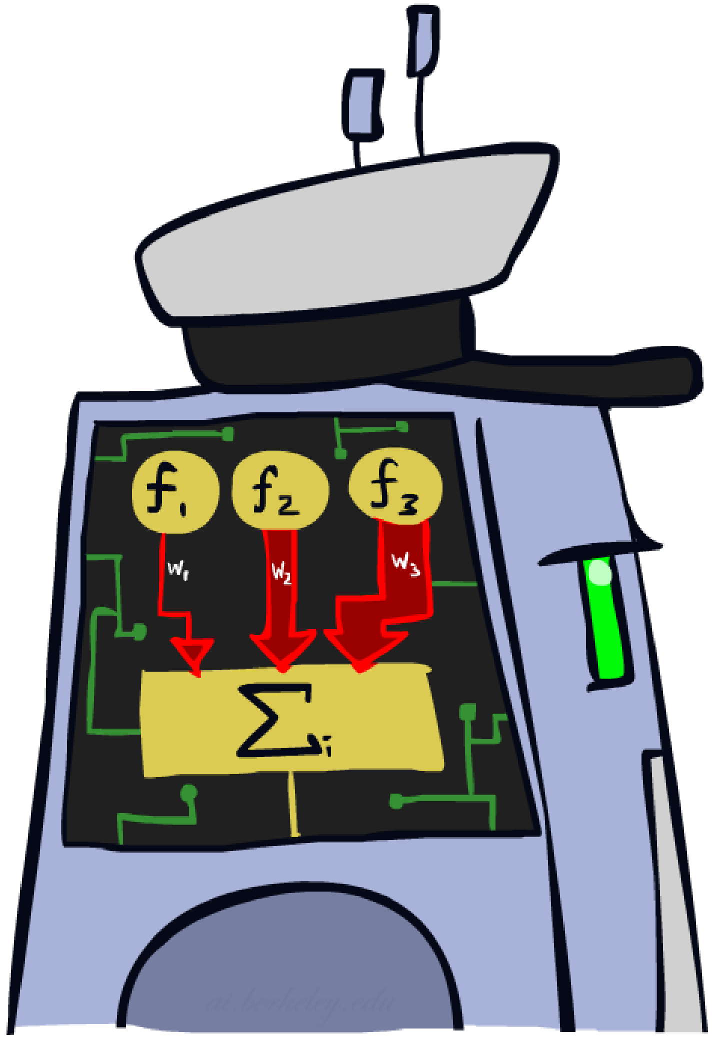


Tuning on Held-Out Data

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



Linear Classifiers



Feature Vectors

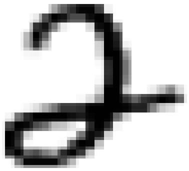
x $f(x)$ y

```
Hello,

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

```
# free        : 2
YOUR_NAME     : 0
MISSPELLED    : 2
FROM_FRIEND   : 0
...
```

SPAM
or
+

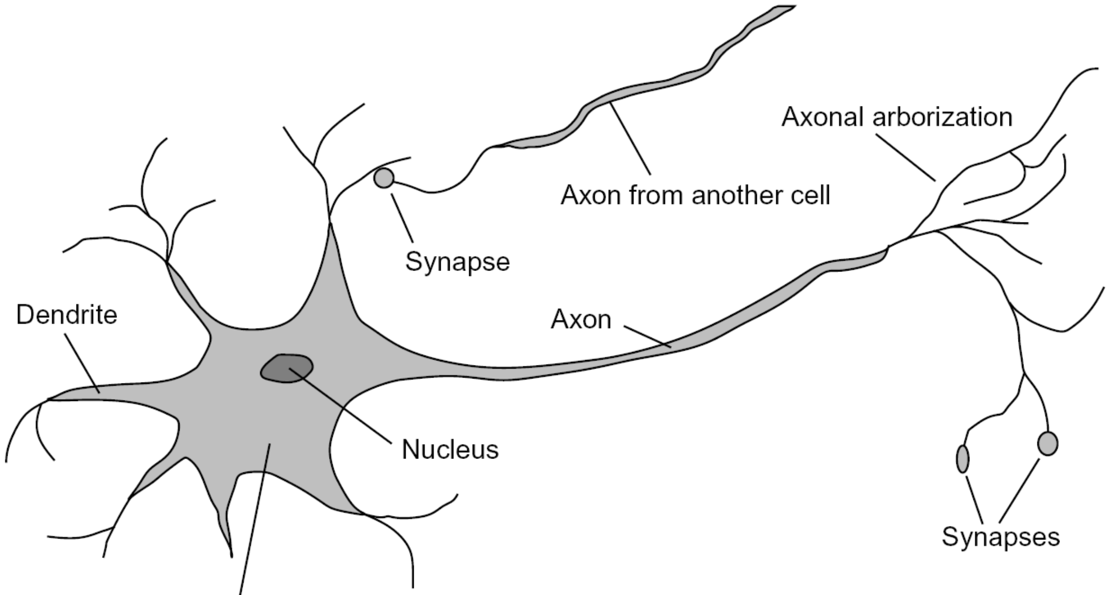


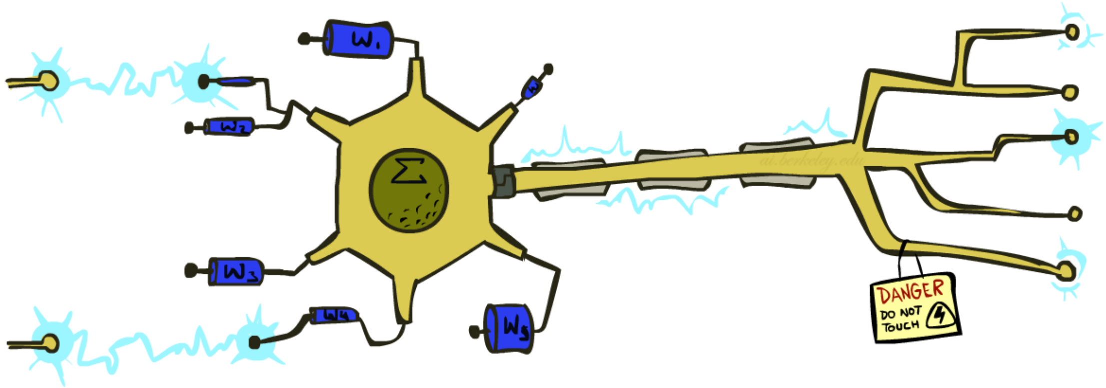
```
PIXEL-7,12    : 1
PIXEL-7,13    : 0
...
NUM_LOOPS     : 1
...
```

“2”

Some (Simplified) Biology

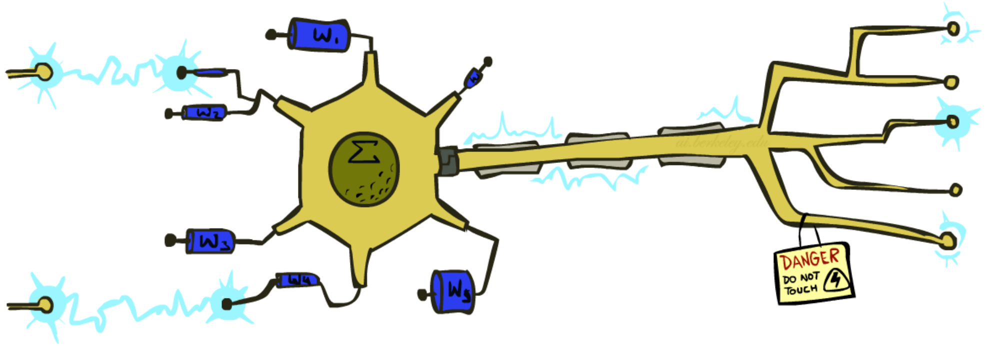
- Very loose inspiration: human neurons





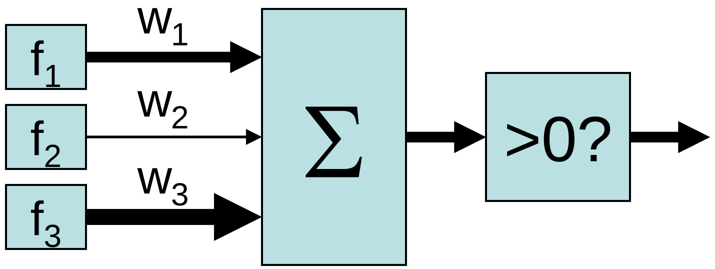
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



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

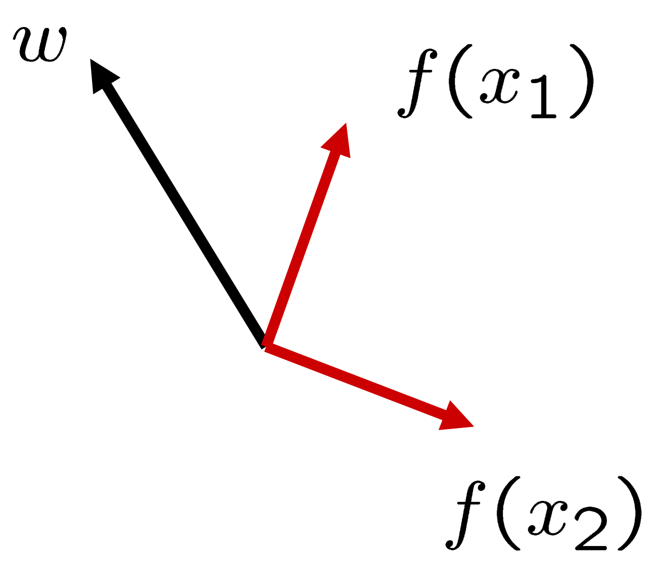
- Binary case: if the activation is:
  - Positive, output +1
  - Negative, output -1



Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples

```
# free       : 4
YOUR_NAME    :-1
MISSPELLED   : 1
FROM_FRIEND  :-3
...
```




```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

```
# free       : 0
YOUR_NAME    : 1
MISSPELLED   : 1
FROM_FRIEND  : 1
...
```

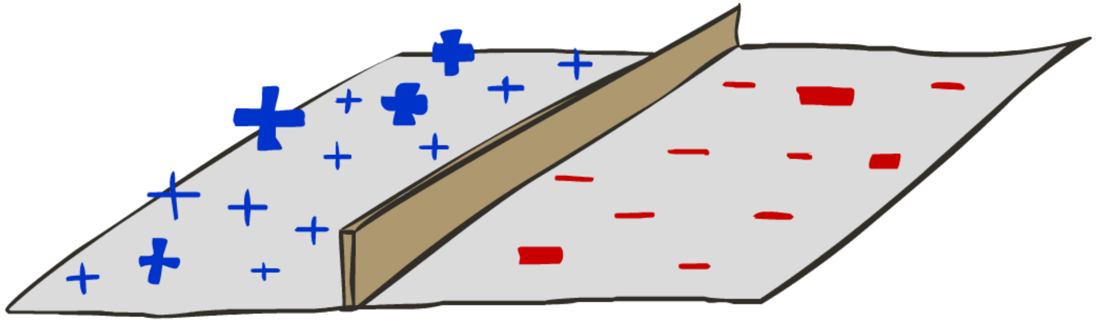
Dot product $w \cdot f$ *positive means the positive class*

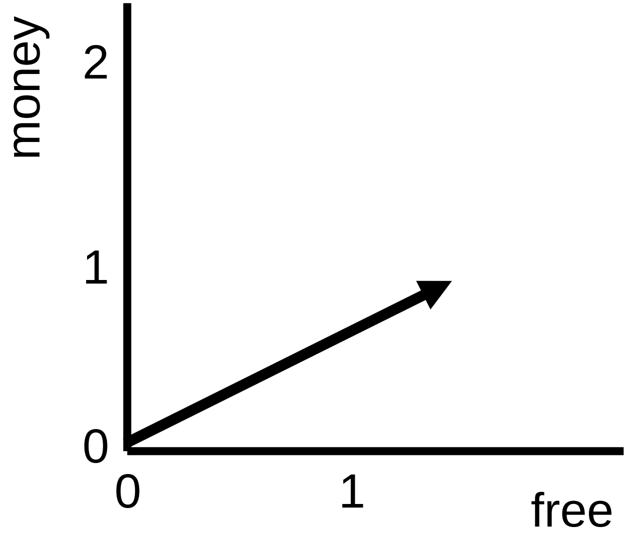
Binary Decision Rule

- 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

$$w$$

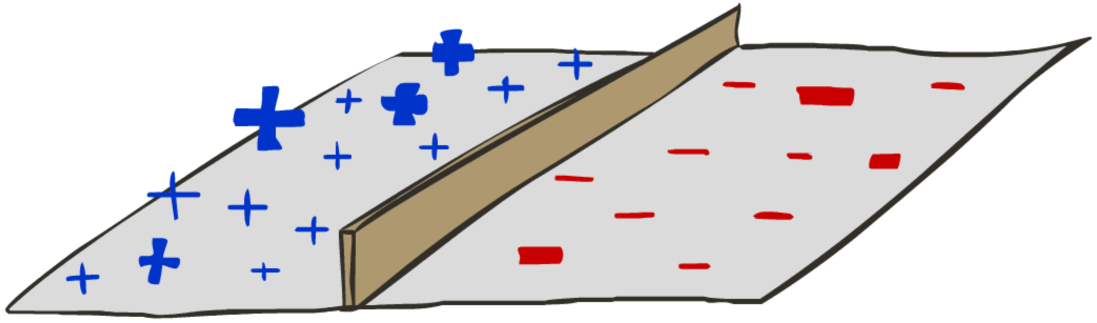
```
BIAS  : -3
free  :  4
money :  2
...
```





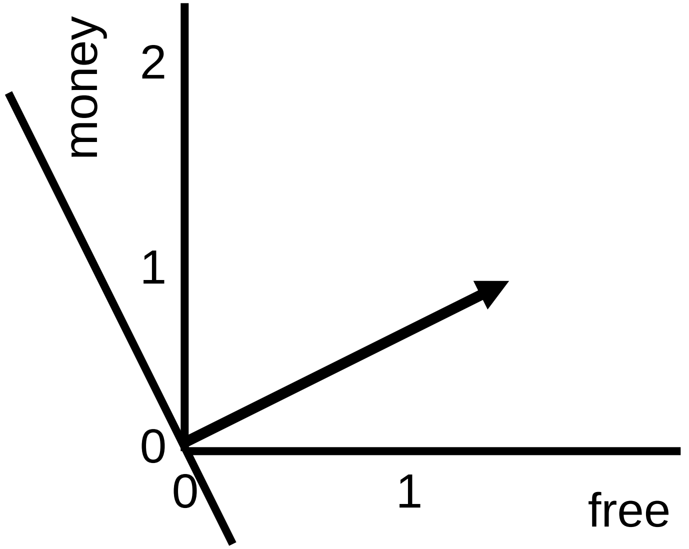
Binary Decision Rule

- 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



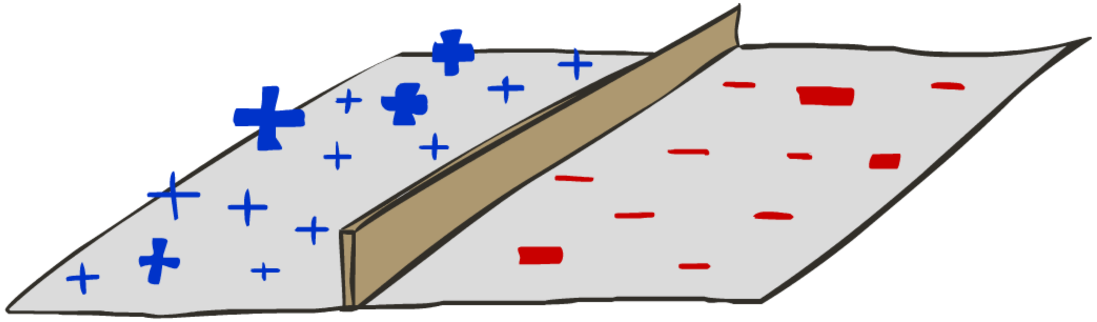
w

```
BIAS  : -3
free  :  4
money :  2
...
```



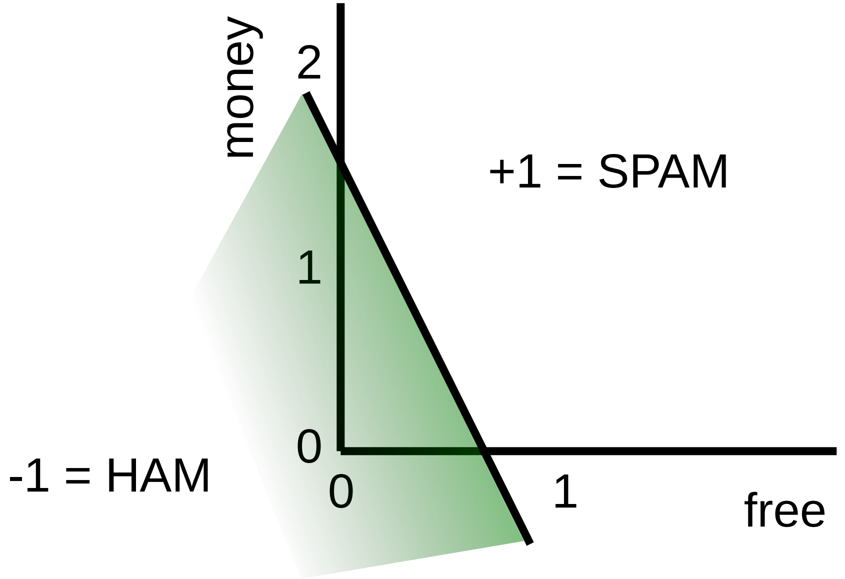
Binary Decision Rule

- 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



$$w$$

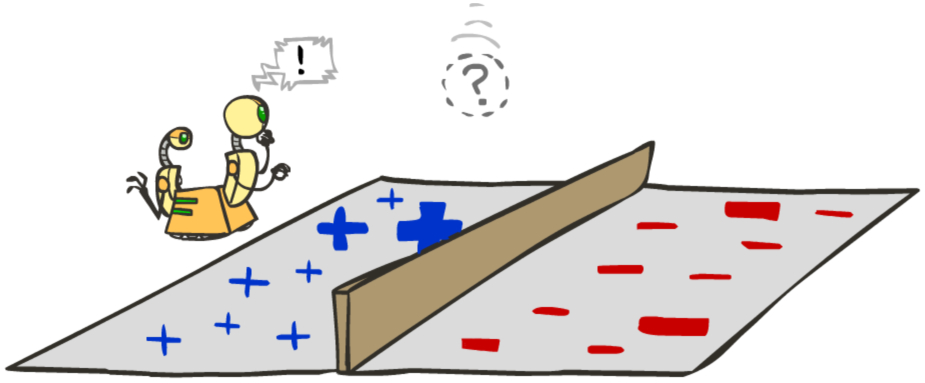
```
BIAS  : -3
free  :  4
money :  2
...
```



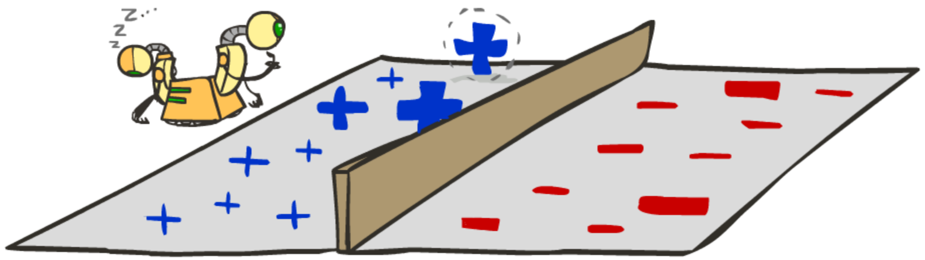
$$f \cdot w = 0$$

Learning: Binary Perceptron

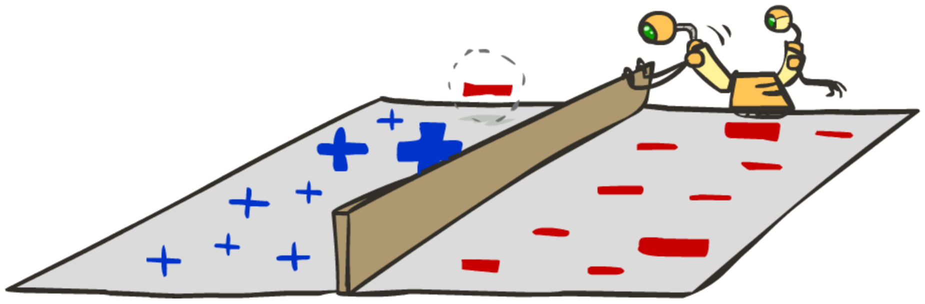
- Start with weights = 0
- For each training instance:
  - Classify with current weights



  - If correct (i.e., y=y*), no change!



  - If wrong: adjust the weight vector



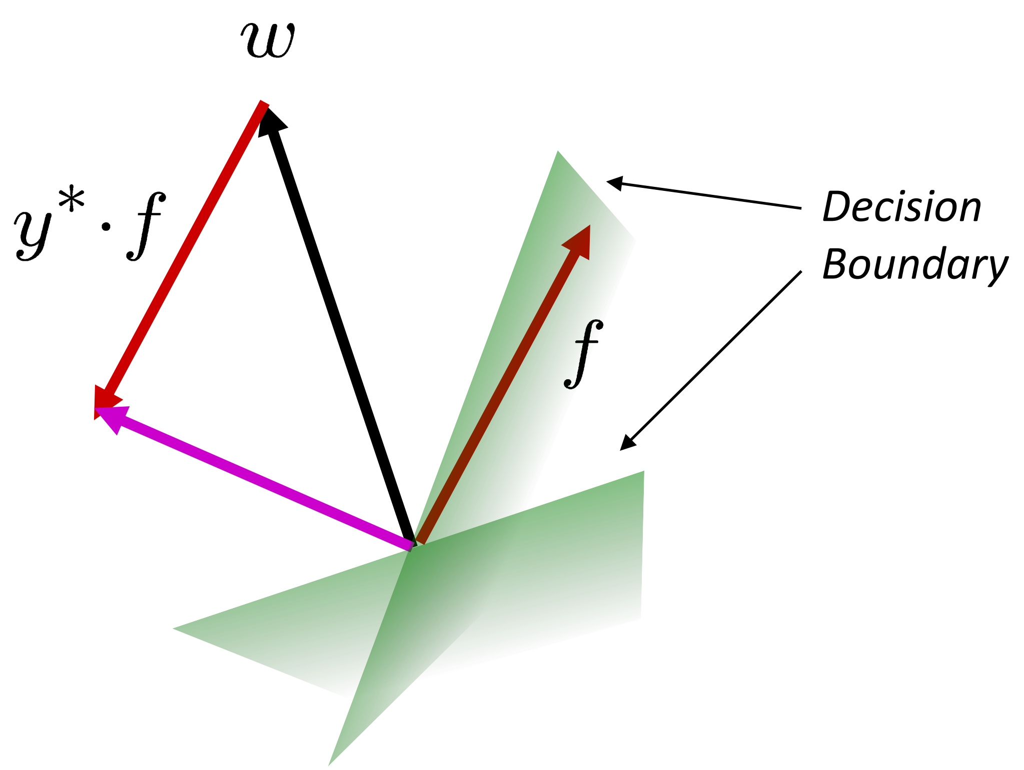
Learning: Binary Perceptron

- 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.

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

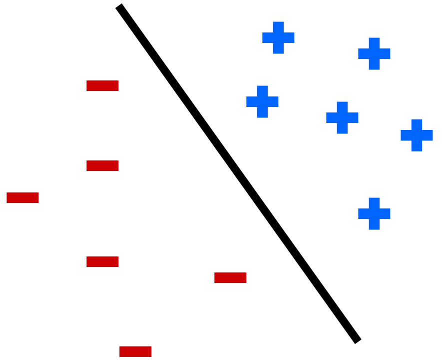


Properties of Perceptrons

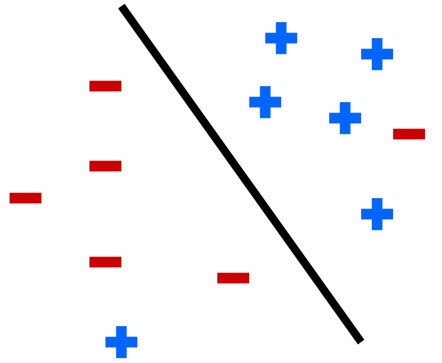
- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

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

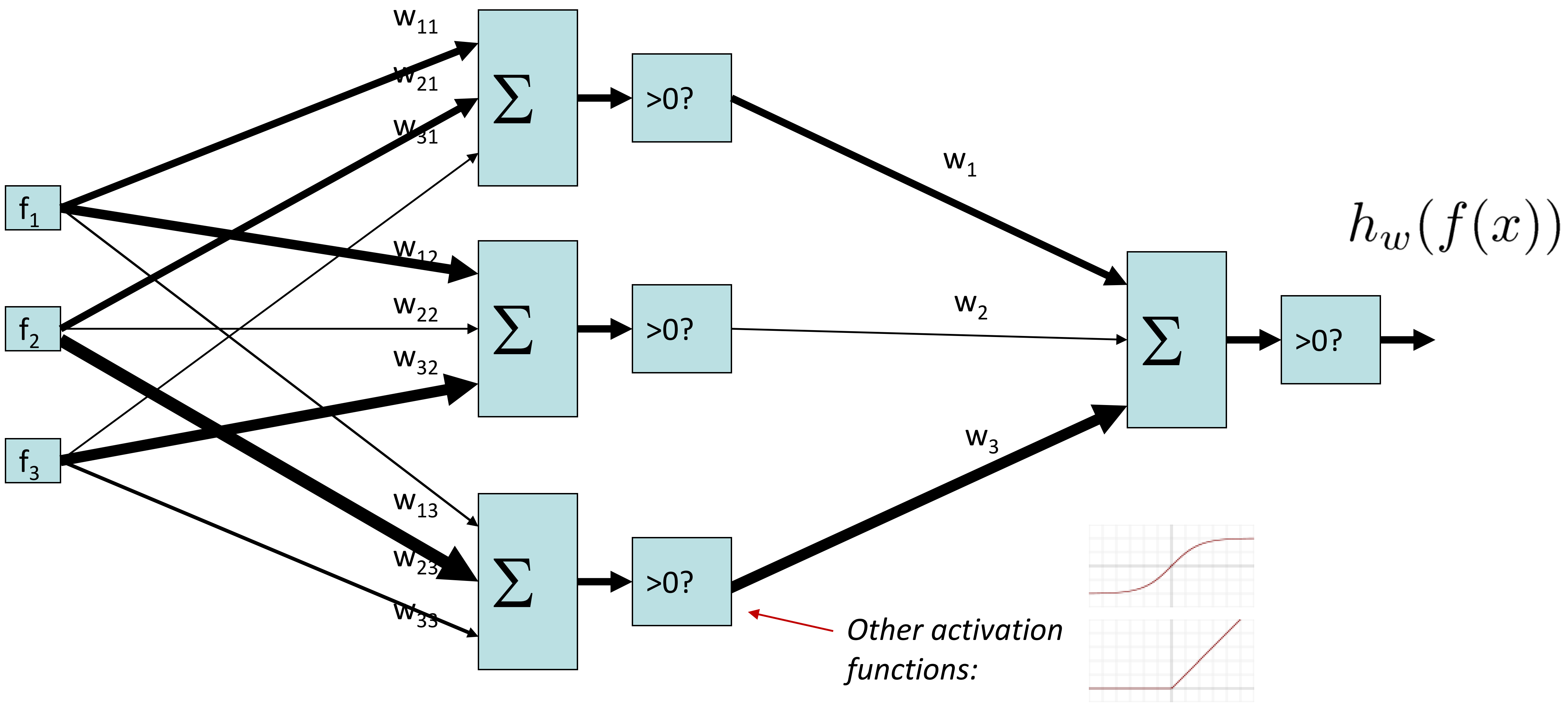
Separable



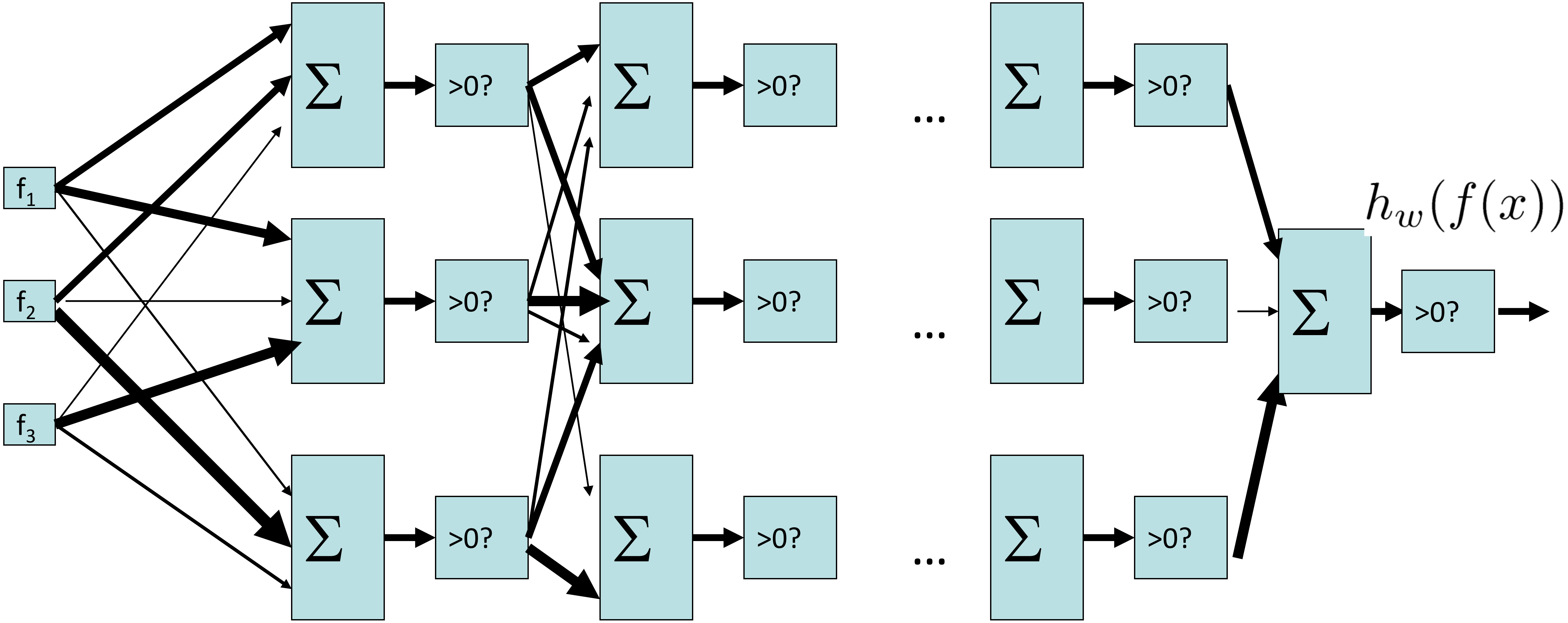
Non-Separable



Two-Layer Perceptron Network

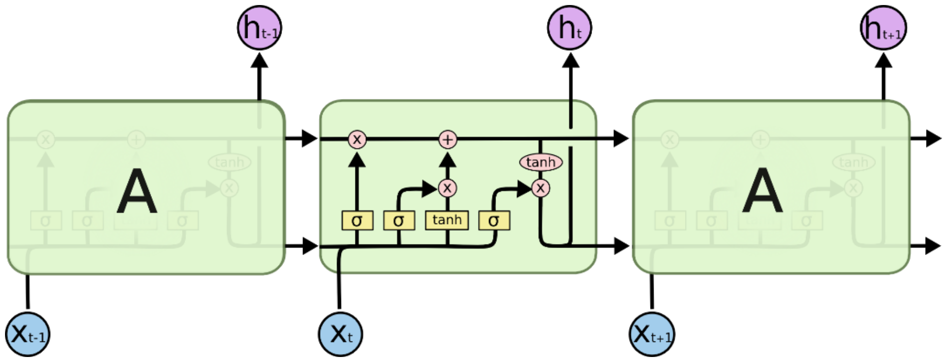


Deep Neural Network



1-page Overview of Deep Learning

- Deep Learning
  - A large number of layers of neural networks
    - Ex. 1000 layers in ResNet
  - More complicated connections between layers
    - Ex. LSTM



1-page Overview of Deep Learning

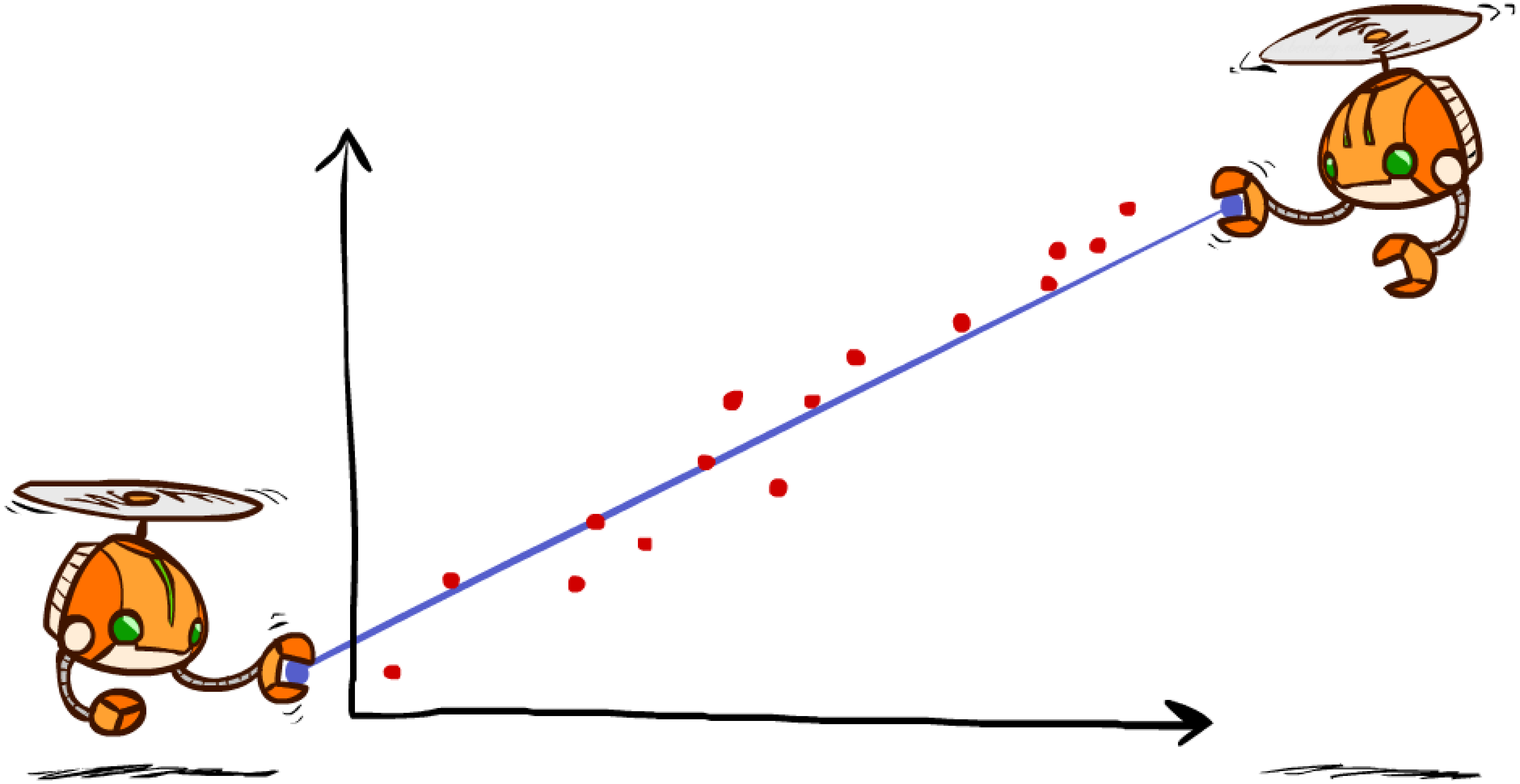
Take CS280 Deep Learning!

- Deep Learning
  - A large number of layers of neural networks
    - Ex. 1000 layers in ResNet
  - More complicated connections between layers
    - Ex. LSTM
  - Lots of new techniques and tricks
    - ReLU, Dropout, Batch Normalization, Adam, …
  - Big data
    - ImageNet (2009): 14 million images
    - NMT (a 2019 paper): 25 billion sentence pairs
  - GPU parallelization
- Performance: superior to human experts in some tasks

More classification methods

- Naive Bayes
- Perceptron / Neural networks
- Decision trees / Random forest
- Support Vector Machines
- Nearest neighbors
- Model ensembles: bagging, boosting, etc.
- ……

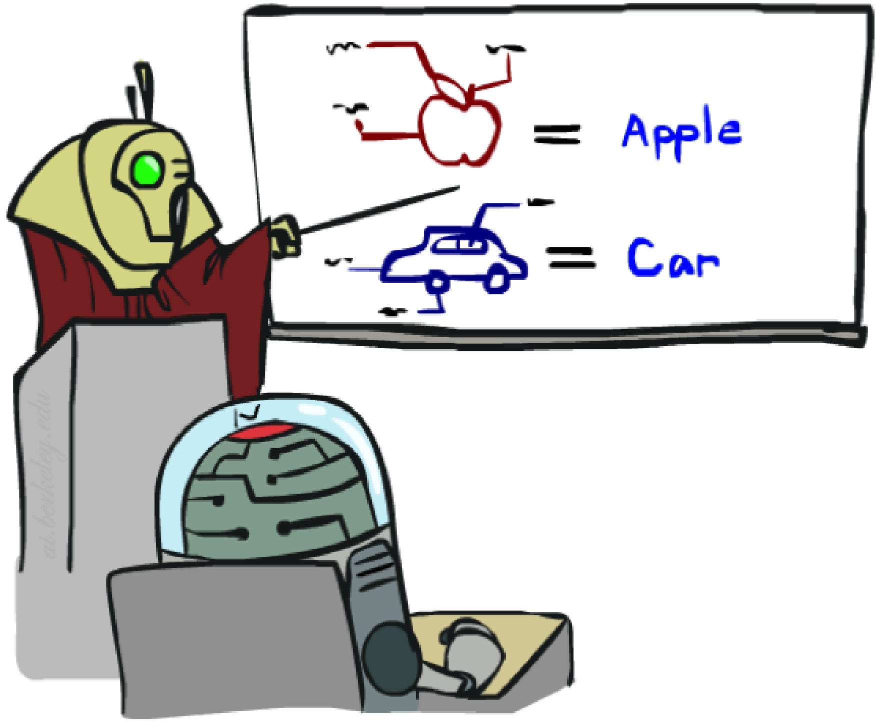
Regression



Supervised learning

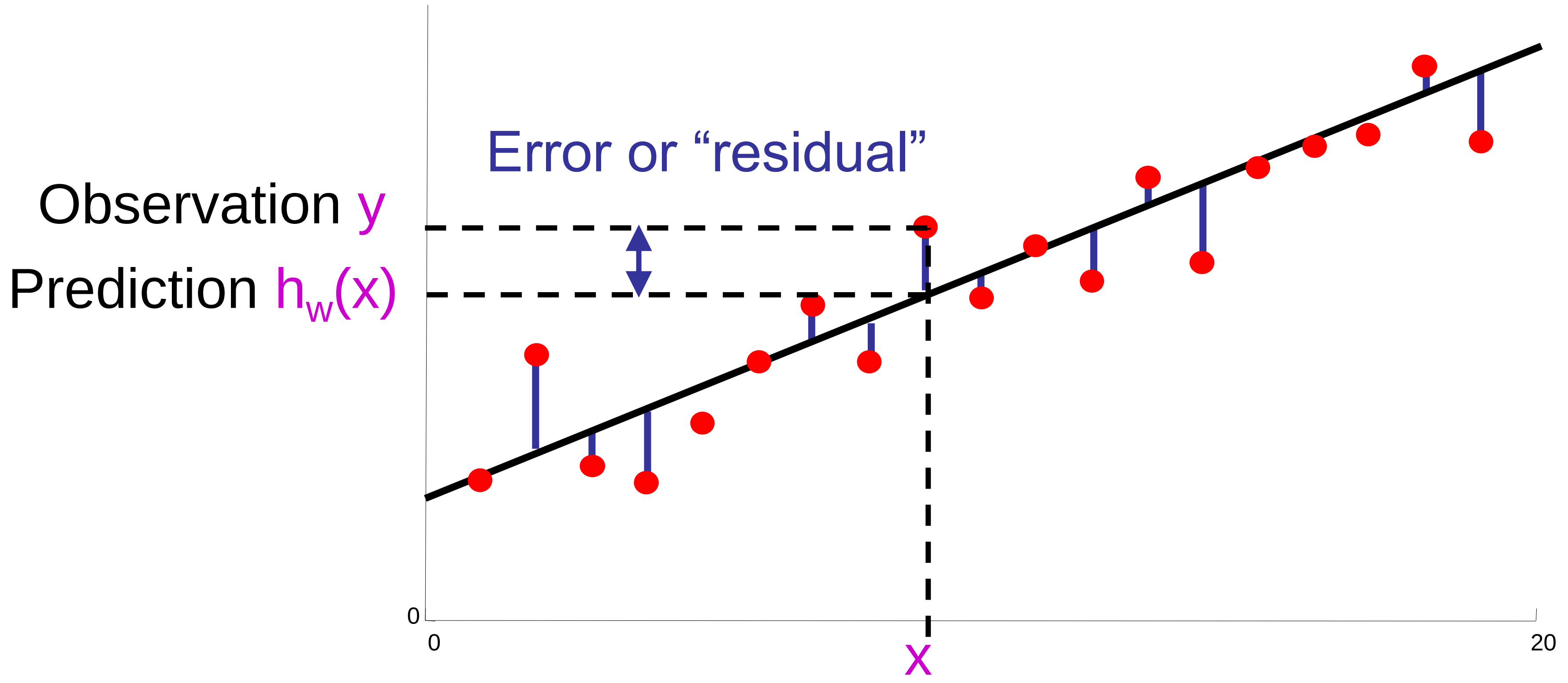
- To learn an unknown ***target function*** f
- Input: a ***training set*** of ***labeled examples*** (x_j, y_j) where $y_j = f(x_j)$
- Output: ***hypothesis*** h that is "close" to f

- Two types of supervised learning
  - Classification = learning f with discrete output value
  - Regression = learning f with real-valued output value



Linear Regression

Prediction: $h_w(x) = w_0 + w_1 x$



Error on one instance: $|y - h_w(x)|$

Least squares: Minimizing squared error

- L2 loss function: sum of squared errors over all examples

$$L(\boldsymbol{w}) = \sum_i \left(y_i - h_w(\boldsymbol{x}_i)\right)^2 = \sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2$$

- We want the weights w* that minimize loss
- Analytical solution: at w* the derivative of loss w.r.t. each weight is zero
  - **X** is the data matrix (all the data, one example per row); **y** is the vector of labels
  - $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

Least squares: Minimizing squared error

- $J(w) = \|Xw - y\|_2^2$ $\quad\quad\quad\quad$ $\nabla J(w) = 2X^T(\mathrm{X}w - y)$

$$X^T\mathrm{X}w - X^T y = 0$$

$$X^T\mathrm{X}w = X^T y$$

$$w = (X^T\mathrm{X})^{+}X^T y$$

Regularized Regression

- Overfitting is also possible in regression
  - Extreme case: *n* features, *n* training examples
- Regularization can be used to alleviate overfitting

- LASSO (Least Absolute Shrinkage and Selection Operator)

$$L(\boldsymbol{w}) = \sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 + \lambda \sum_k |w_k|$$

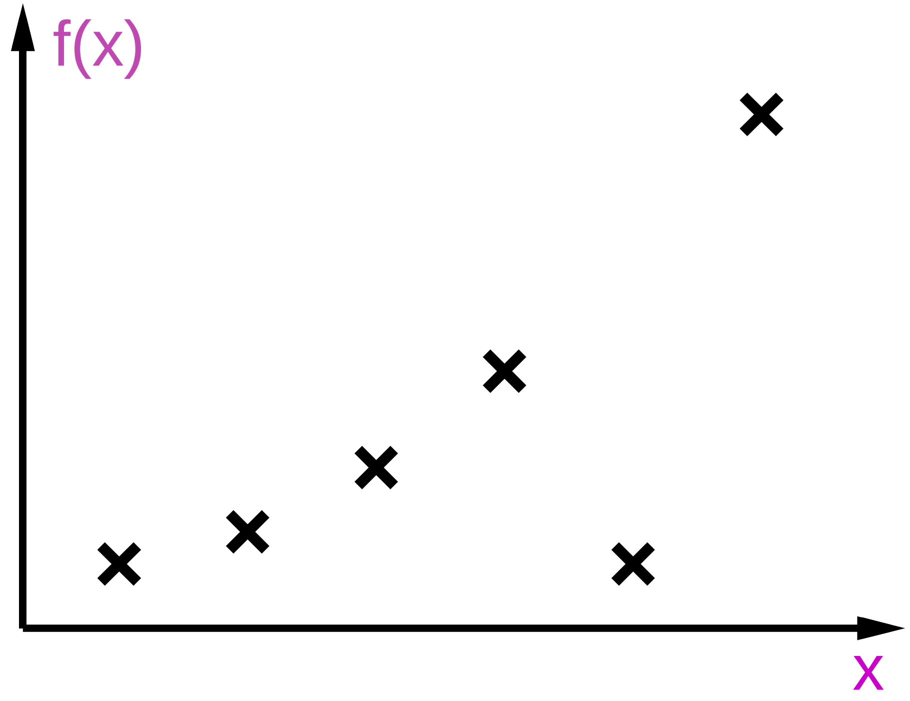
- Ridge Regression

$$L(\boldsymbol{w}) = \sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 + \lambda \sum_k w_k^2$$

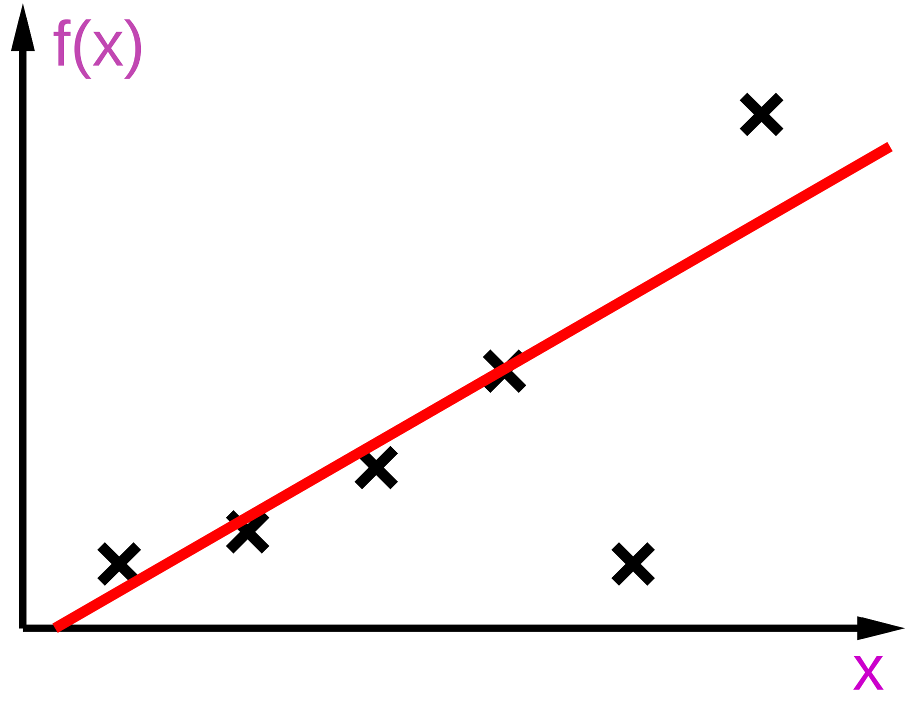
Non-linear least squares

- No closed-form solution in general
- Numerical algorithms are typically used
  - Choose initial values for the parameters and then refine the parameters iteratively
  - Gradient descent
  - Gauss–Newton method
  - Limited-memory BFGS
  - Derivative-free methods
  - etc.

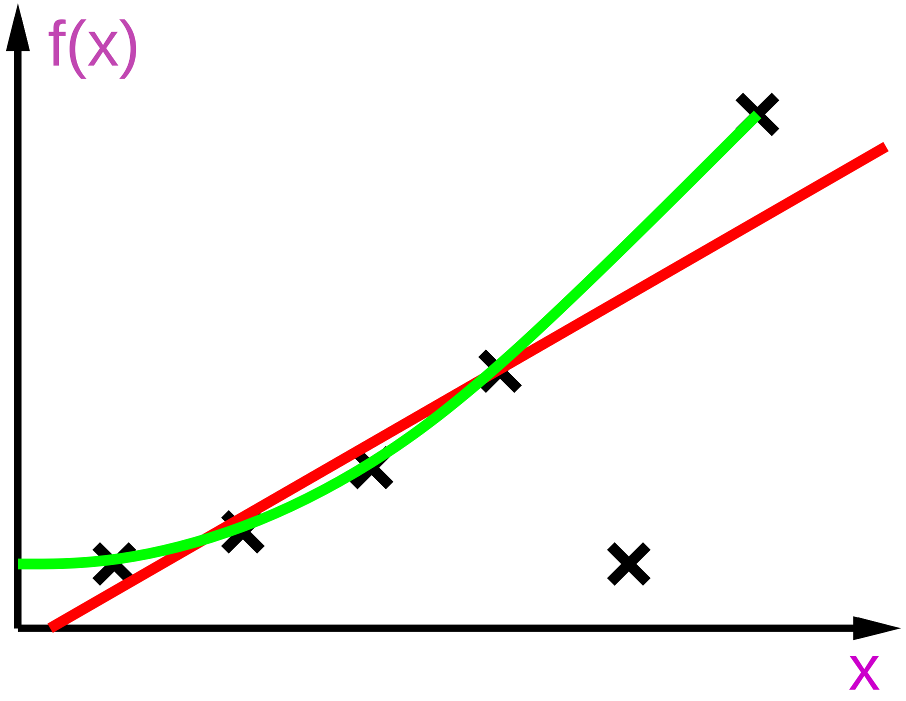
Overfitting in non-linear regression



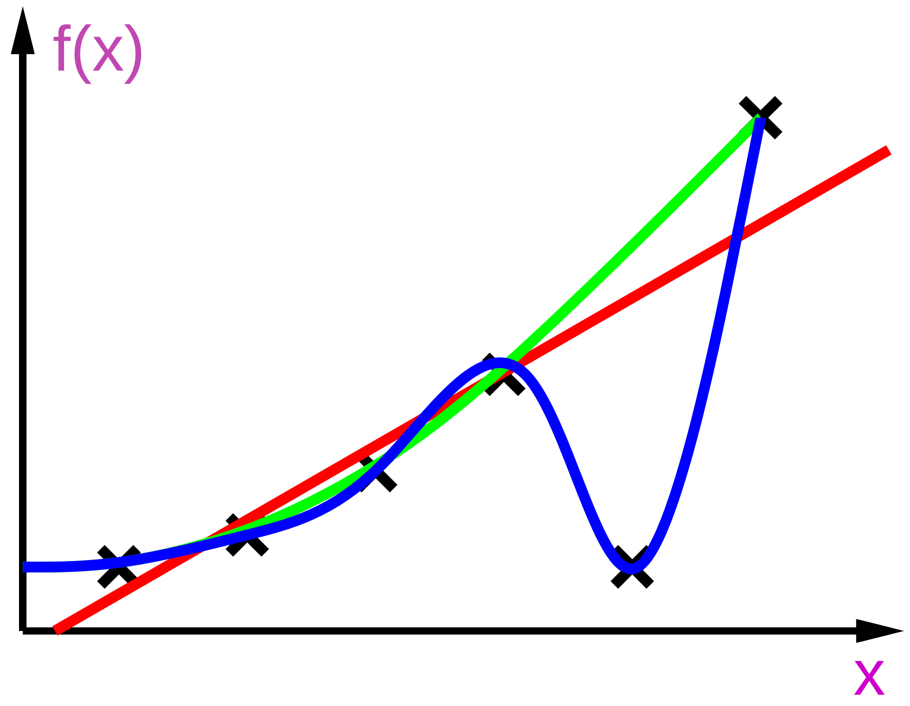
Overfitting in non-linear regression



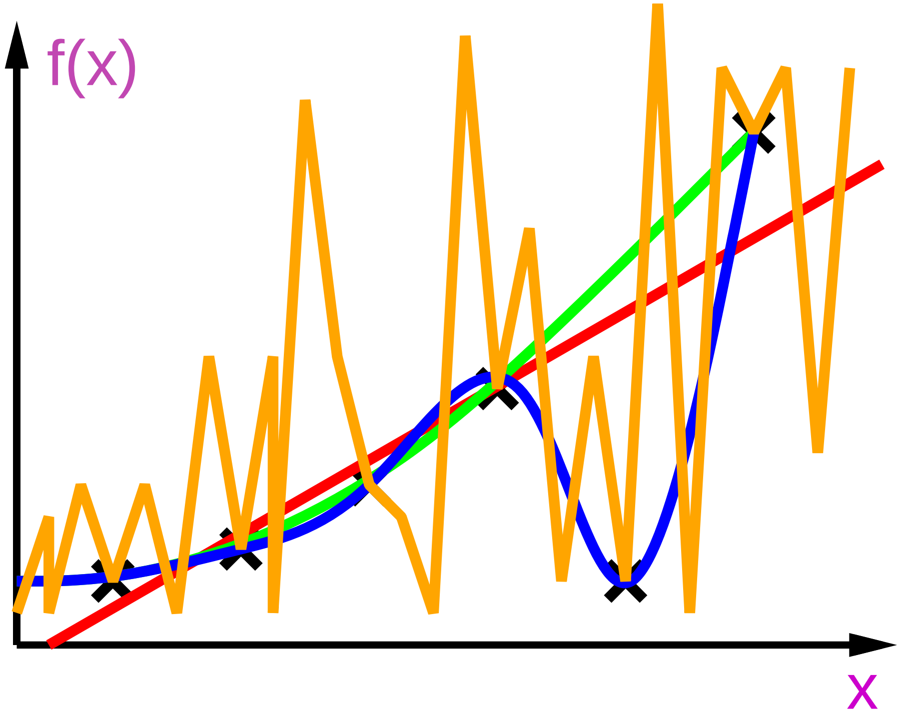
Overfitting in non-linear regression



Overfitting in non-linear regression



Overfitting in non-linear regression



Fit vs. complexity: a tradeoff

"*Ockham's razor*": prefer the *simplest* hypothesis consistent with the data

Summary

- Supervised learning:
  - Learning a function from labeled examples
- Classification: discrete-valued function
  - Naïve Bayes
  - Generalization and overfitting, smoothing
  - Perceptron
- Regression: real-valued function
  - Linear regression