Supervised Machine Learning

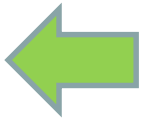
AIMA Chapter 18, 20

[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 for reinforcement learning
- Machine learning: how to acquire a model from data / experience
- Related courses
  - CS182 Introduction to Machine Learning
  - CS282 Machine Learning
  - CS280 Deep Learning

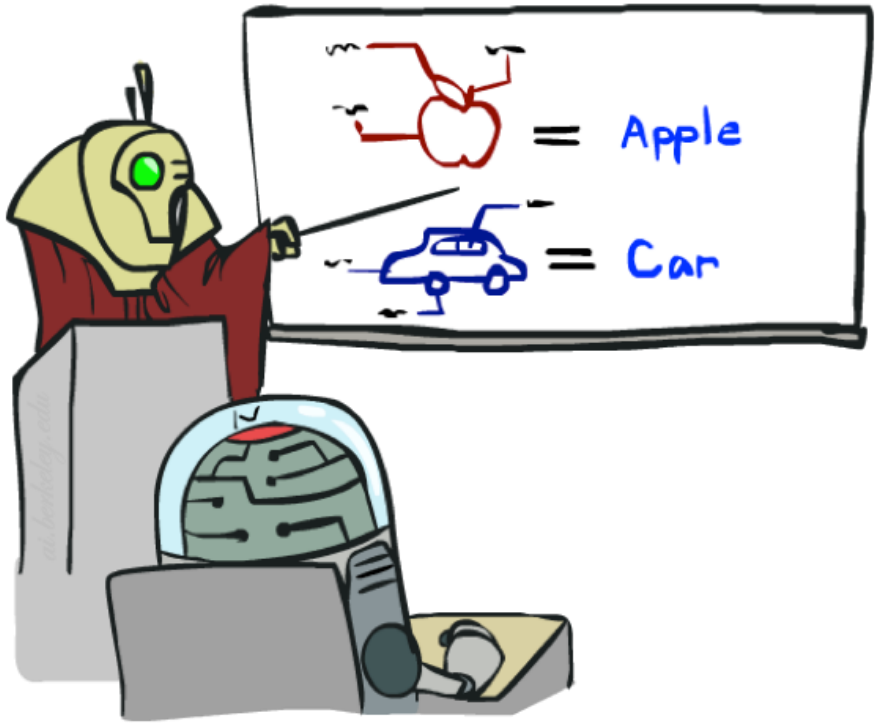
Types of Learning

- 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

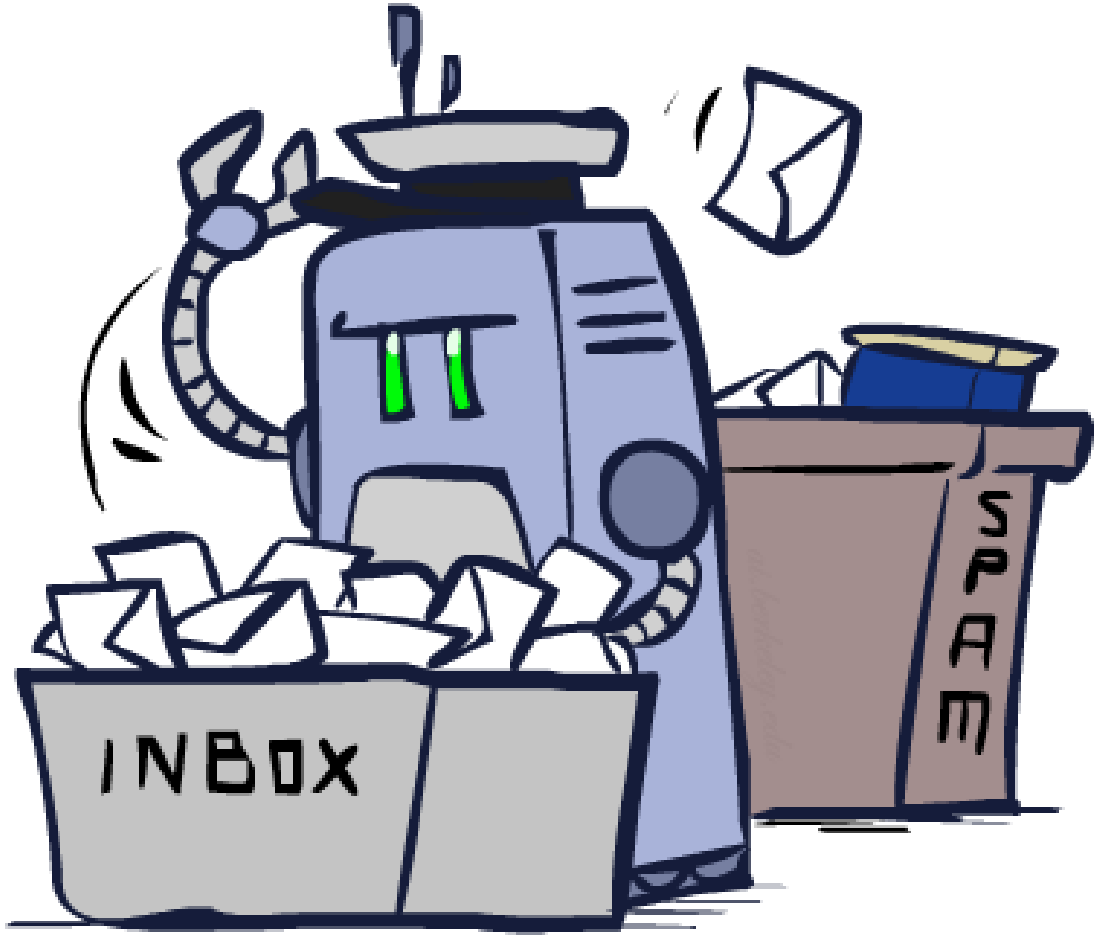
Supervised learning

- 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

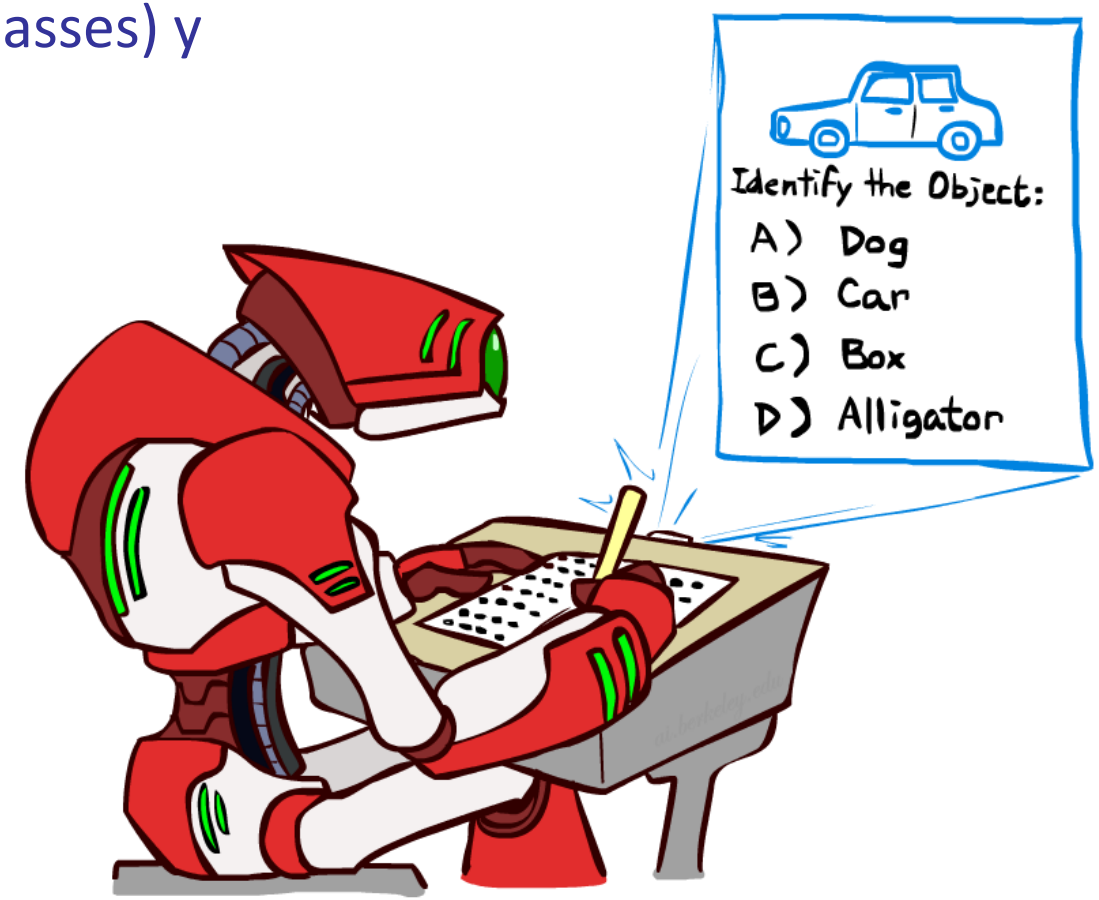
1

??

Other Classification Tasks

- Classification: given inputs x, predict labels (classes) 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!

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?

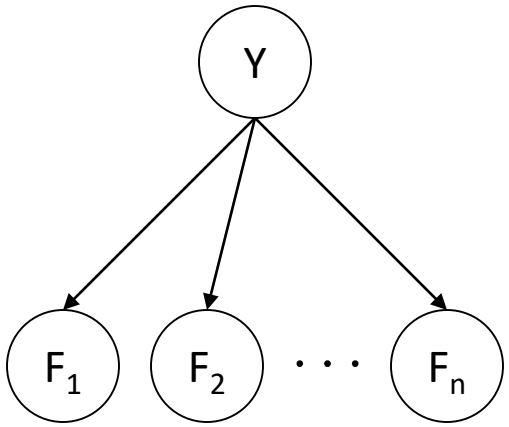
Naïve Bayes

- Naive Bayes model:

|Y| parameters

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

n x |F| x |Y| parameters



- Assume all features are independent effects of the label

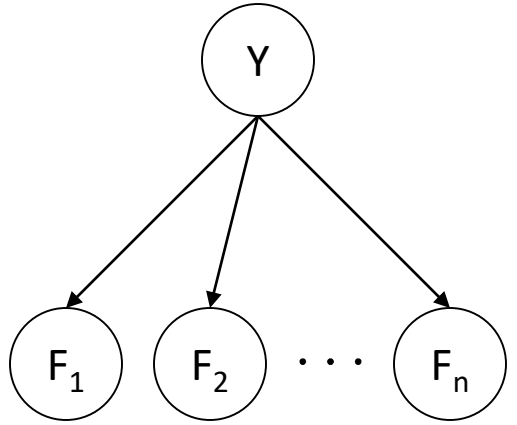
- Total number of parameters is *linear* in n
- Model is very simplistic, but often works anyway

Naïve Bayes for Digits

- 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.

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

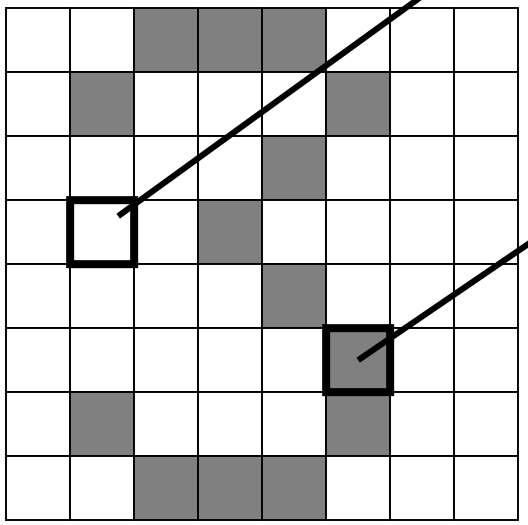
  - Here: lots of features, each is binary valued



Naïve Bayes for Digits

$P(Y)$

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

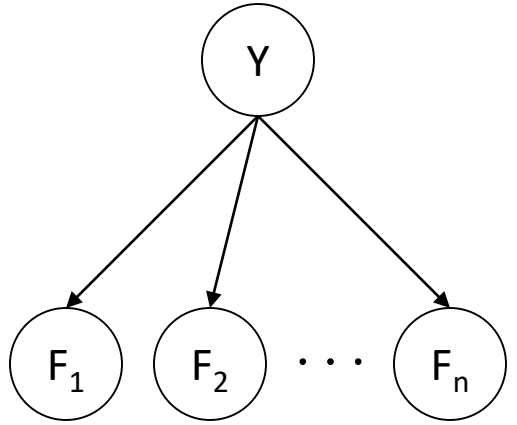


$P(F_{3,1} = on|Y)$

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)
  - This is 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
...
```

Inference for Naïve Bayes

- Goal: compute posterior distribution over label variable Y: $P(Y|f_1 \dots f_n)$
  - Step 1: get joint probability of label and evidence for each label

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

$$\frac{\begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix}}{P(f_1 \dots f_n)} \quad +$$

  - Step 2: normalization

$$\Downarrow$$

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

Spam Example

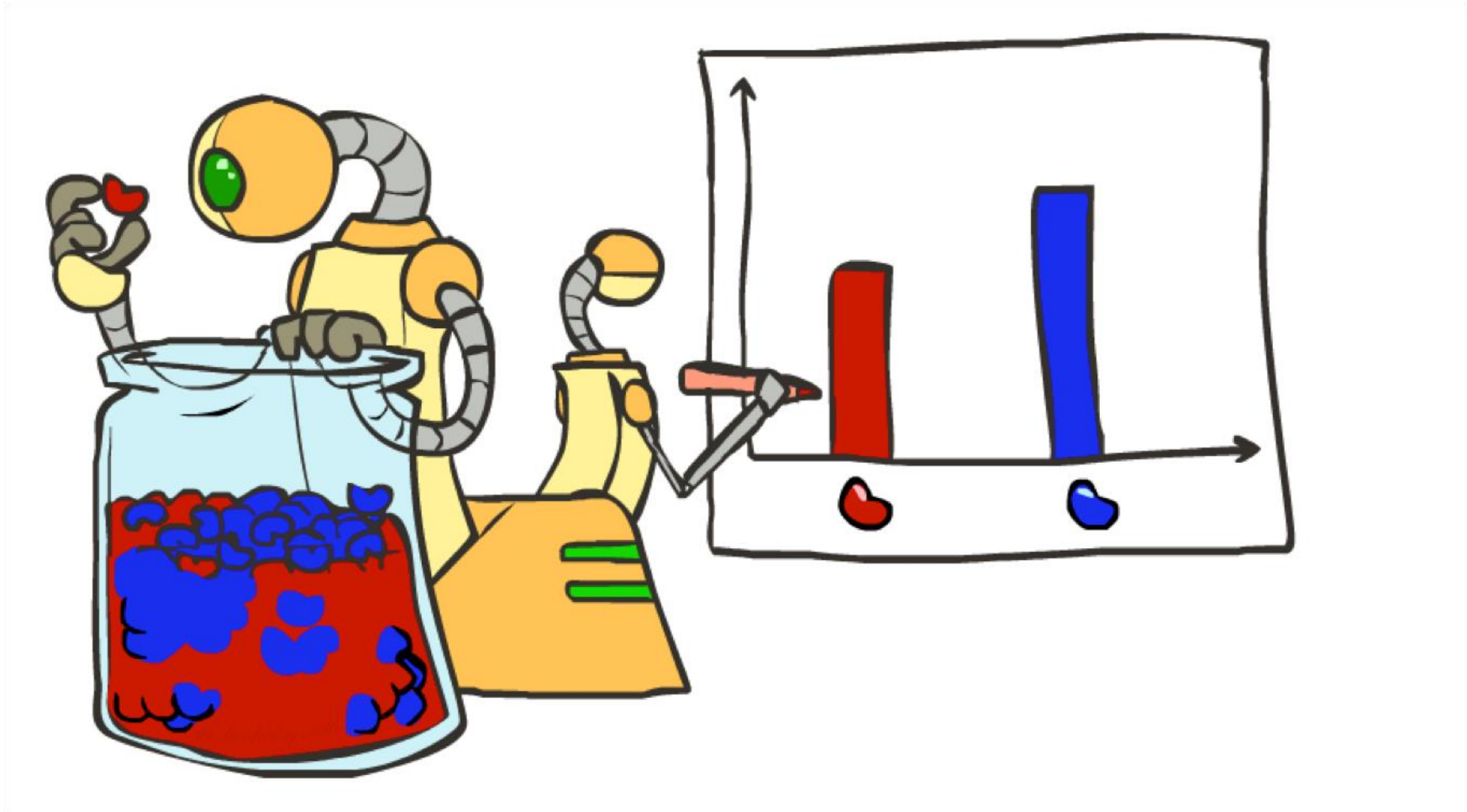
$$P(Y|w_1, w_2, \ldots, w_n) \propto P(Y) \prod_i P(w_i|Y)$$

Log product of probabilities

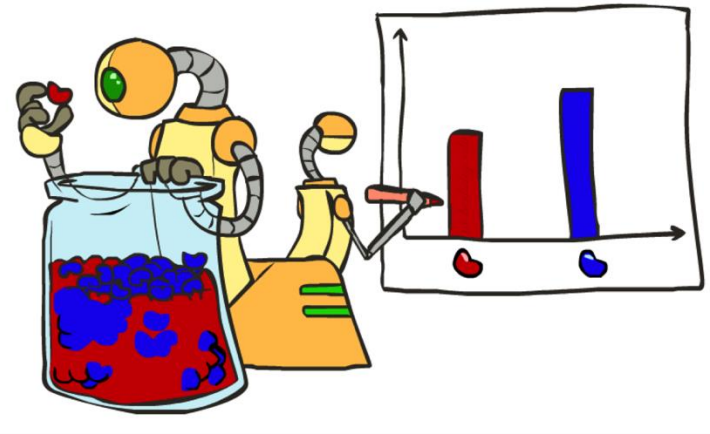
Word	P(w|spam)	P(w|ham)	Tot Spam	Tot Ham
(prior)	0.33333	0.66666	-1.1	-0.4
Gary	0.00002	0.00021	-11.8	-8.9
would	0.00069	0.00084	-19.1	-16.0
you	0.00881	0.00304	-23.8	-21.8
like	0.00086	0.00083	-30.9	-28.9
to	0.01517	0.01339	-35.1	-33.2
lose	0.00008	0.00002	-44.5	-44.0
weight	0.00016	0.00002	-53.3	-55.0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

Learning (Training)



Parameter Estimation



- For Naïve Bayes, we know the model structure; we need to estimate the CPTs (conditional distributions)
- *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*

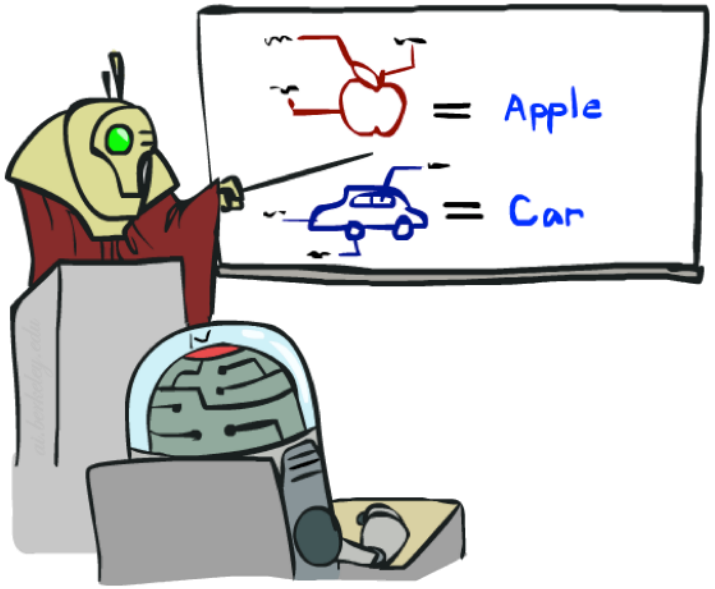
$$\begin{aligned}\widehat{\theta} &= \arg\max_{\theta} \ln P(\mathcal{D} \mid \theta) \\ &= \arg\max_{\theta} \ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}\end{aligned}$$

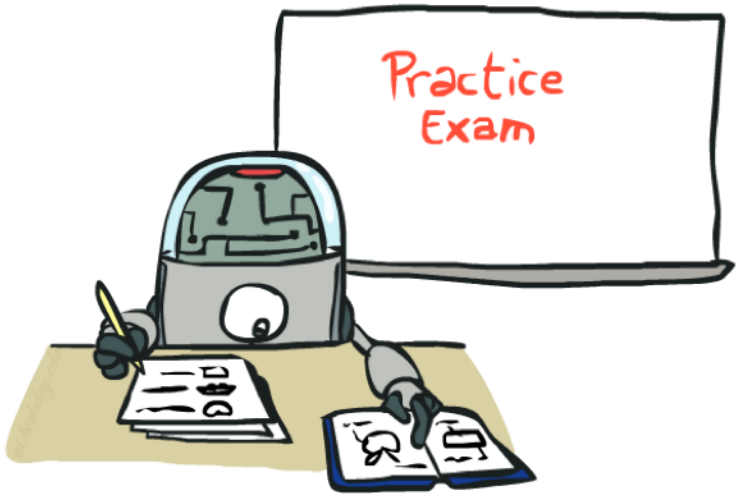
- Set derivative to zero, and solve!

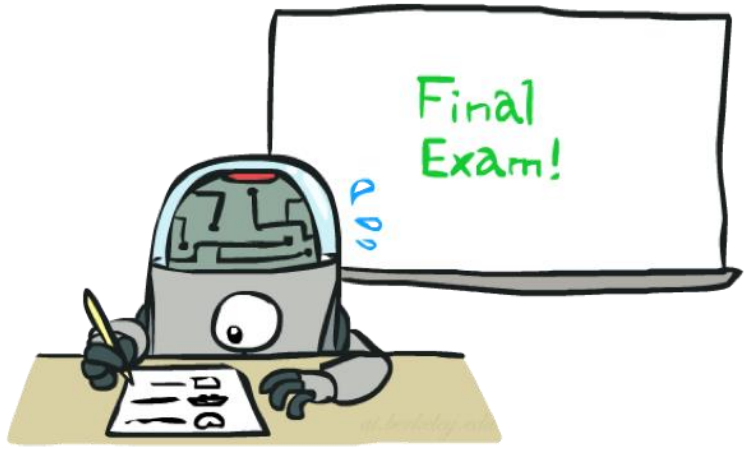
$$\begin{aligned}\frac{d}{d\theta} \ln P(\mathcal{D} \mid \theta) &= \frac{d}{d\theta}\left[\ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}\right] \\ &= \frac{d}{d\theta}\left[\alpha_H \ln\theta + \alpha_T \ln(1-\theta)\right] \\ &= \alpha_H \frac{d}{d\theta}\ln\theta + \alpha_T \frac{d}{d\theta}\ln(1-\theta) \\ &= \frac{\alpha_H}{\theta} - \frac{\alpha_T}{1-\theta} = 0\end{aligned}$$

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

Training and Testing

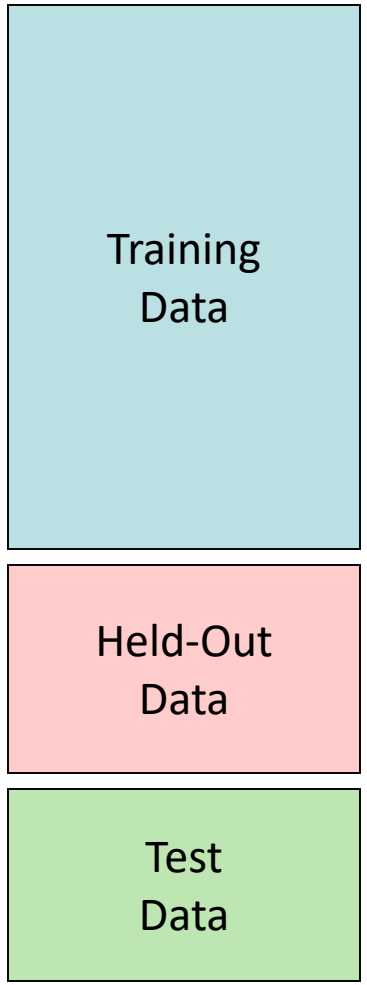






Important Concepts

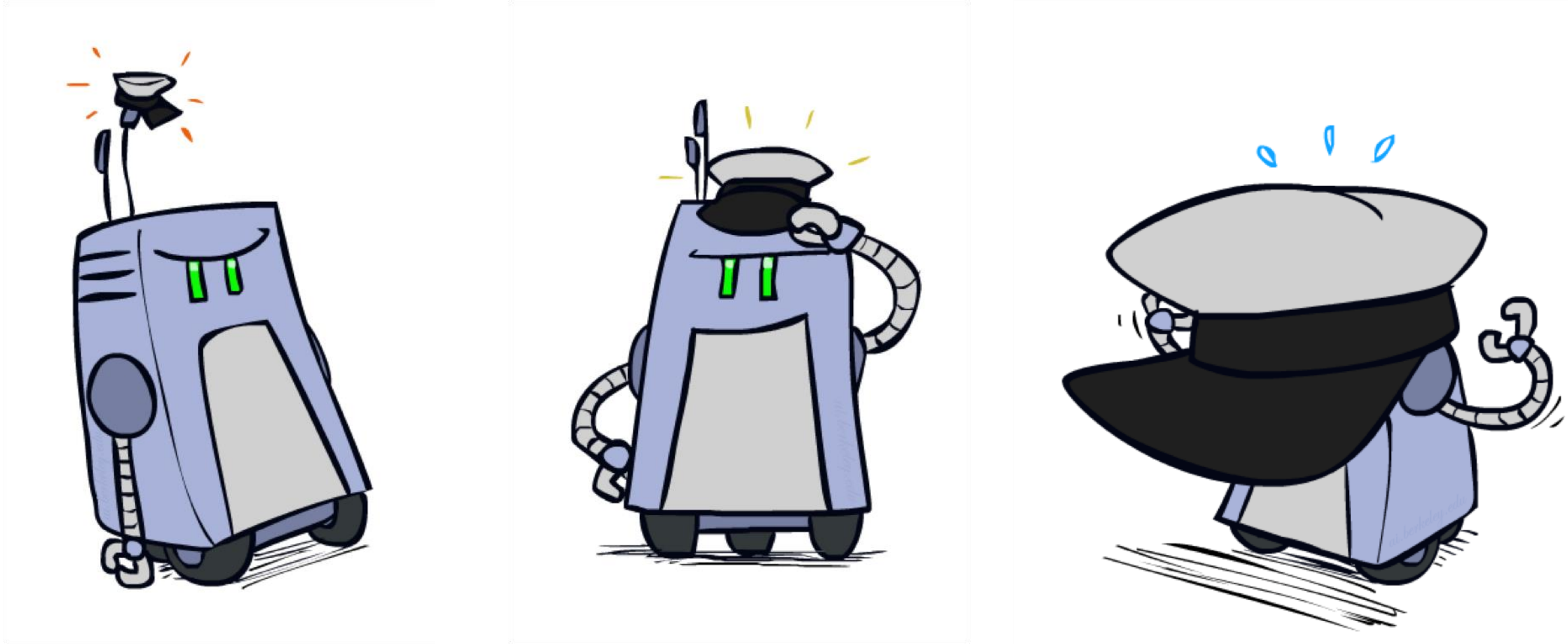
- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - Tune hyperparameters on held-out set
  - Compute accuracy of test set (fraction of instances predicted correctly)
  - Very important: never "peek" at the test set!
- Typical problems
  - Underfitting: fitting the training set poorly
  - Overfitting: fitting the training data very well, but not the test data



Baselines

- Is your testing accuracy good or bad?
- 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. if calling everything “ham” gets 66%, then a classifier that gets 70% isn’t very good…
- For real research, usually use previous work as a (strong) baseline

Generalization and Overfitting



Example: Overfitting

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

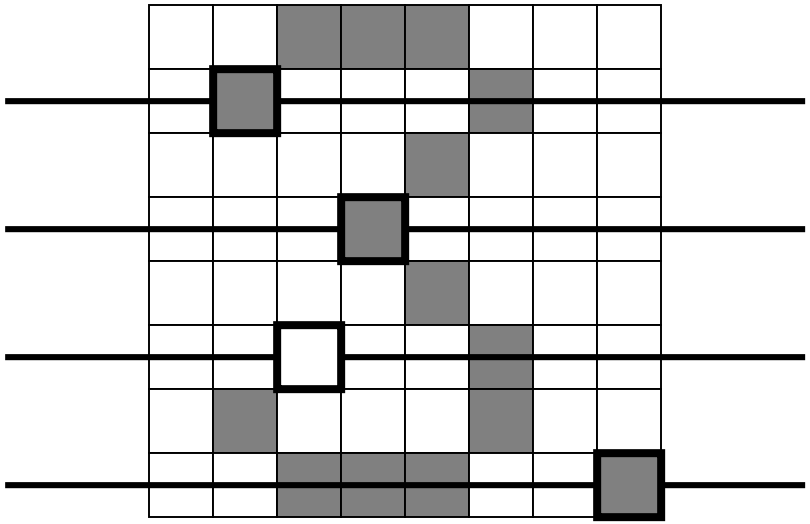
$P(C = 2) = 0.1$

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

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

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

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



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

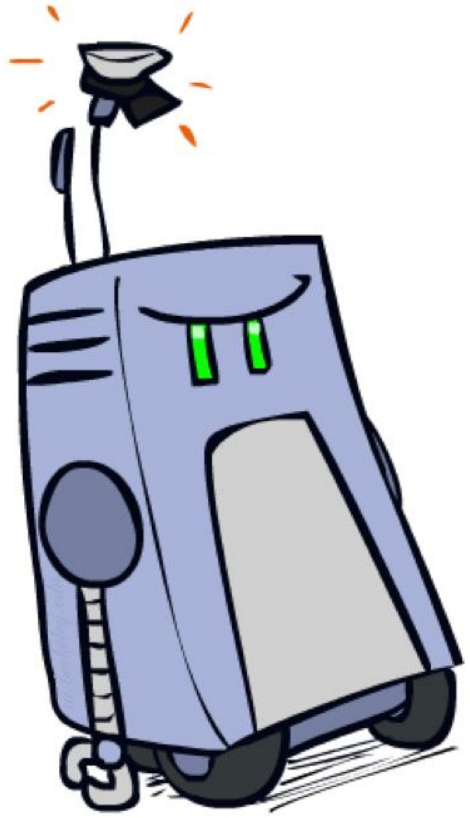
$P(C = 3) = 0.1$

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

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

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

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



2 wins!!

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

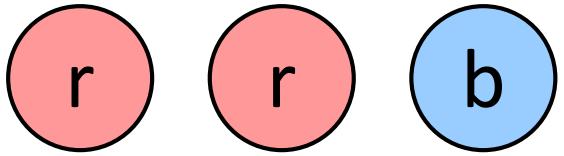
Generalization and Overfitting

- 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



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

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

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

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

Laplace Smoothing

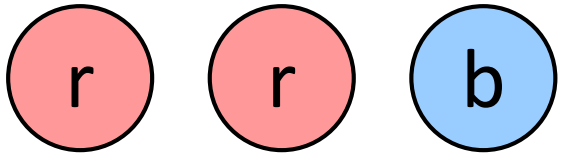
- Laplace's estimate (extended):
  - Pretend you saw every outcome k extra times

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

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

- Laplace for conditionals:
  - Smooth each condition independently:

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



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

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

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

Maximum Likelihood?

- Empirical rates are the maximum likelihood estimates

$$\theta_{ML} = \arg\max_{\theta} P(\mathbf{X}|\theta)$$
$$= \arg\max_{\theta} \prod_i P_\theta(X_i)$$

$$\Rightarrow \quad P_{\mathsf{ML}}(x) = \frac{\mathsf{count}(x)}{\mathsf{total\ samples}}$$

- Laplace's estimate is the most likely parameter value given the data

$$\theta_{MAP} = \arg\max_{\theta} P(\theta|\mathbf{X})$$
$$= \arg\max_{\theta} P(\mathbf{X}|\theta)P(\theta)/P(\mathbf{X})$$
$$= \arg\max_{\theta} P(\mathbf{X}|\theta)P(\theta)$$

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

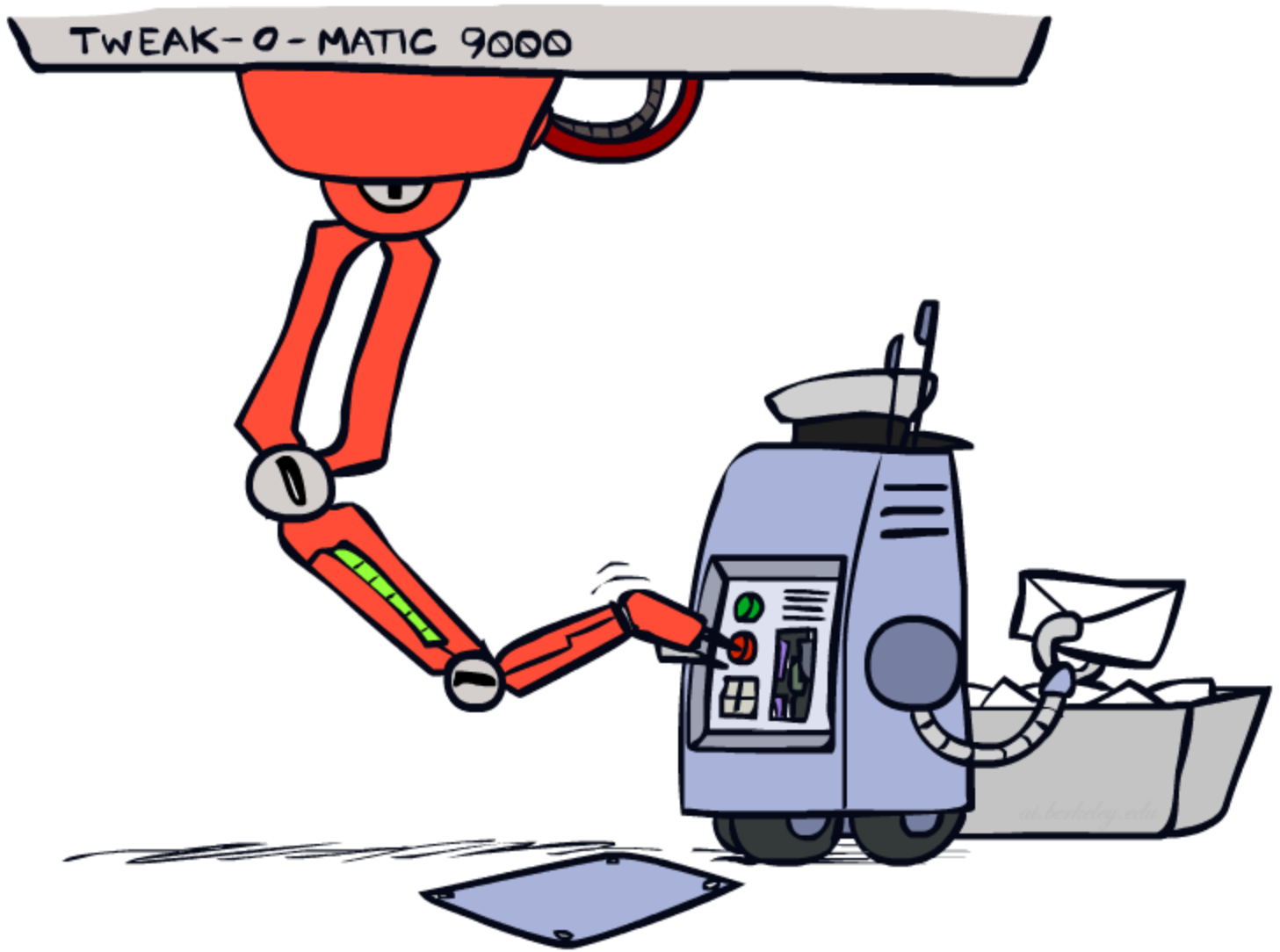
Dirichlet distribution (parameterized by k)

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)

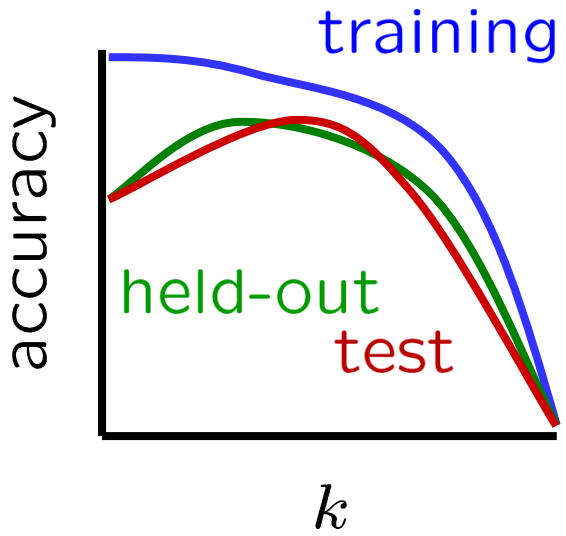
$$P_{LIN}(x|y) = \alpha \hat{P}(x|y) + (1.0 - \alpha)\hat{P}(x)$$

Tuning



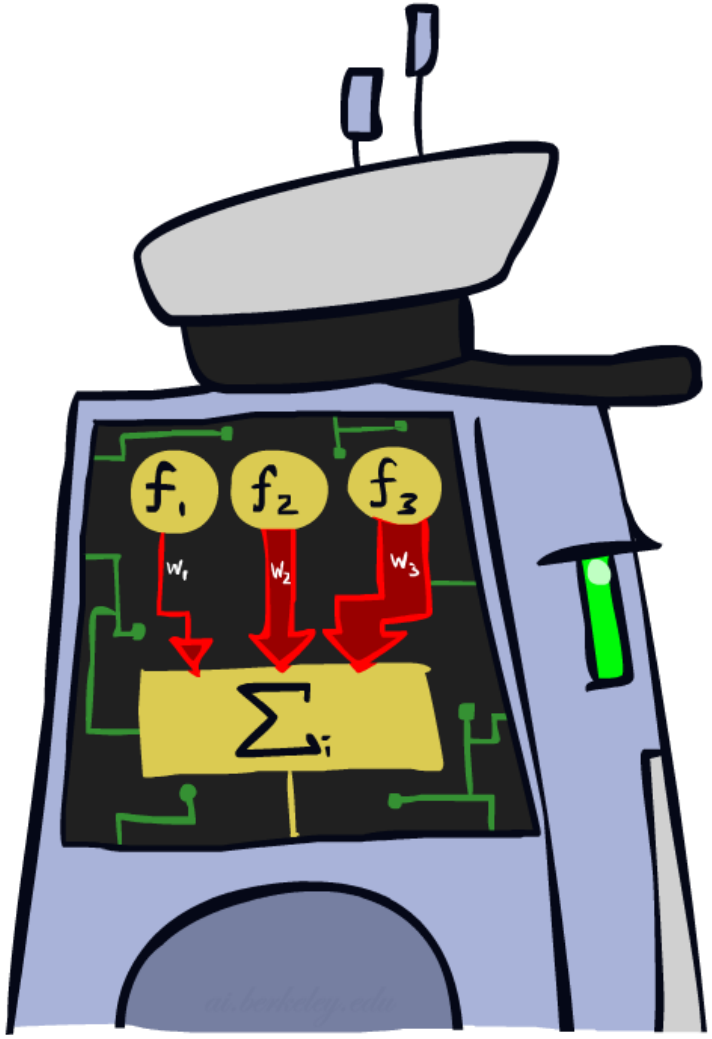
Tuning on Held-Out Data

- 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



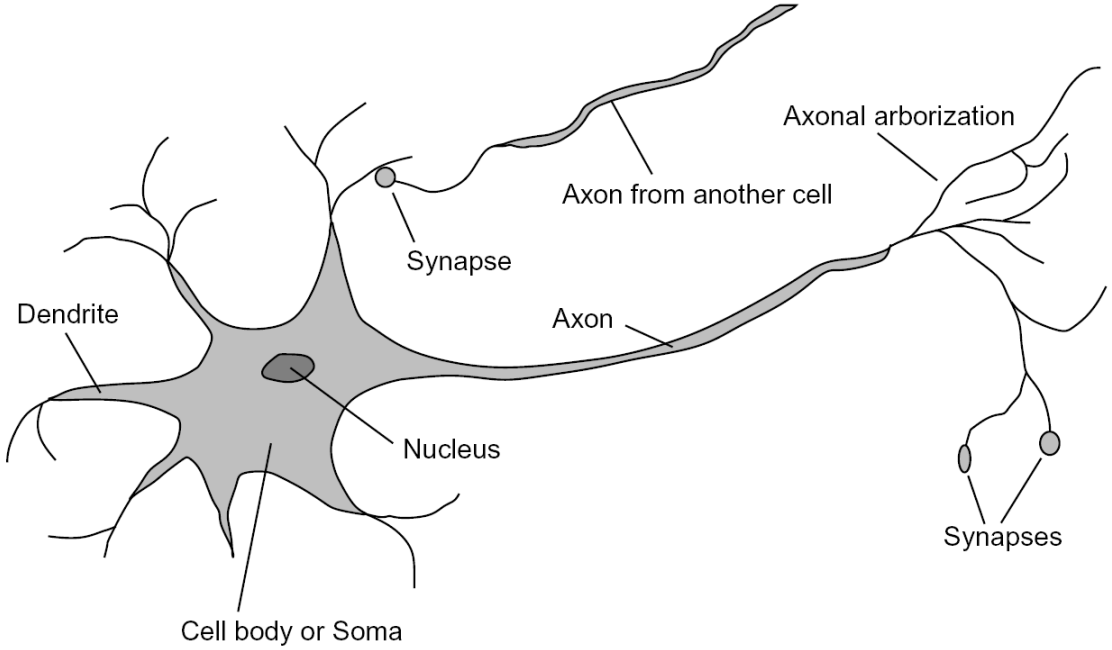
Held-out data is also known as validation data or development data

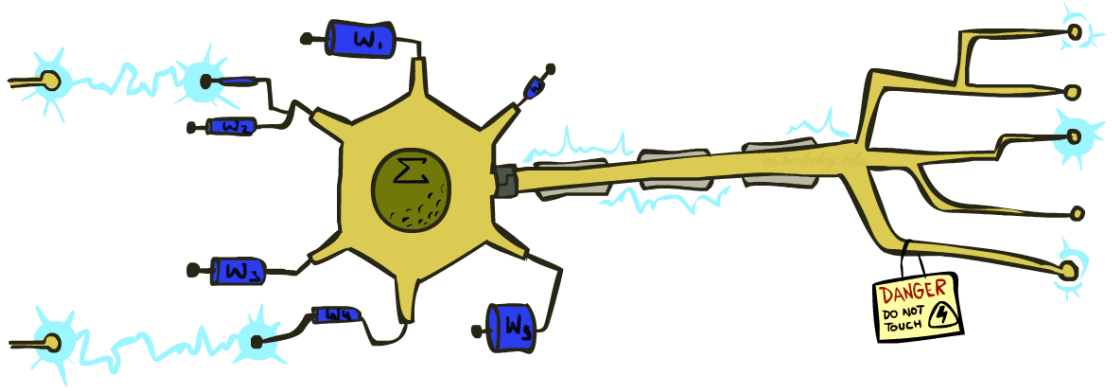
Linear Classifiers (Perceptrons)



Some (Simplified) Biology

- Very loose inspiration: human neurons



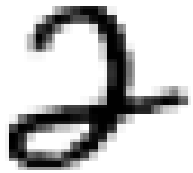


Feature Vectors

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
...
```

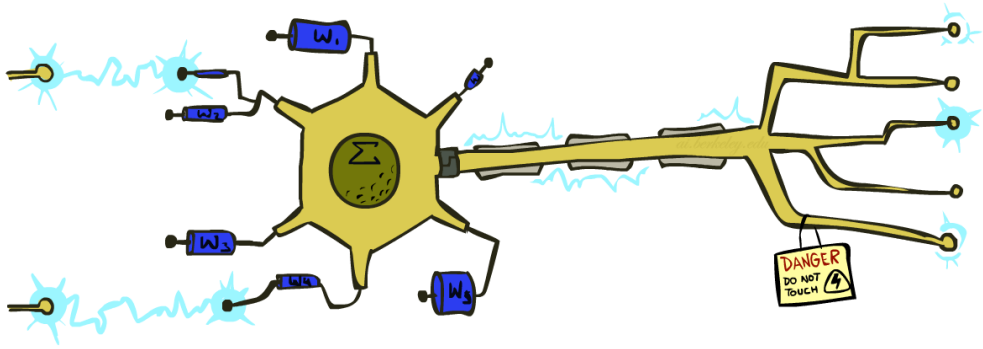
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

Spam
or
Ham

“2”

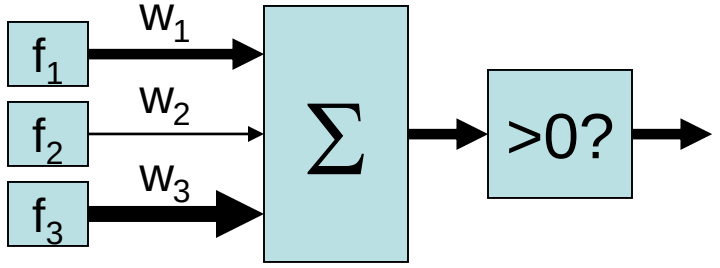
Linear Classifiers

- 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



Linear Classifiers

- Sometimes we also add a bias term

$$\sum_i w_i \cdot f_i(x) + b = w \cdot f(x) + b$$

- This is equivalent to appending a constant feature

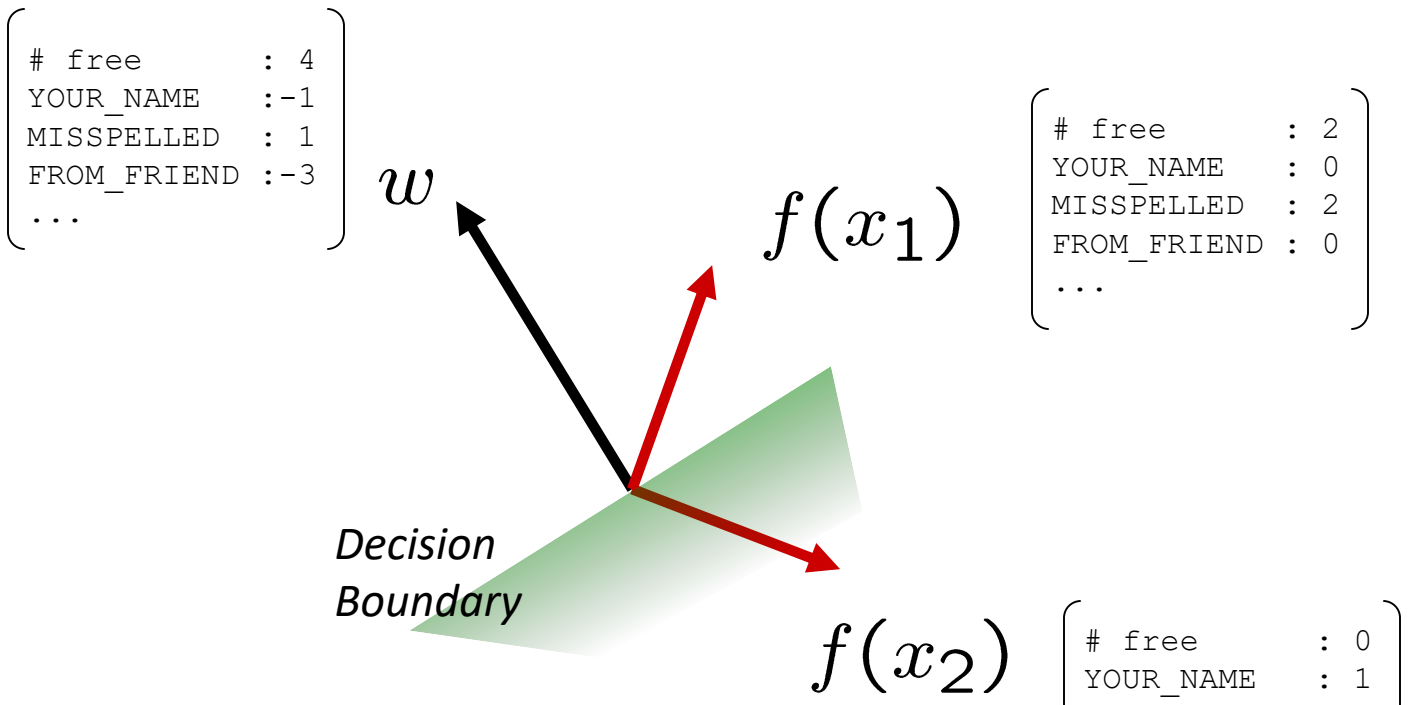
$f(x)$

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
Bias         ≡ 1
```

$$\sum_{i=1}^{n+1} w_i \cdot f_i(x) = \sum_{i=1}^{n} w_i \cdot f_i(x) + w_{n+1}$$

Decision Boundary

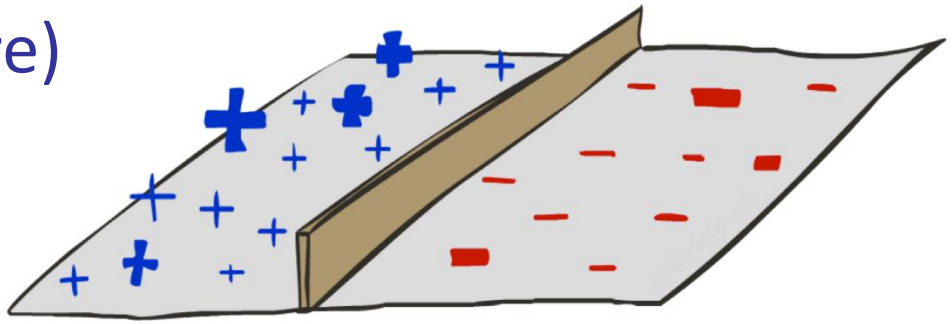
- Binary case: compare features to a weight vector

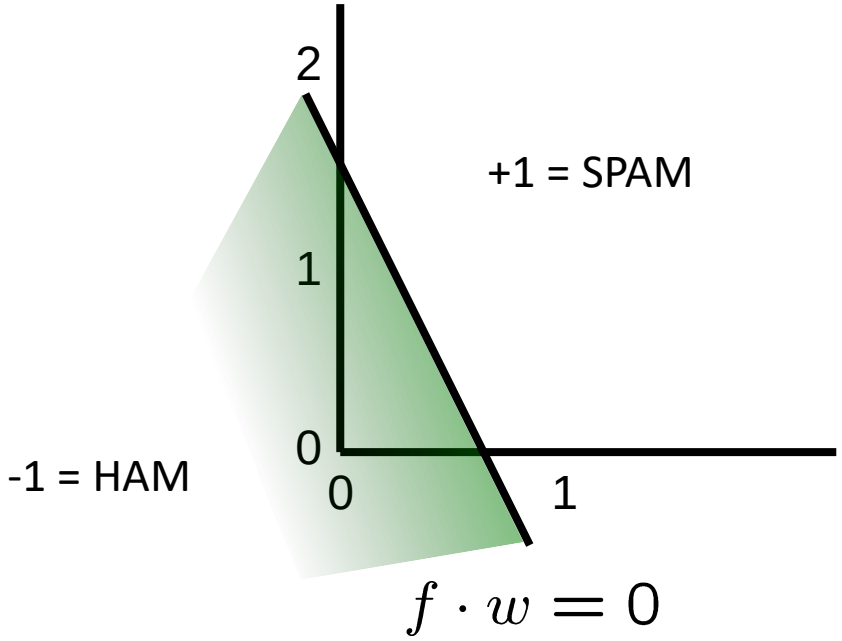


Dot product $w \cdot f$ positive means the positive class

Decision Boundary

- In the space of feature vectors (no bias feature)
  - Examples are points
  - Weight vector specifies a hyperplane
    - $w \cdot f(x) + b = 0$
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1

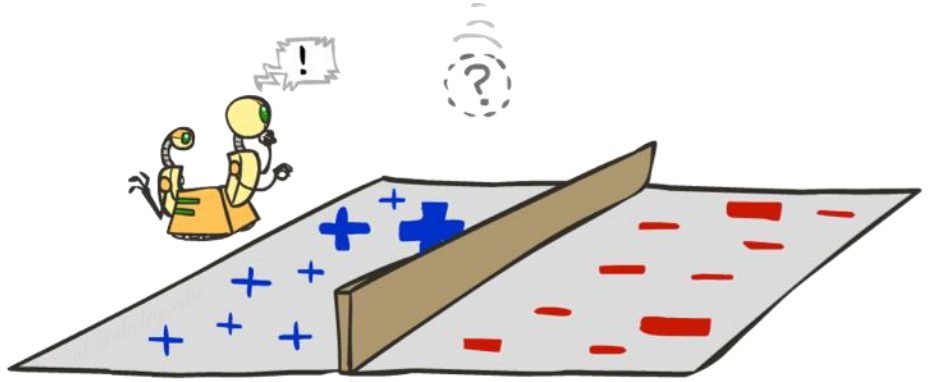




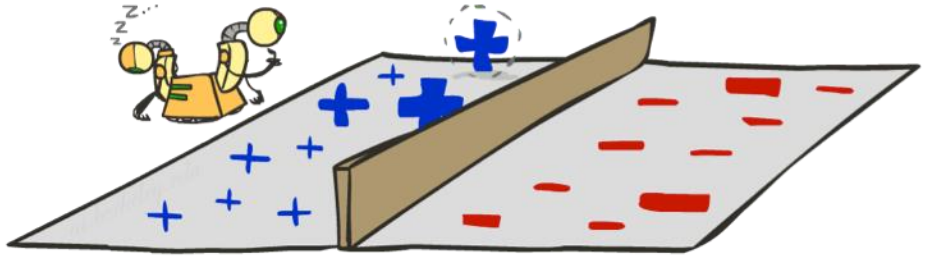
Weight Updates

Learning: Binary Perceptron

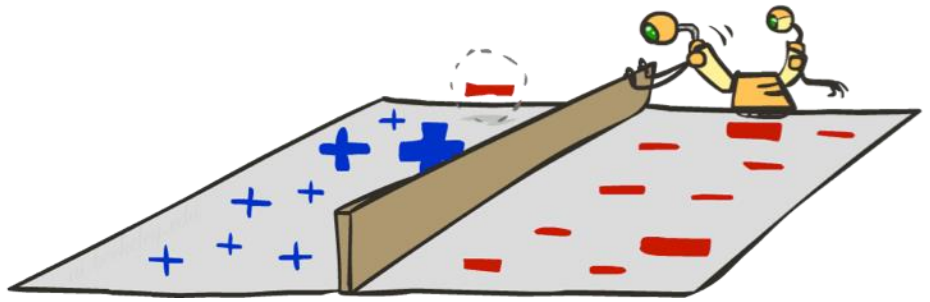
- Start with weights = 0
- For each training instance:
  - Classify with current weights



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



  - If wrong: adjust the weight vector



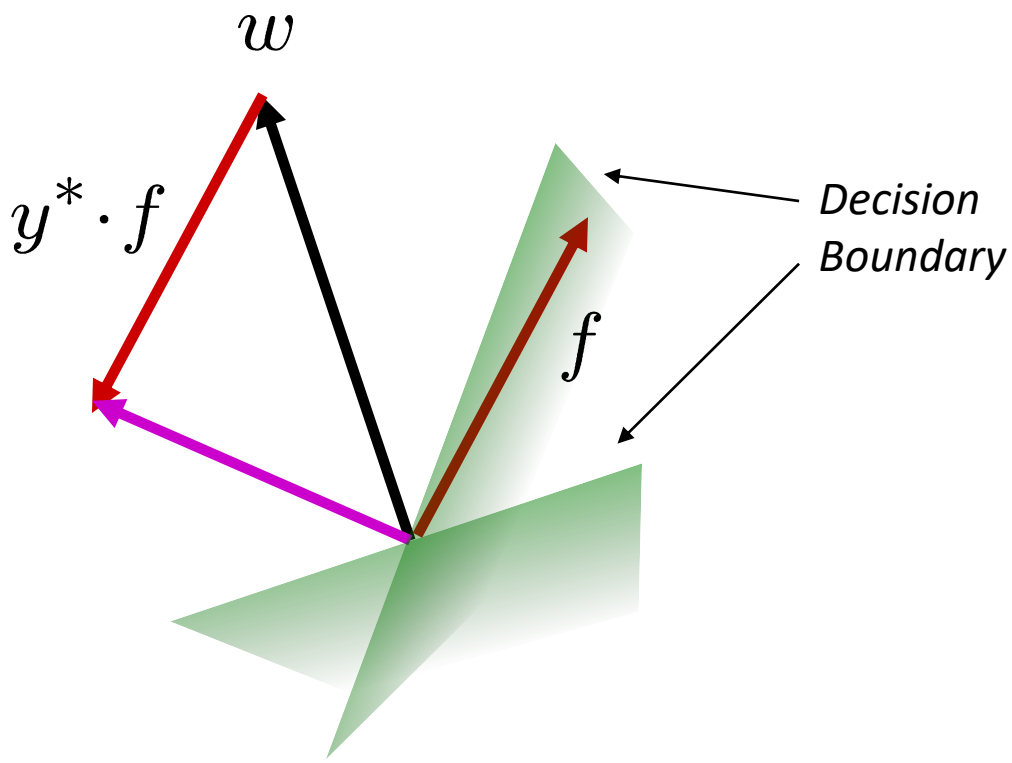
Learning: Binary Perceptron

- 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.

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

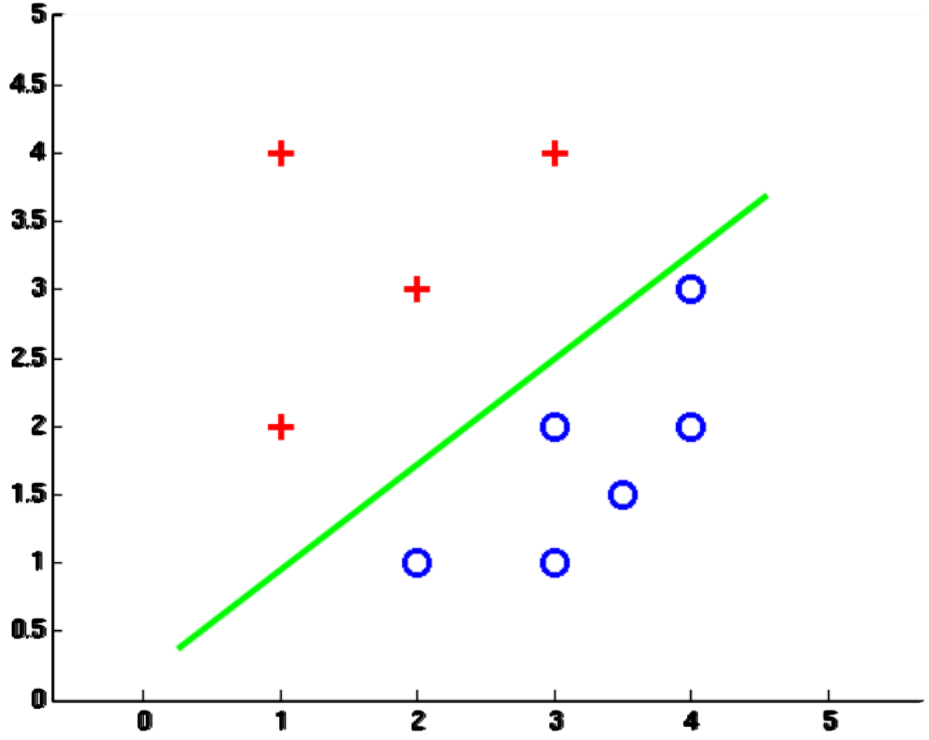


Before: $w \cdot f(x)$

After: $w \cdot f(x) + y^* f(x) \cdot f(x)$ ≥ 0

Examples: Perceptron

- Separable Case



Multiclass Decision Rule

- If we have multiple classes:
  - A weight vector for each class:

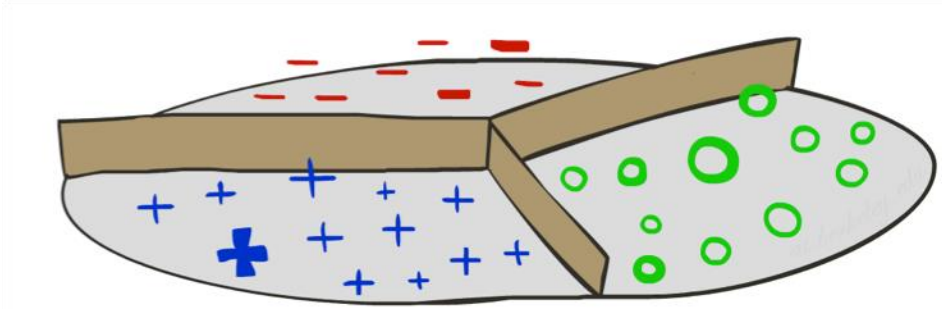
$$w_y$$

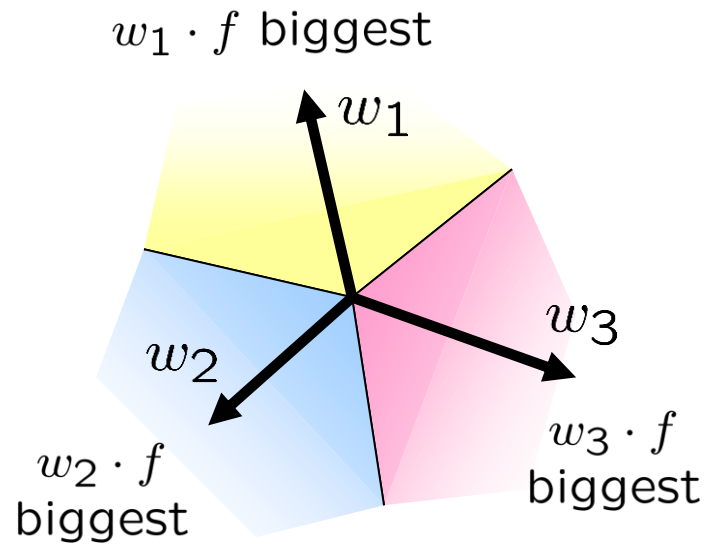
  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - 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

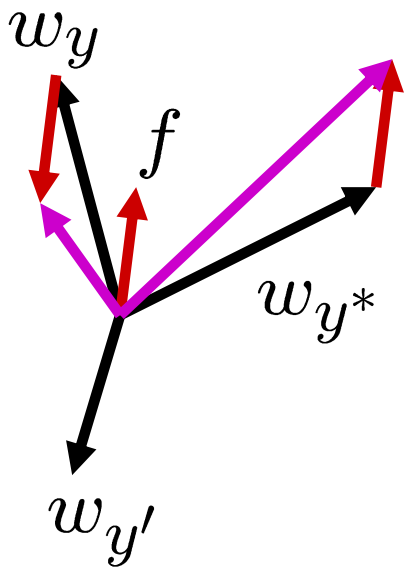
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

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

- 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" [1 1 0 1 1]

"win the election" [1 1 0 0 1]

"win the game" [1 1 1 0 1]

w_{SPORTS}

	1	-2	-2
BIAS : 1	0		1
win : 0	-1		0
game : 0	0		1
vote : 0	-1		-1
the : 0	-1		0
...			

$w_{POLITICS}$

	0	3	3
BIAS : 0	1		0
win : 0	1		0
game : 0	0		-1
vote : 0	1		1
the : 0	1		0
...			

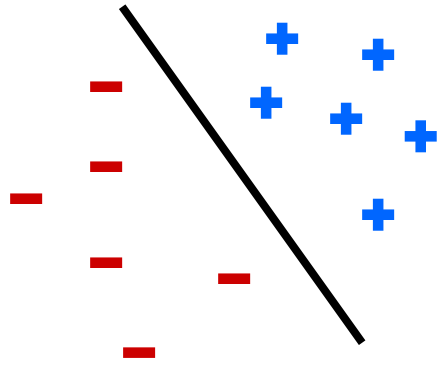
w_{TECH}

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

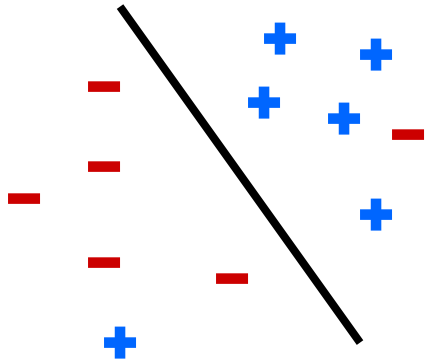
Properties of Perceptrons

- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training set is separable, perceptron will eventually converge (binary case)

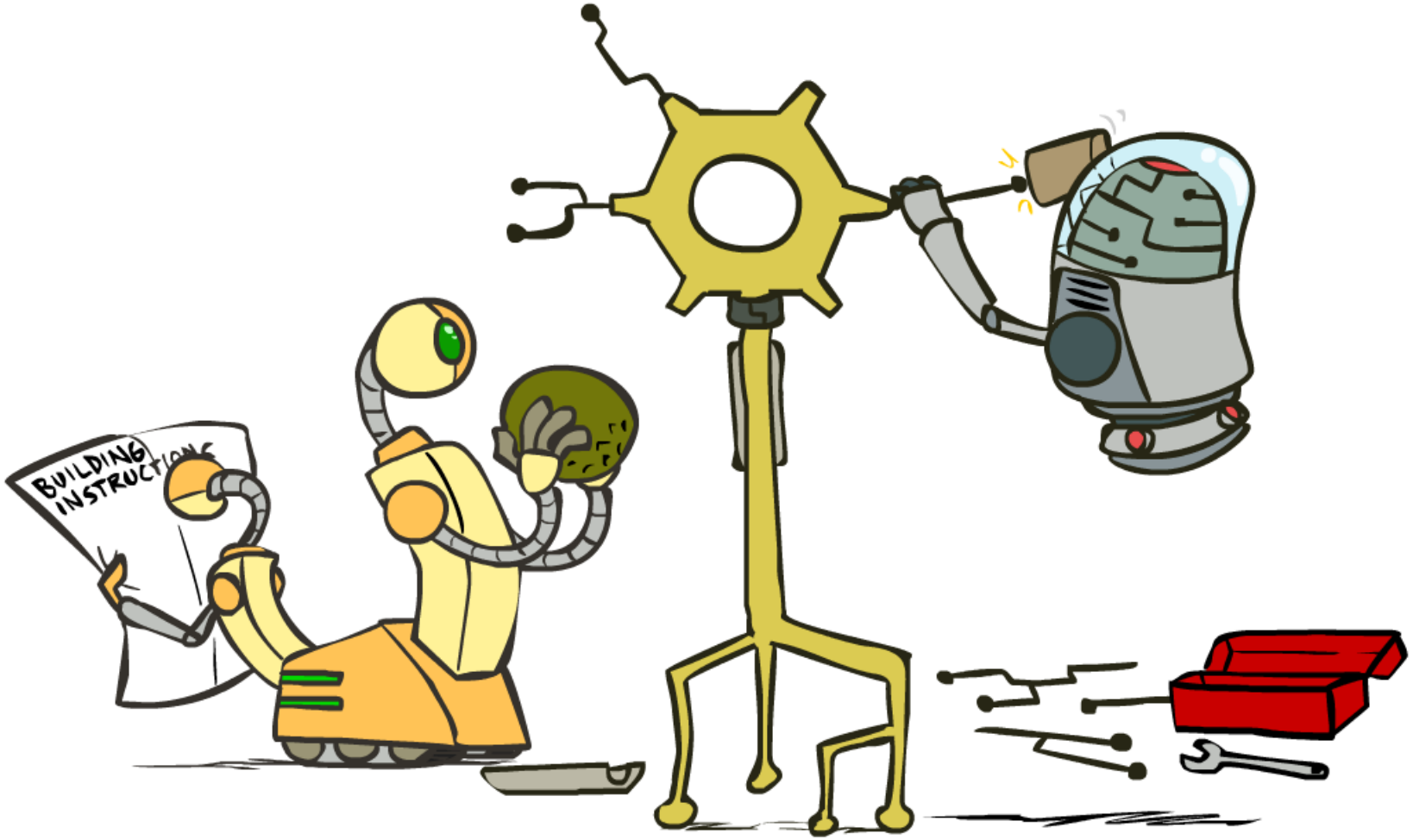
Separable



Non-Separable

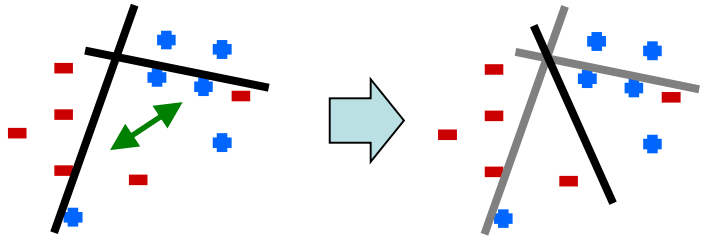


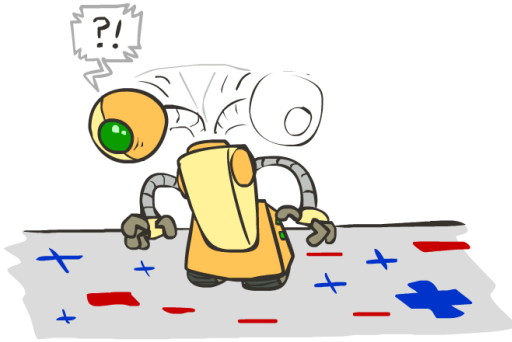
Improving Perceptrons (Logistic Regression)



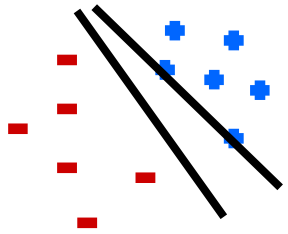
Problems with Perceptrons

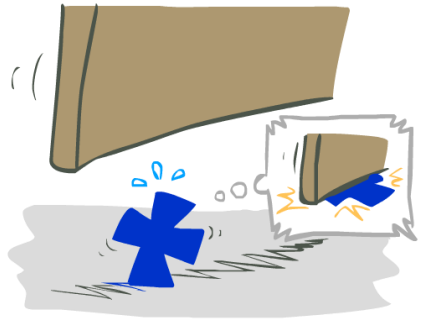
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



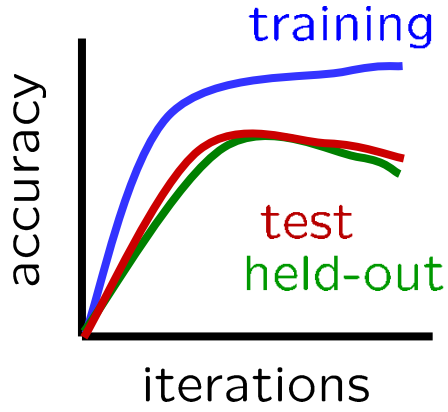


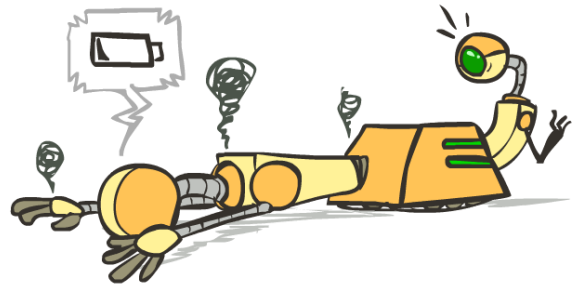
- Mediocre generalization: finds a "barely" separating solution





- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting

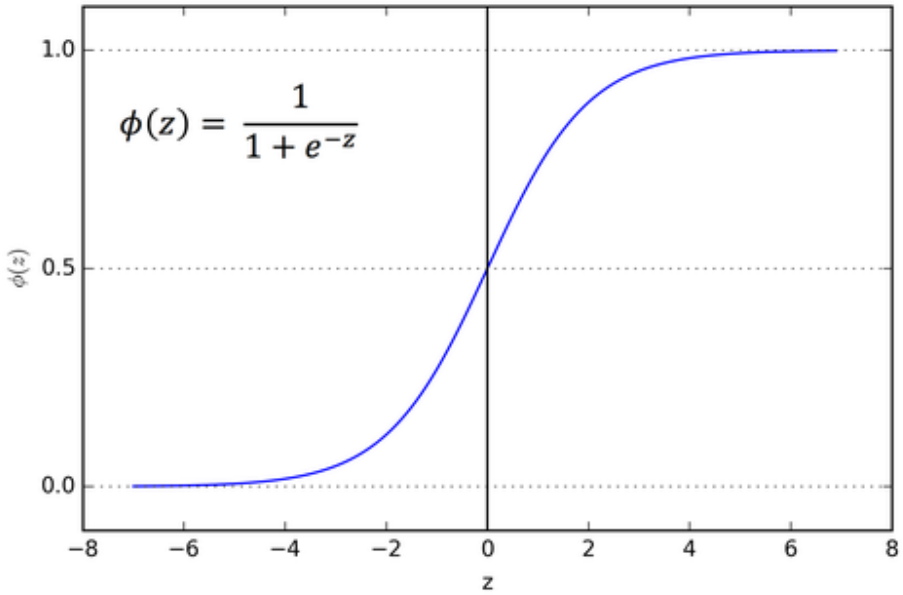




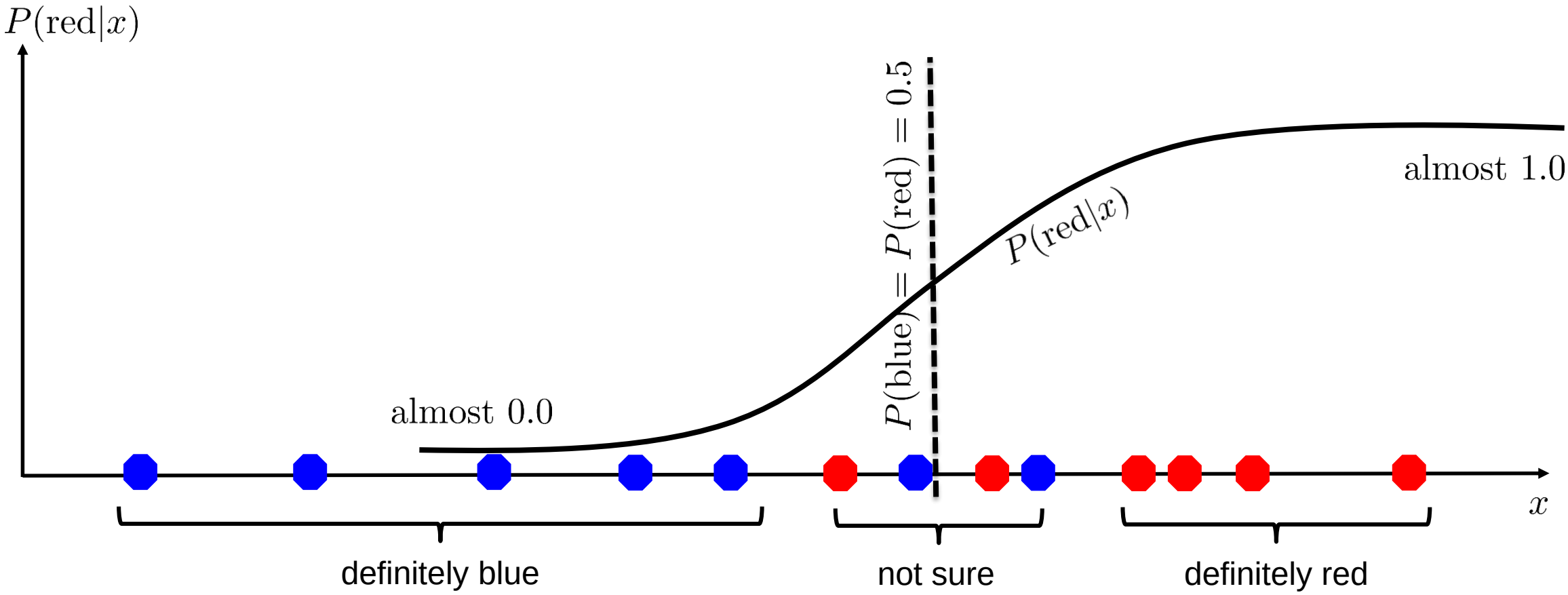
Probabilistic decisions

- Perceptron scoring: $z = w \cdot f(x)$
  - z=0.1 and z=100 both produces +1
- Probabilistic decisions
  - z very positive → prob of +1 going to 1
  - z close to 0 → prob of +1 close to 0.5
  - z very negative → prob of +1 going to 0
- Sigmoid function

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



A 1D Example


$P(\text{red}|x)$
$P(\text{blue}) = P(\text{red}) = 0.5$
$P(\text{red}|x)$
almost 1.0
almost 0.0
x
definitely blue
not sure
definitely red

Best w?

- Maximum likelihood estimation:

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

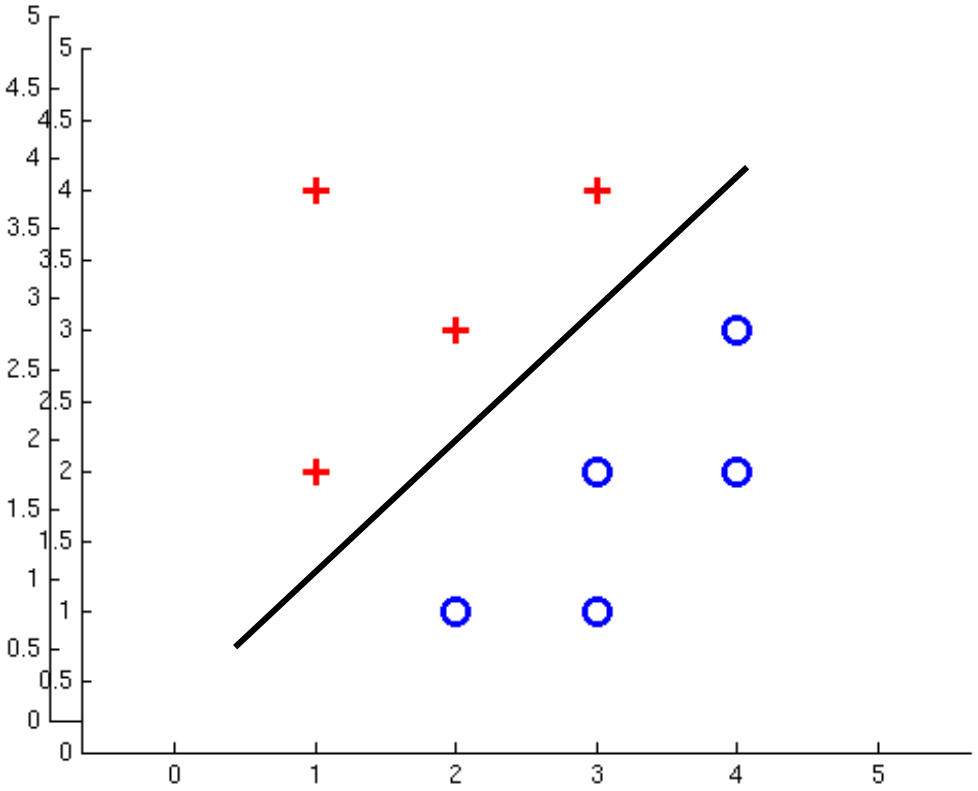
with:

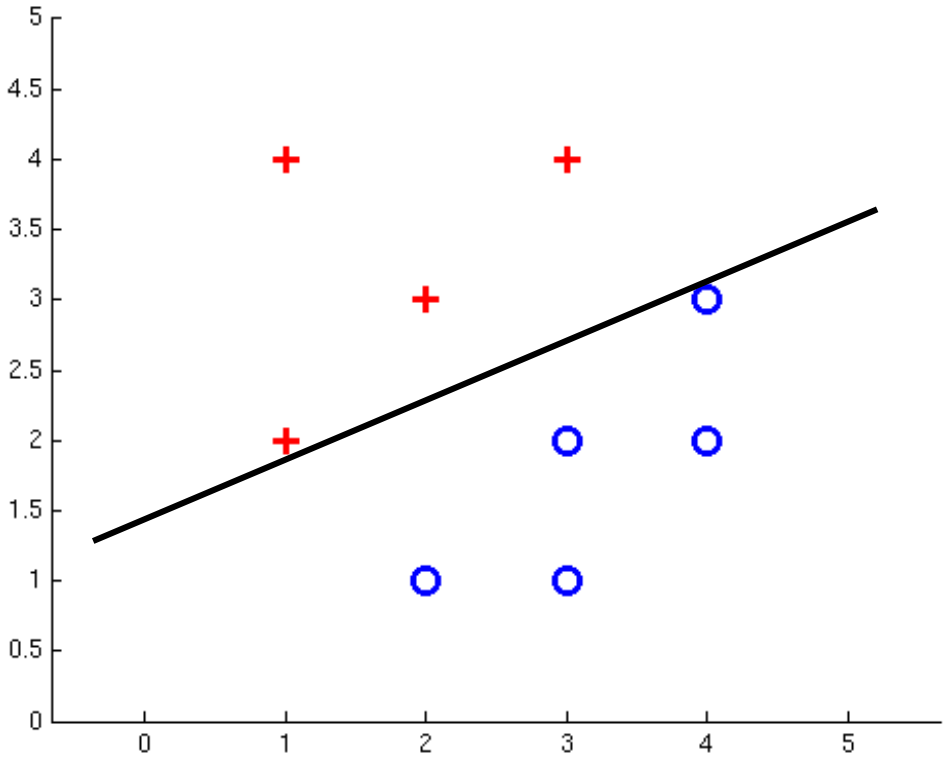
$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

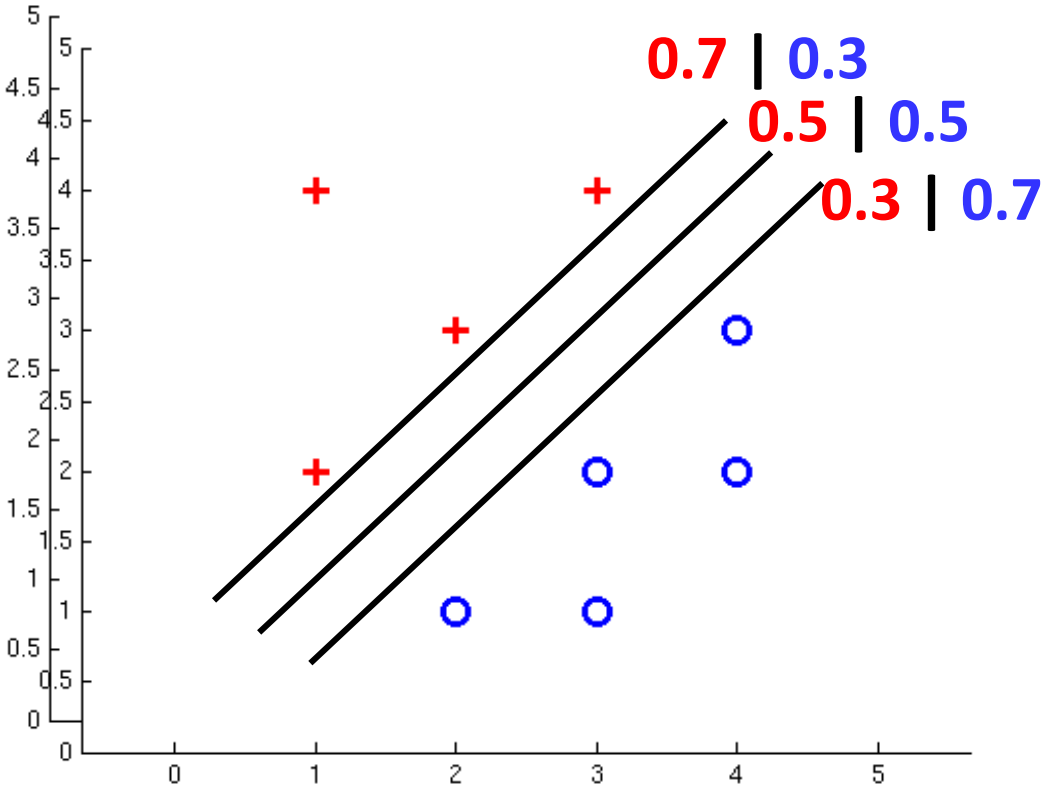
= Logistic Regression

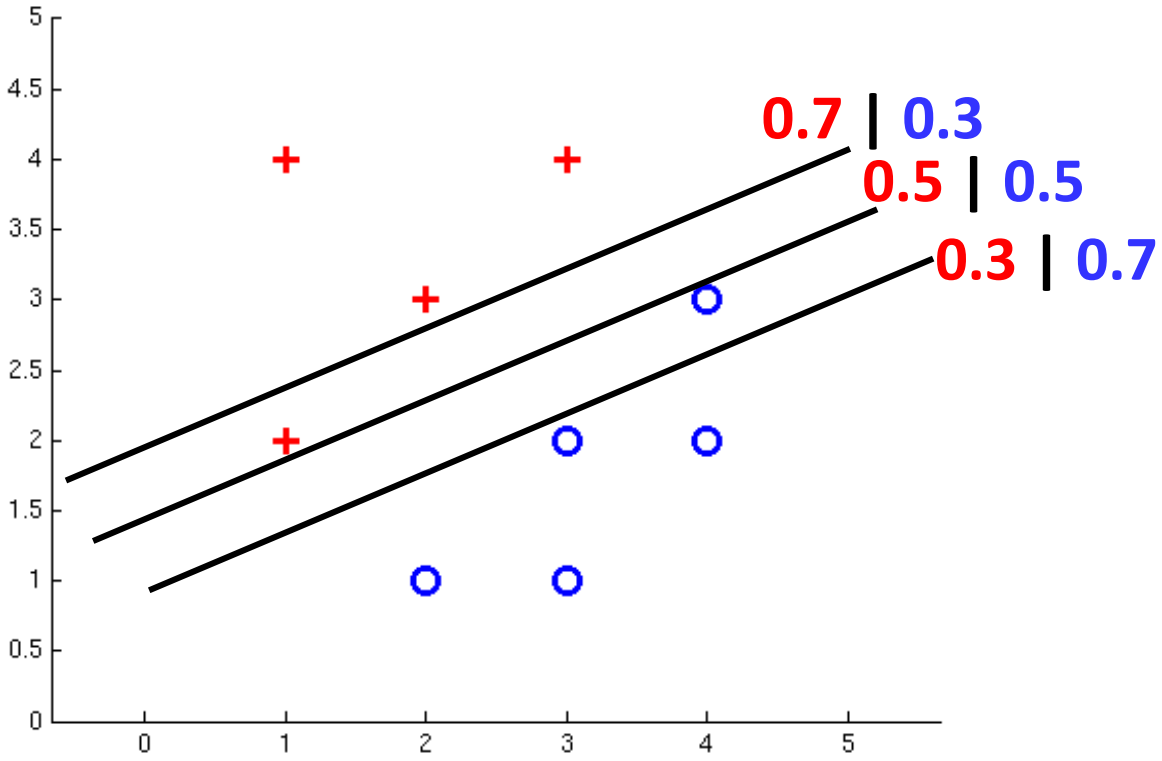
Separable Case: Deterministic Decision – Many Options





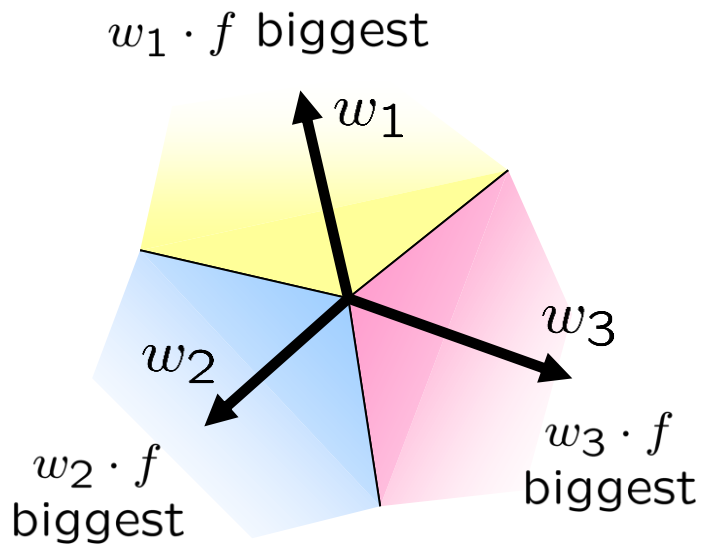
Separable Case: Probabilistic Decision – Clear Preference





Multiclass Logistic Regression

- Recall Perceptron:
  - A weight vector for each class: w_y
  - Score (activation) of a class y: $w_y \cdot f(x)$
  - Prediction highest score wins $y = \arg\max_y \ w_y \cdot f(x)$



- How to make the scores into probabilities?

$$\underbrace{z_1, z_2, z_3}_{\text{original activations}} \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}}_{\text{softmax activations}}$$

Best w?

- Maximum likelihood estimation:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

with:

$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

= Multi-Class Logistic Regression

Optimization

- How do we solve:

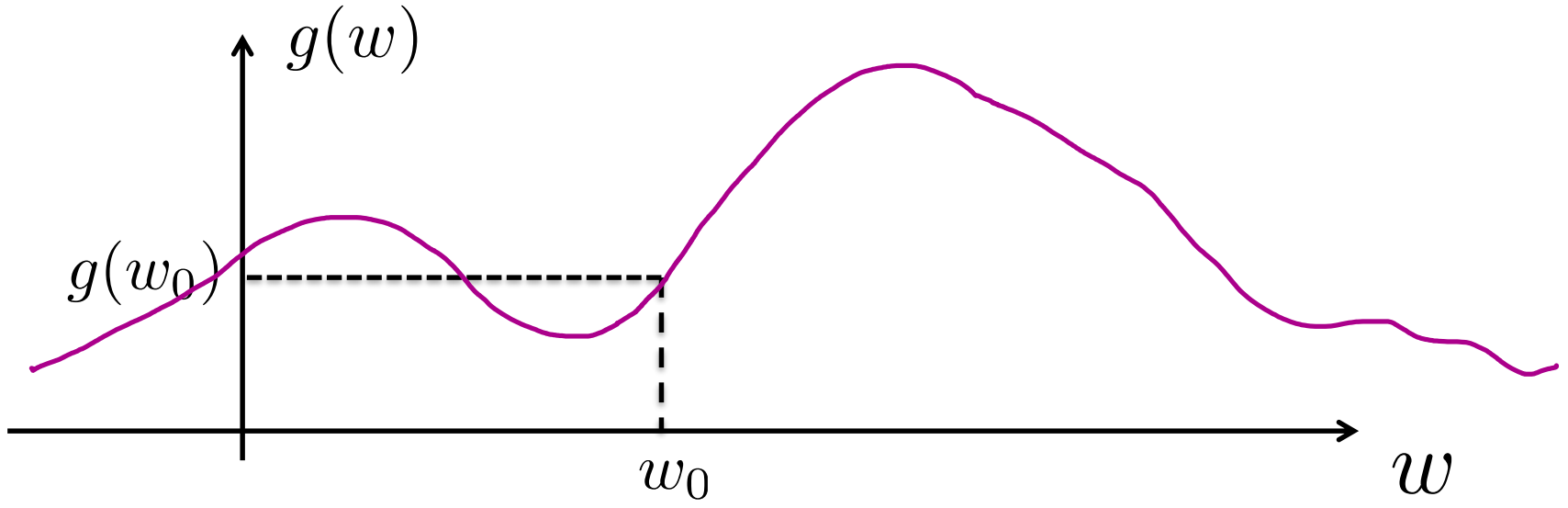
$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Hill Climbing

- Recall from CSP lecture: simple, general idea
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

- Can we do hill-climbing for multiclass logistic regression?
  - Optimization over a continuous space
    - Infinitely many neighbors!
    - How to do this efficiently?

1-D Optimization



- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction
- Or, evaluate derivative:

$$\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$$

  - Tells which direction to step into

Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider: $g(w_1, w_2)$
  - Updates:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

  - Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ **= gradient**

Gradient Ascent

- `init` w
- `for iter = 1, 2, …`

$$w \leftarrow w + \alpha * \nabla g(w)$$

- α : learning rate
  - A hyperparameter that needs to be chosen carefully

Gradient Ascent

- Start somewhere
- Repeat: Take a step in the gradient direction

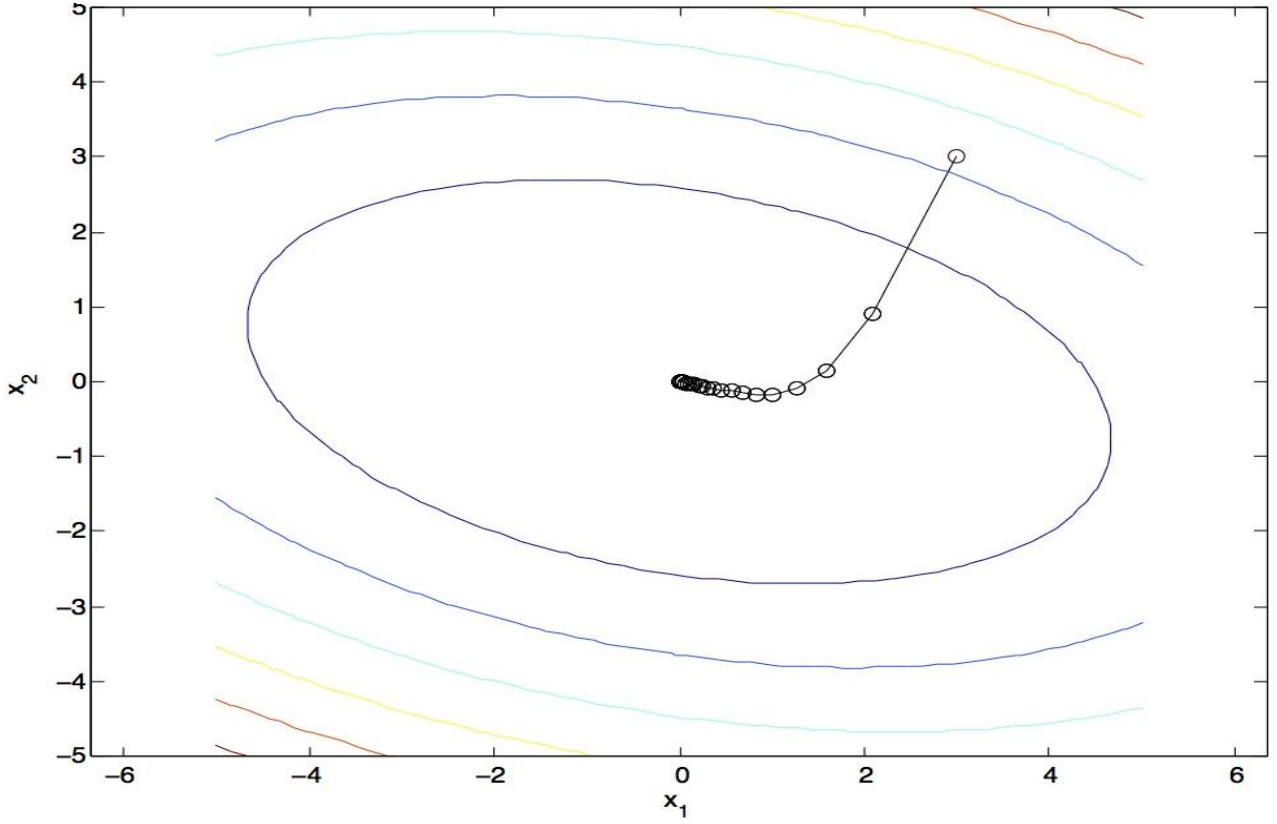


Figure source: Mathworks

Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \underbrace{\sum_i \log P(y^{(i)}|x^{(i)};w)}_{g(w)}$$

- init w
- for iter = 1, 2, …

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)};w)$$

What will gradient ascent do?

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)}; w)$$

$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

$$\nabla w_{y^{(i)}} f(x^{(i)}) - \nabla \log \sum_y e^{w_y f(x^{(i)})}$$

add f to weights of the correct class

Increase the score of the correct class

for y' weights: $\frac{1}{\sum_y e^{w_y f(x^{(i)})}} e^{w_{y'} f(x^{(i)})} f(x^{(i)})$

$P(y'|x^{(i)}; w) f(x^{(i)})$

Decrease the scores of all the classes

subtract f from y' weights in proportion to the current probability of y'

Stochastic Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Idea: once gradient on one training example has been computed, might as well update before computing next one

- `init` w
- `for iter = 1, 2, …`
  - `pick random j`

$$w \leftarrow w + \alpha * \nabla \log P(y^{(j)}|x^{(j)}; w)$$

Mini-Batch Gradient Ascent on the Log Likelihood Objective

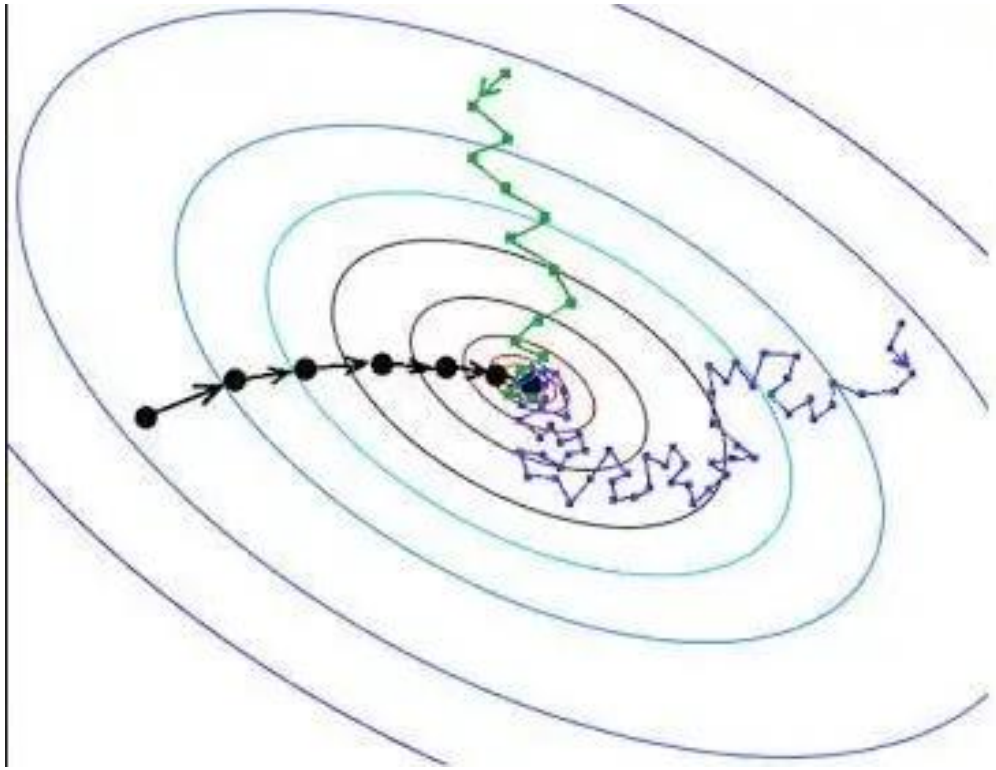
$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Idea: gradient over a small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- `init` w
- `for iter = 1, 2, …`
  - `pick random subset of training examples J`

$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$$

Batch vs. Stochastic vs. Mini-batch GD



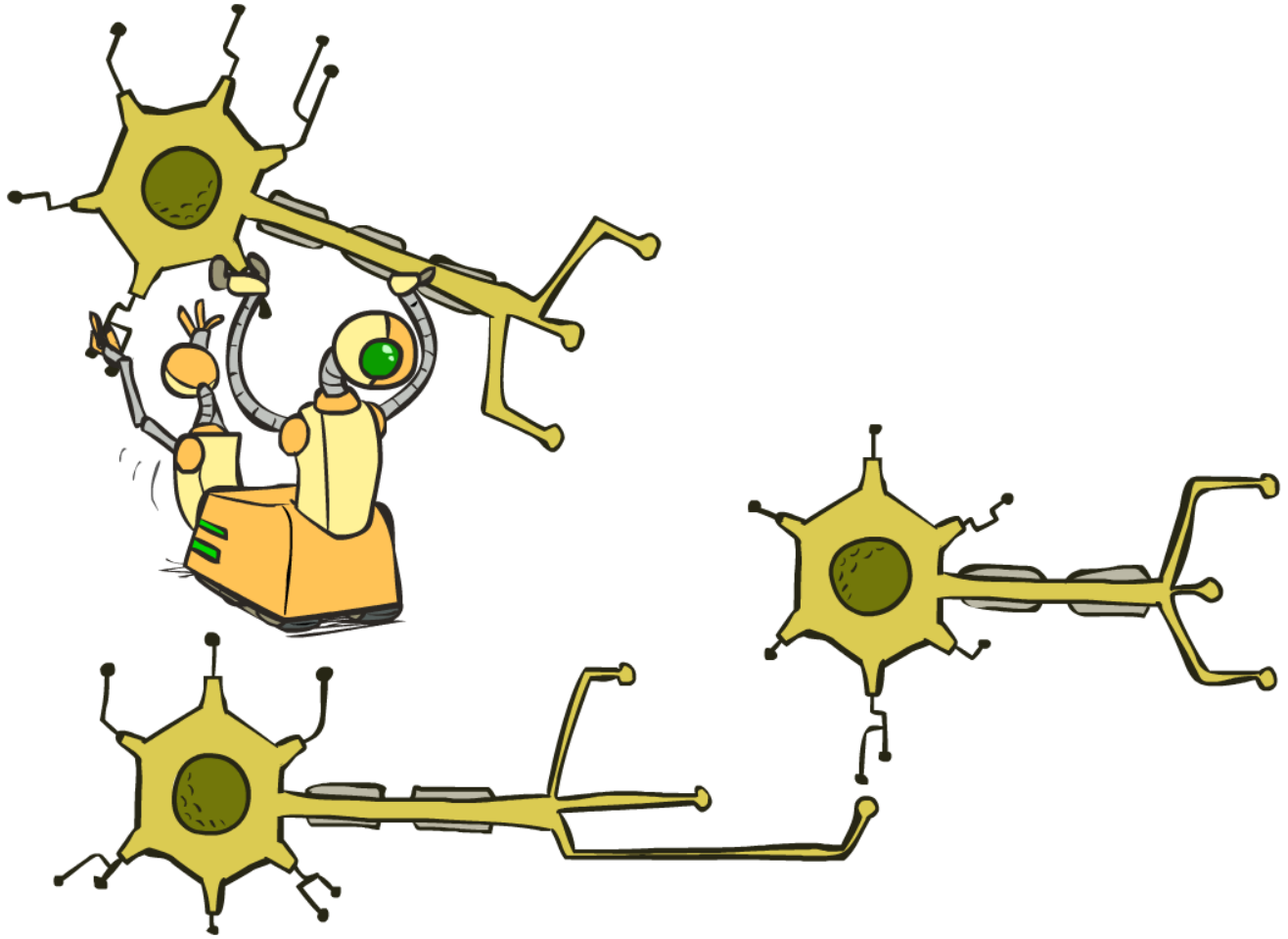
Batch GD
- Slowest
- Perfect gradient

Stochastic GD
- Fastest
- Rough-estimate grad

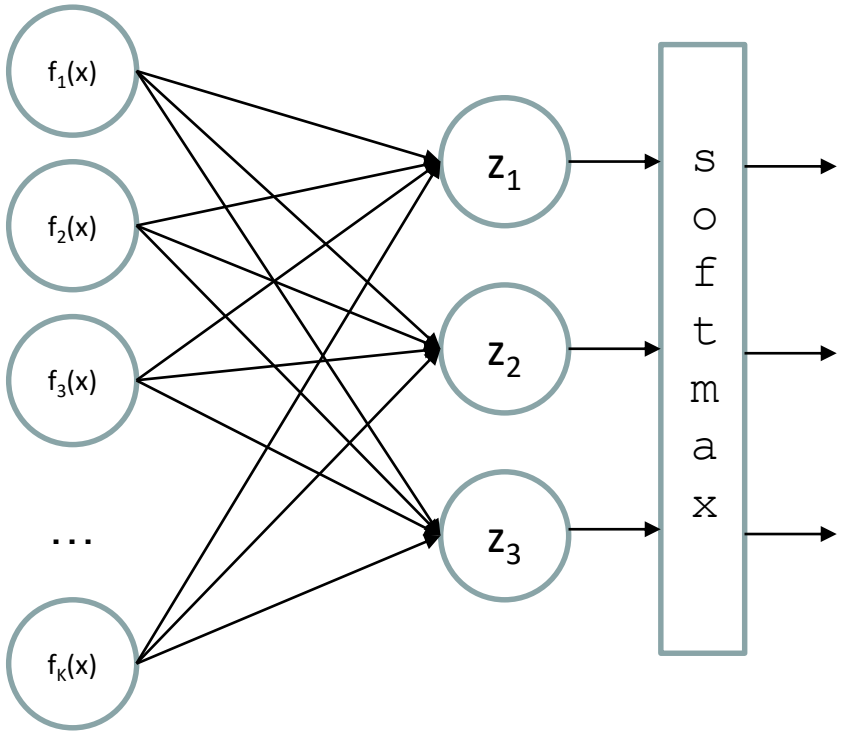
Mini-batch GD
- Compromise

https://sweta-nit.medium.com/batch-mini-batch-and-stochastic-gradient-descent-e9bc4cacd461

Neural Networks



Multi-class Logistic Regression

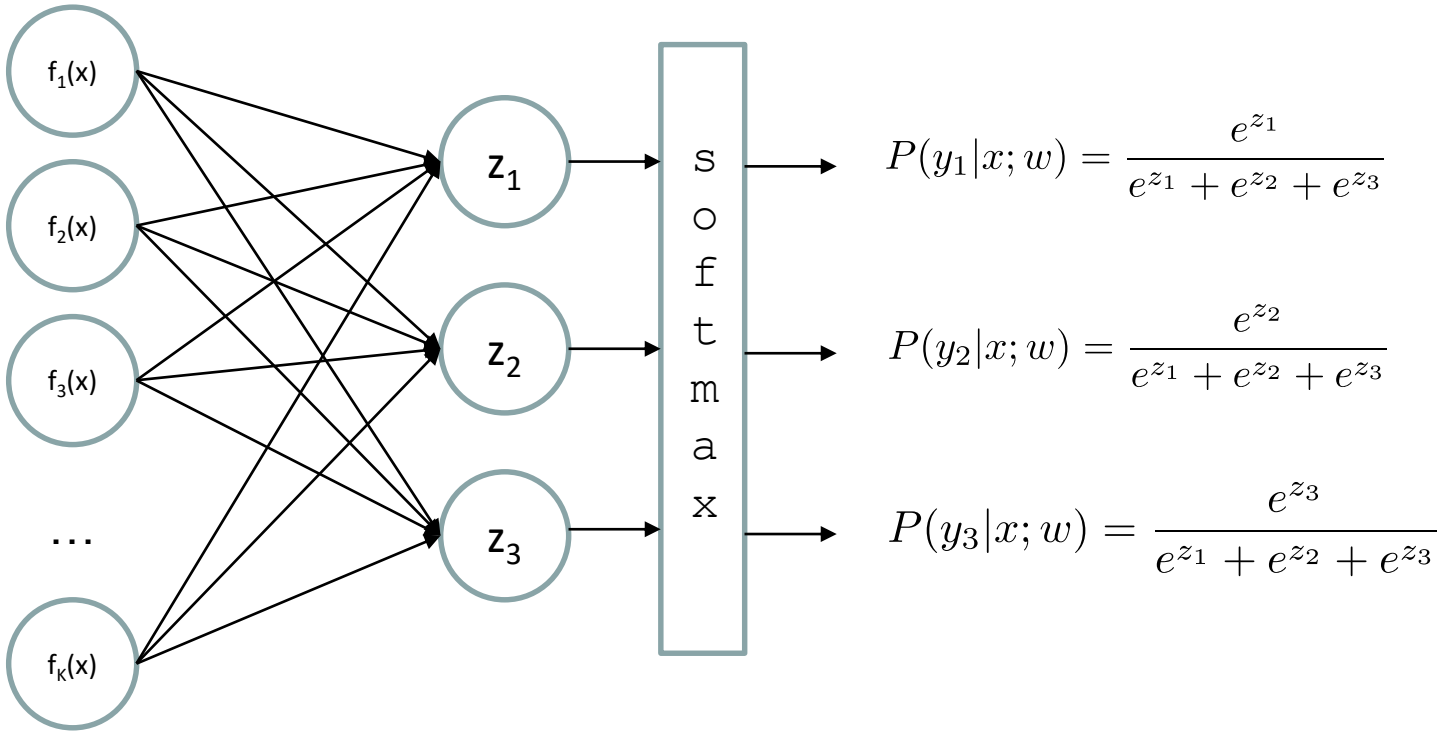


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

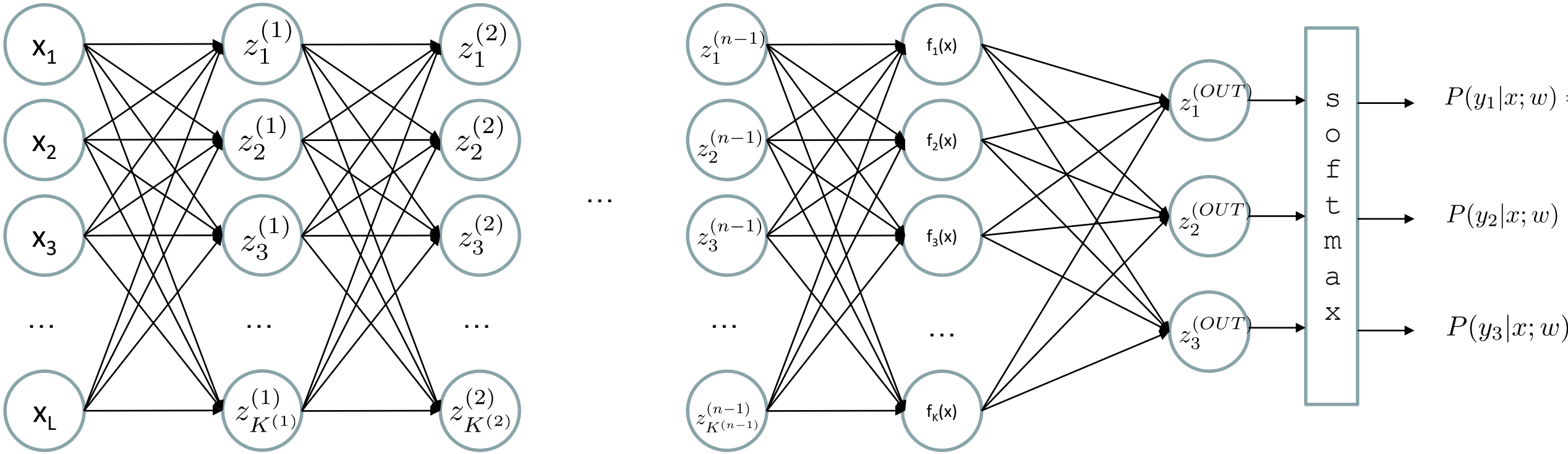
$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Deep Neural Network = Also learn the features!



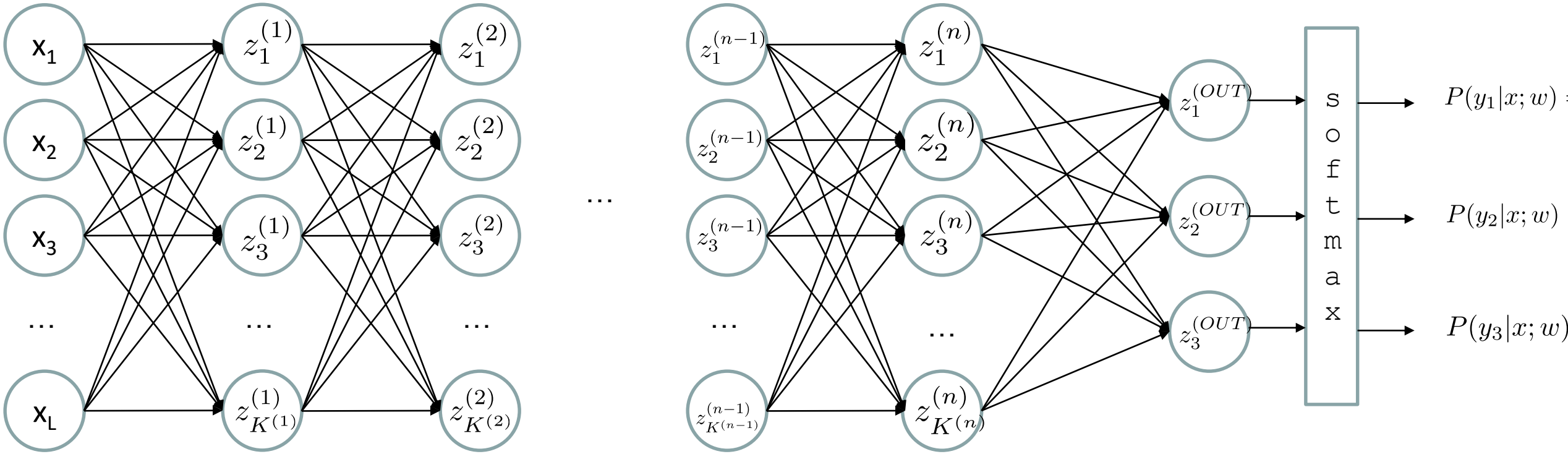
Deep Neural Network = Also learn the features!



$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Deep Neural Network = Also learn the features!



$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function