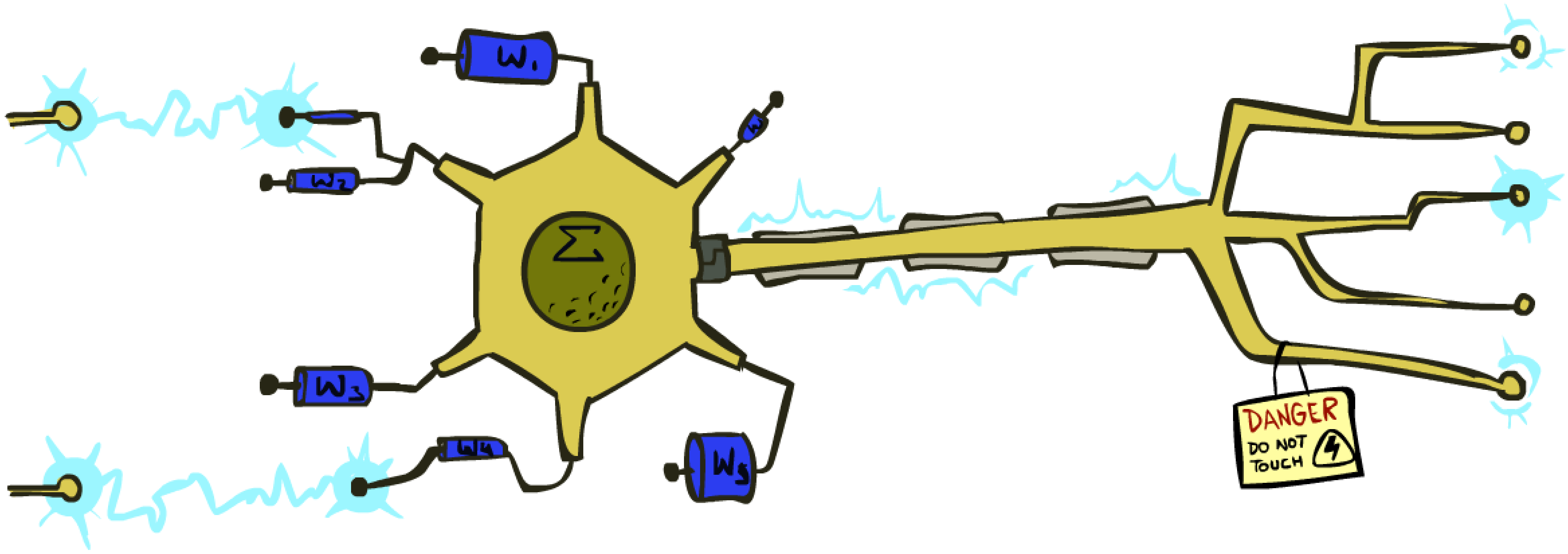
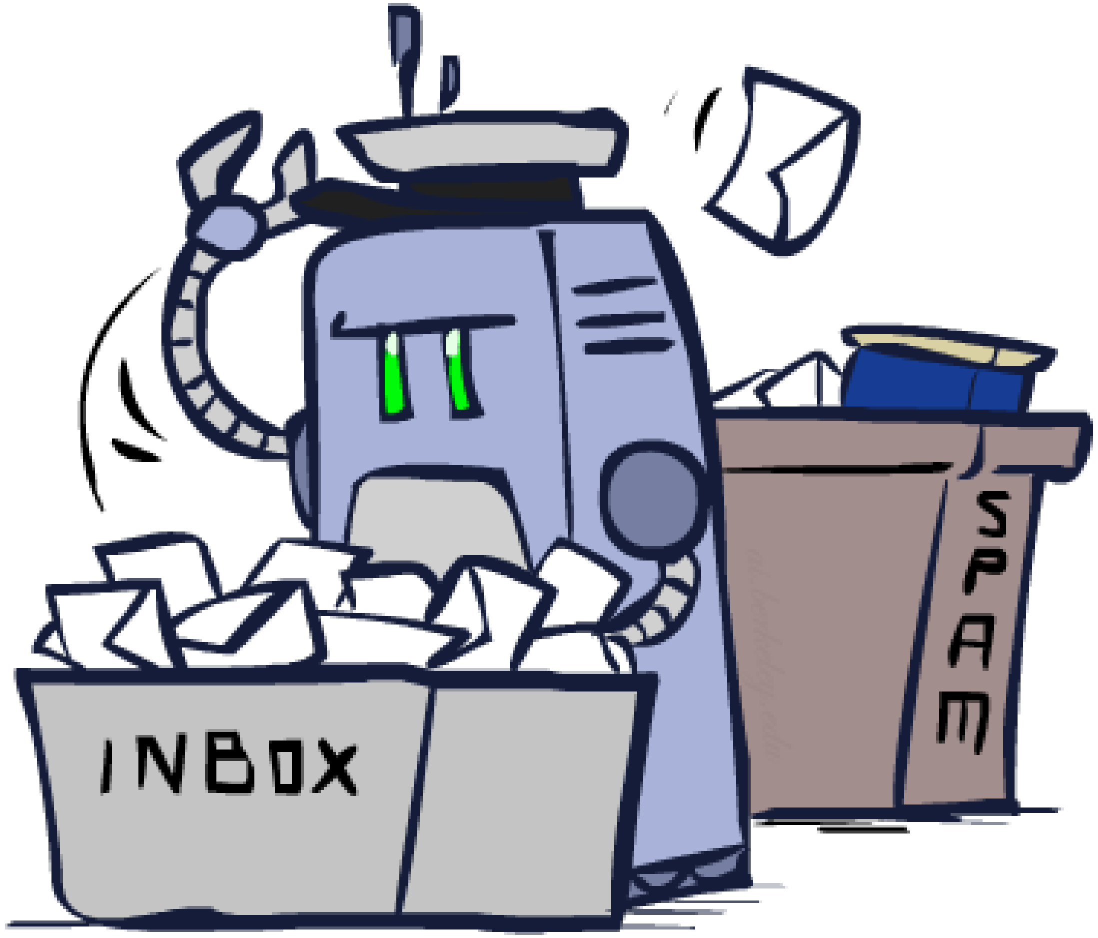
CS 188: Artificial Intelligence

Discriminative Learning



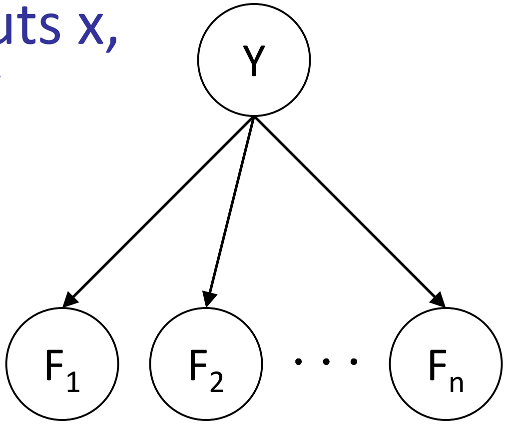
Instructors: Sergey Levine and Stuart Russell --- University of California, Berkeley

Classification



Last Time

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



- Naïve Bayes

$$P(Y|F_{0,0} \dots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$$

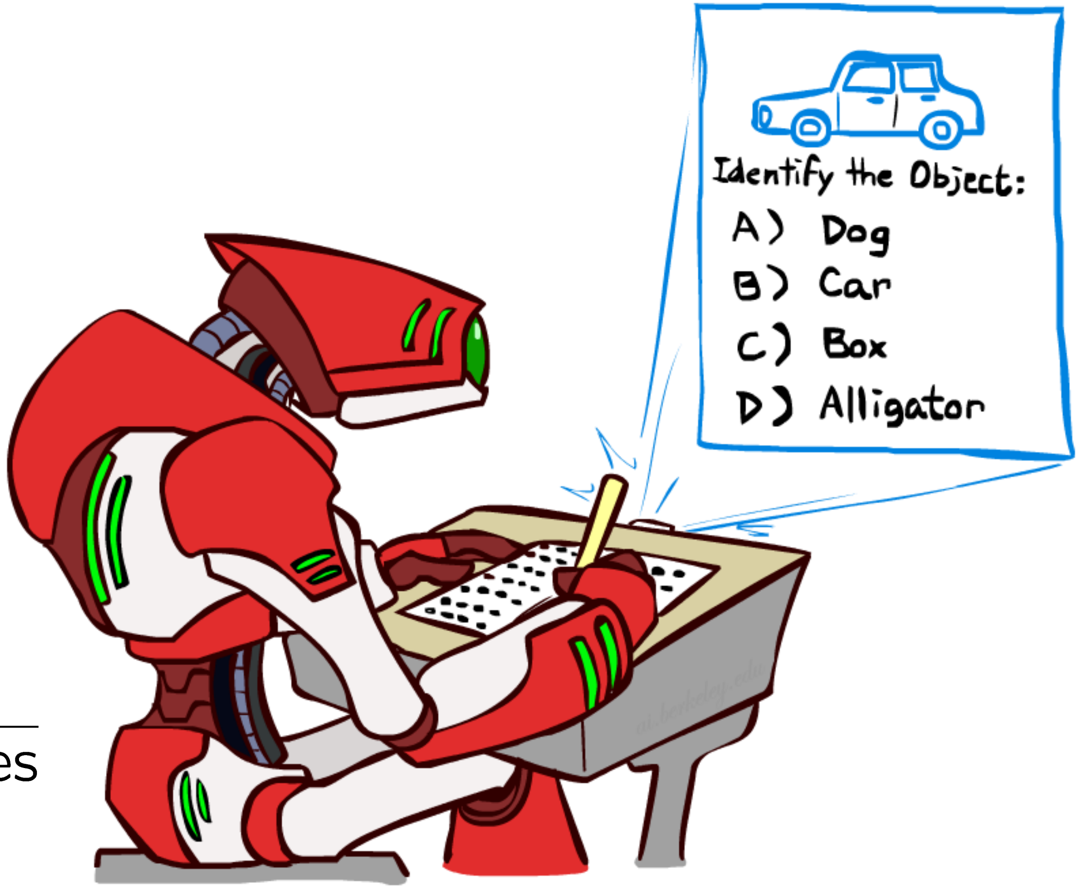
- Parameter estimation:
  - MLE, MAP, priors

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

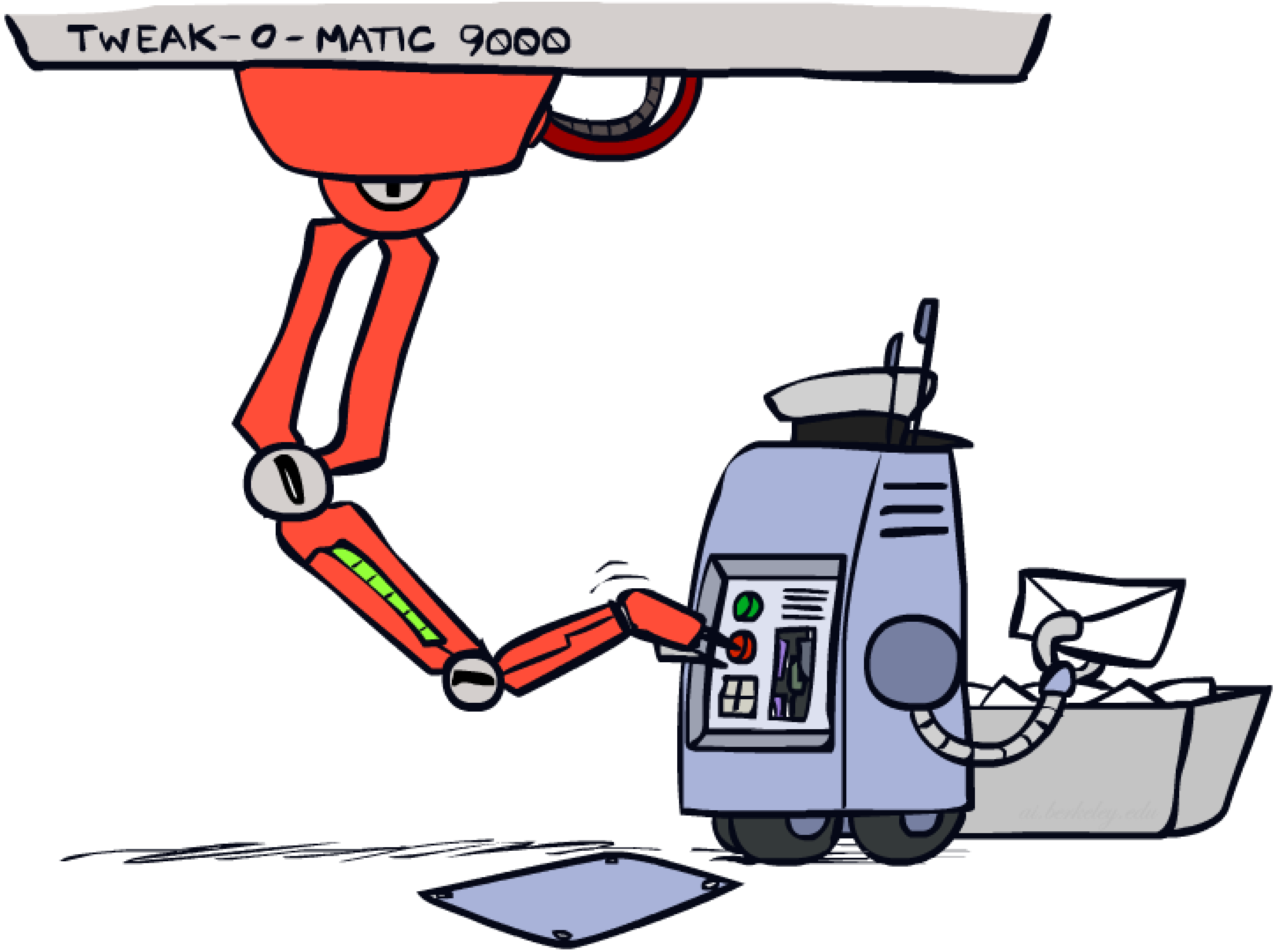
- Laplace smoothing

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

- Training set, held-out set, test set

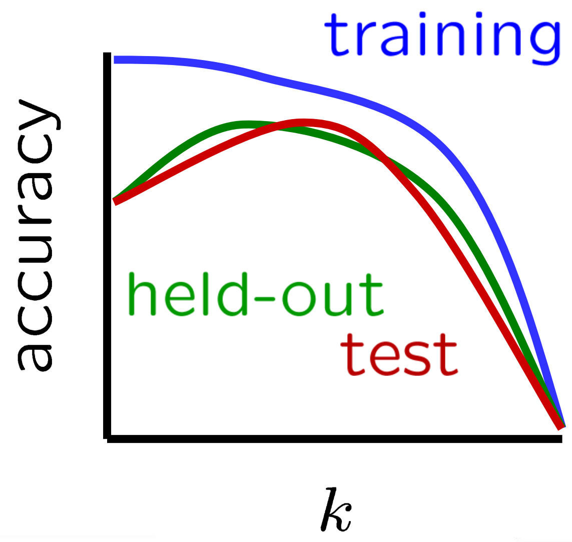


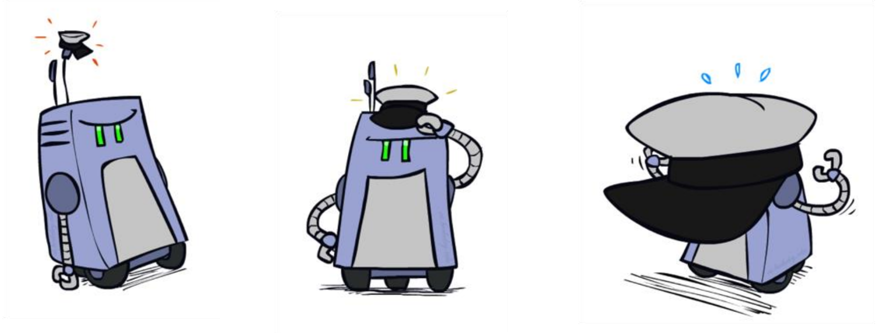
Tuning



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 hyperparameters, train and test on the held-out data
  - Choose the best value and do a final test on the test data





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

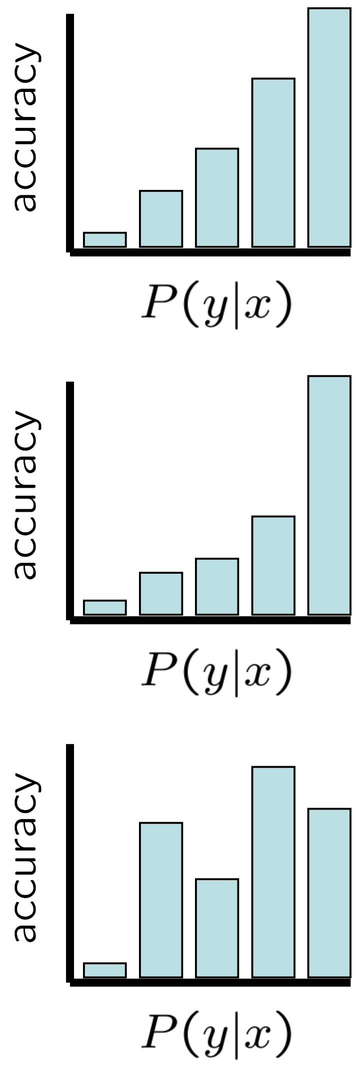
Confidences from a Classifier

- 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?



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?

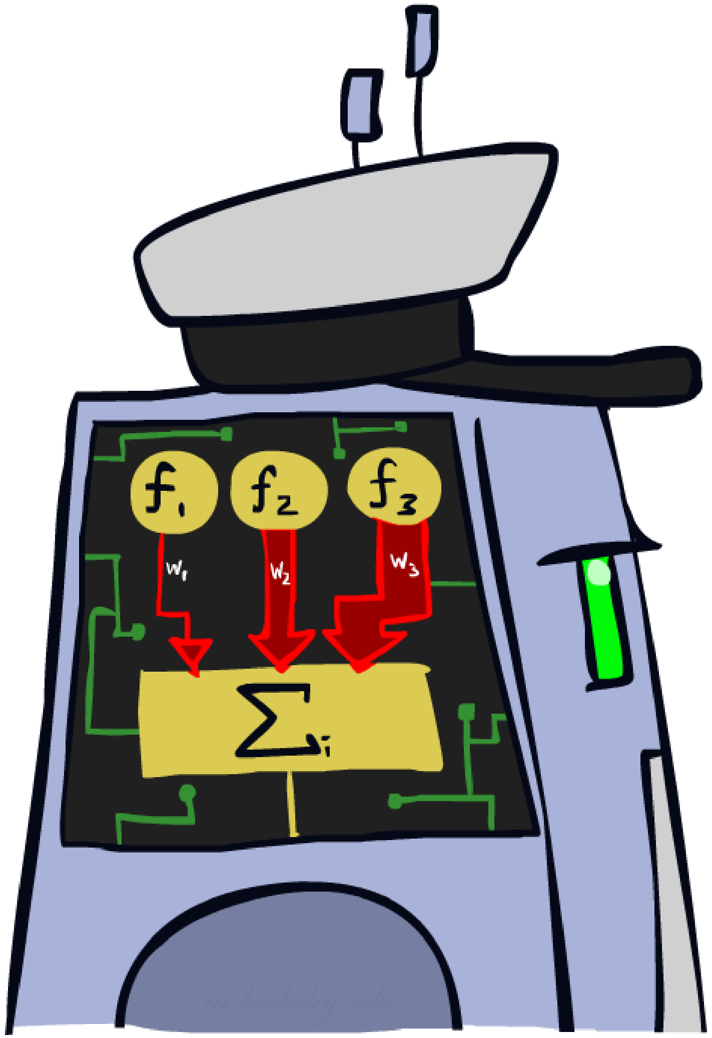
- Can add these information sources as new variables in the NB model

- ...but NB must *model* all of the features

- Features often not independent, NB is not a good model in this case

Error-Driven Classification

Linear Classifiers



Feature Vectors

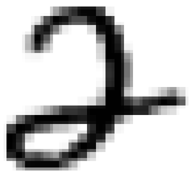
x $\quad$ $f(x)$ $\quad$ y

```
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
+

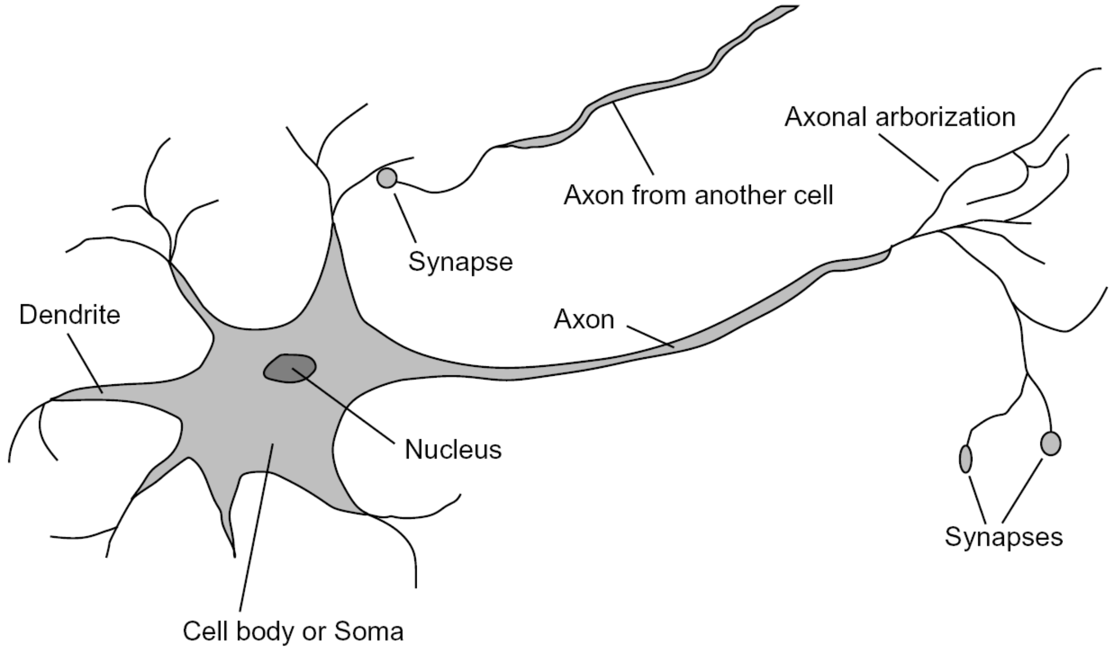


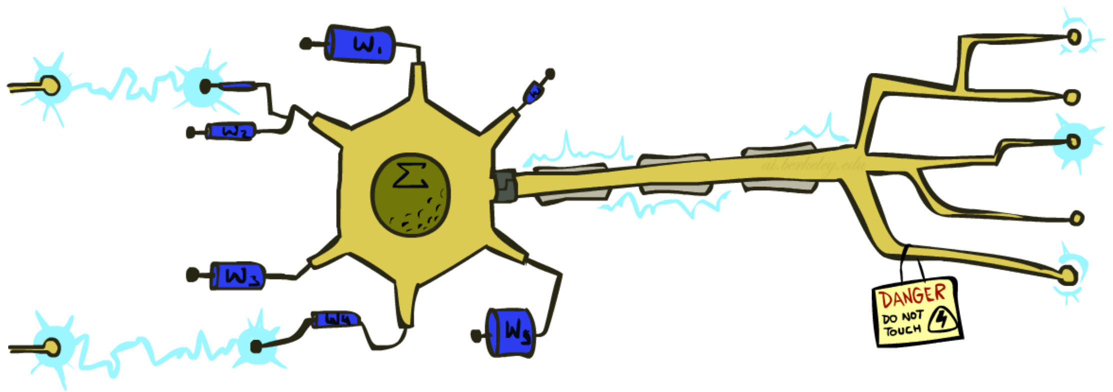
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

“2”

Some (Simplified) Biology

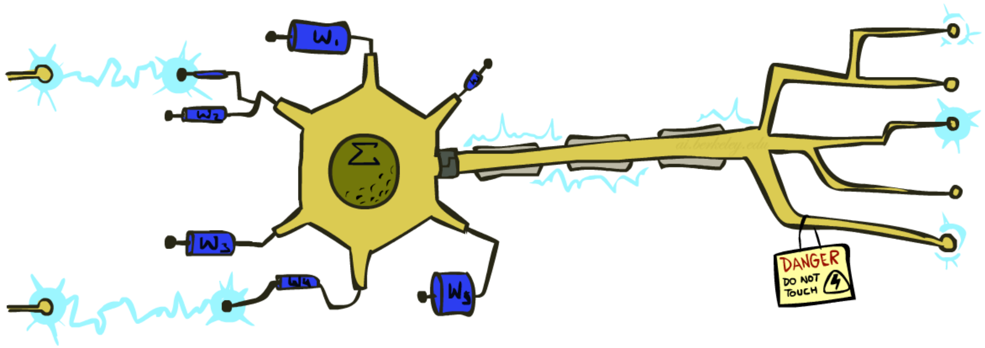
- Very loose inspiration: human neurons





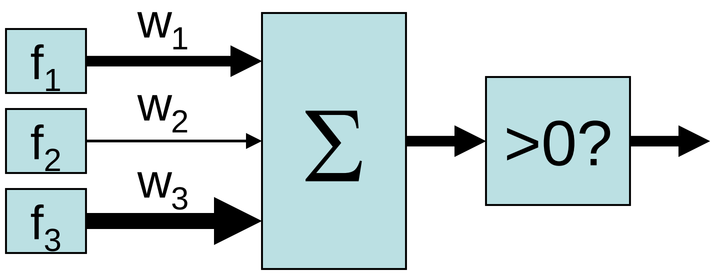
Linear Classifiers

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



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

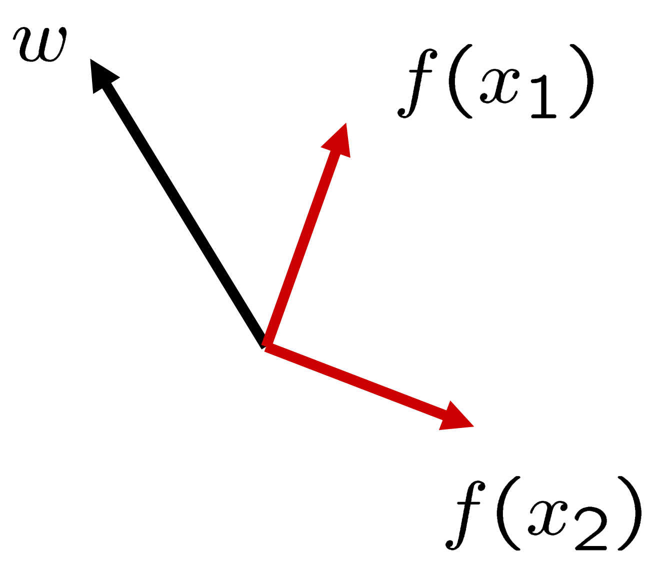
- If the activation is:
  - Positive, output +1
  - Negative, output -1



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
...
```

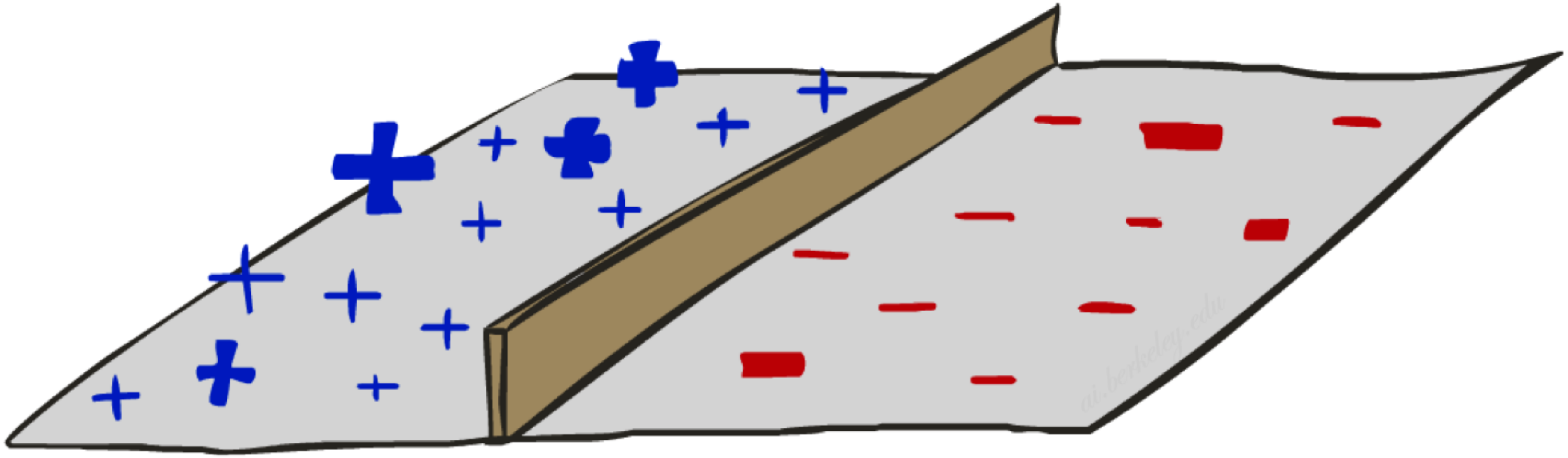



```
# free      : 2
YOUR_NAME   : 0
MISSPELLED  : 2
FROM_FRIEND : 0
...
```

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

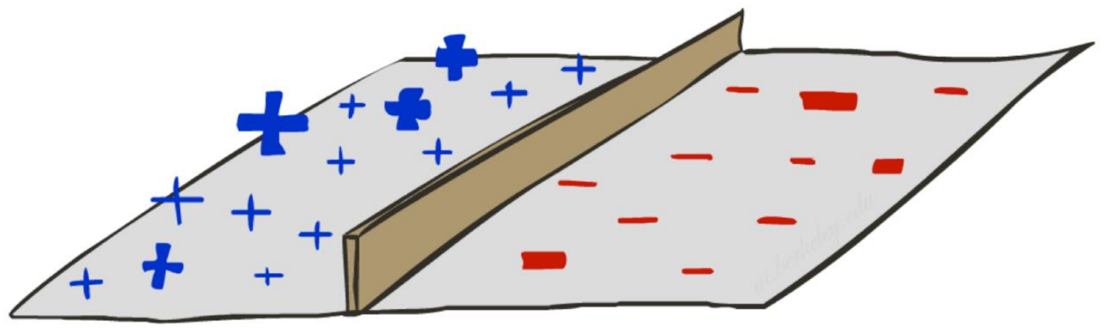
Dot product $w \cdot f$ positive means the positive class

Decision Rules



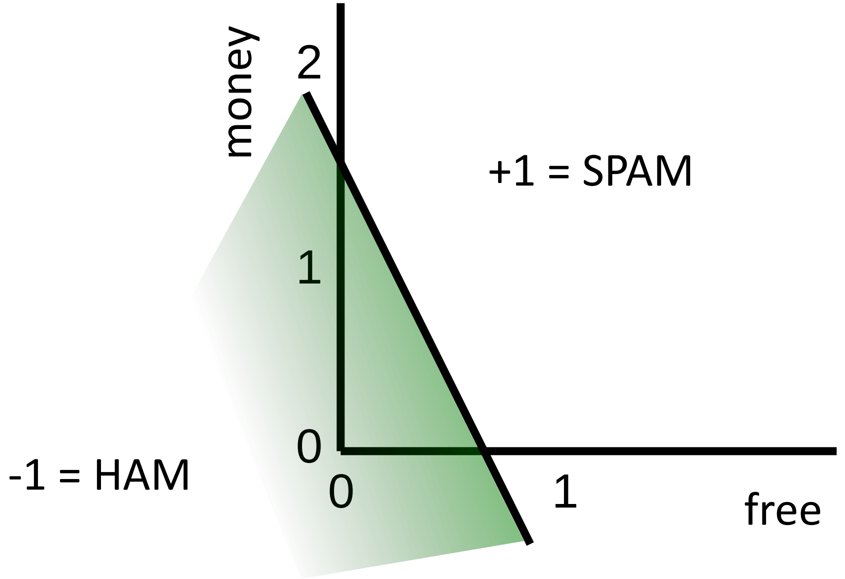
Binary Decision Rule

- 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
...
```

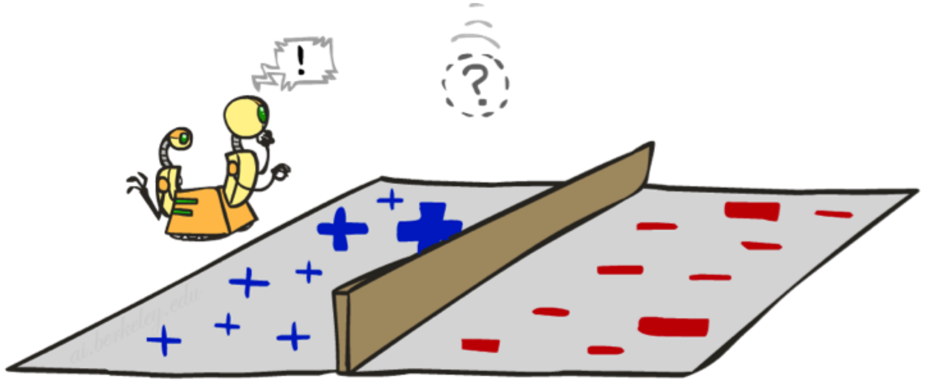


$f \cdot w = 0$

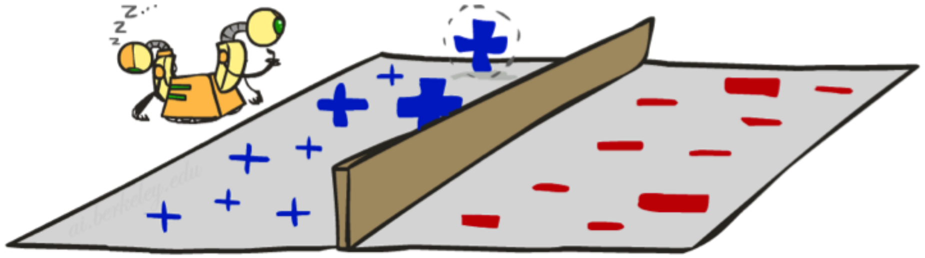
Weight Updates

Learning: Binary Perceptron

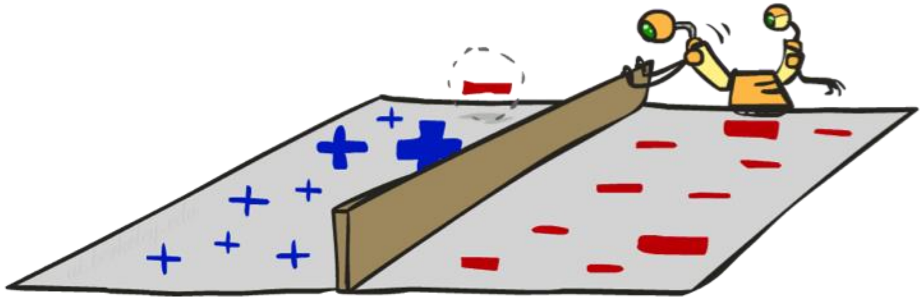
- Start with weights = 0
- For each training instance:
  - Classify with current weights



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



  - If wrong: adjust the weight vector



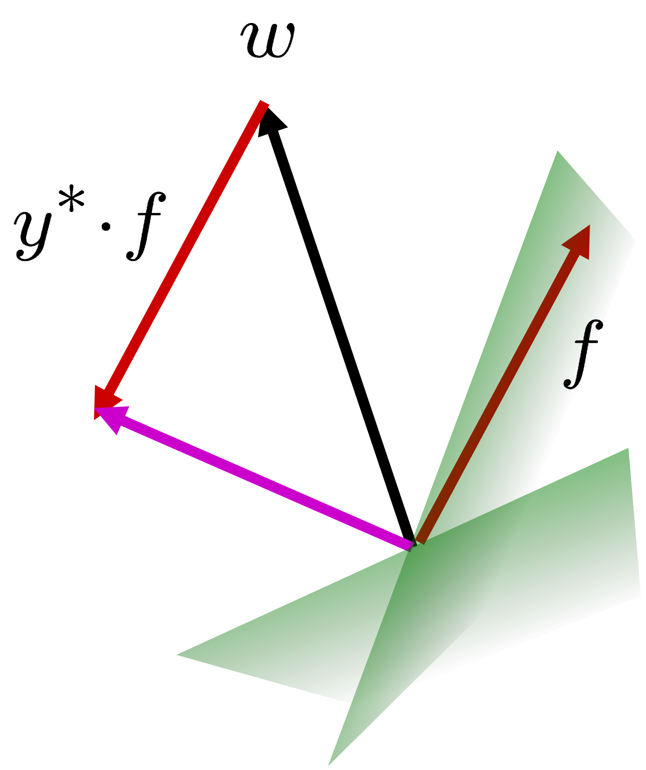
Learning: Binary Perceptron

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

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

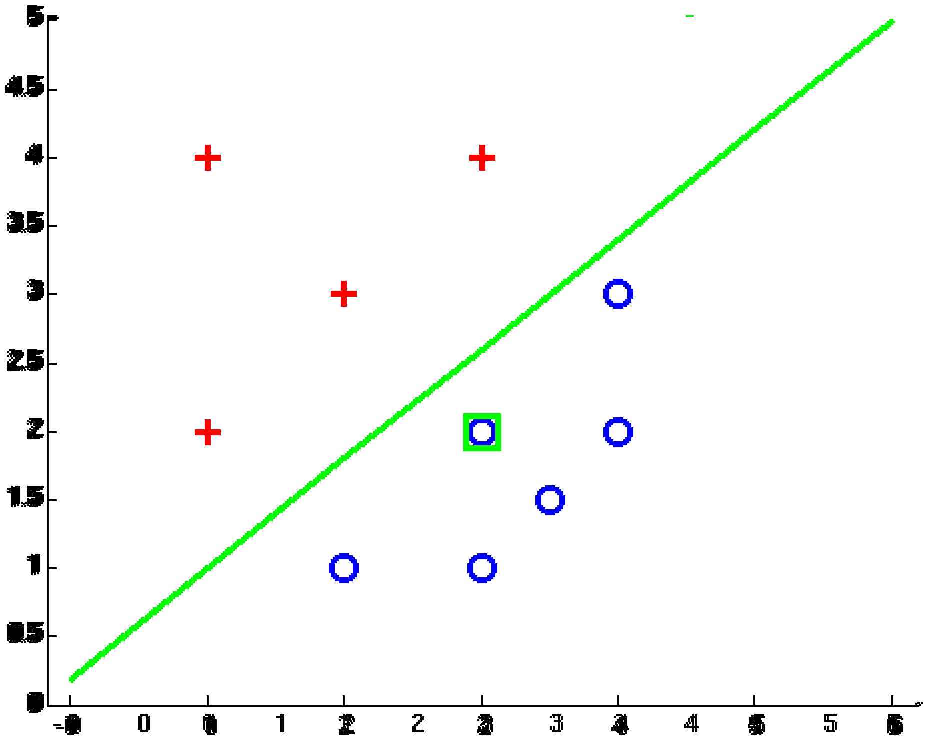
  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

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



Examples: Perceptron

- Separable Case



Multiclass Decision Rule

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

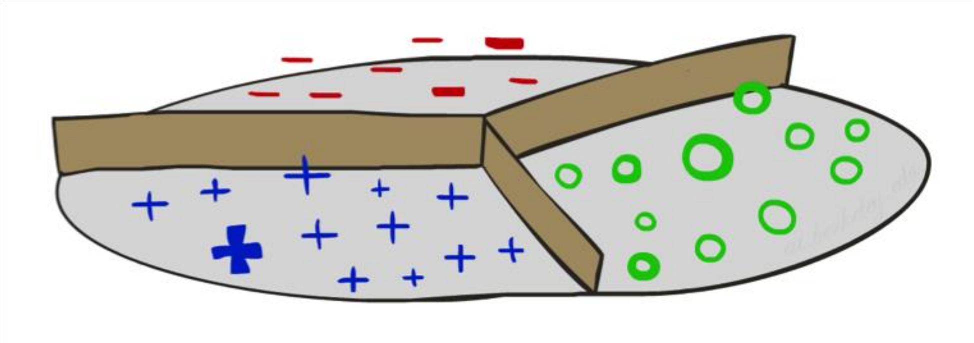
    $$w_y$$

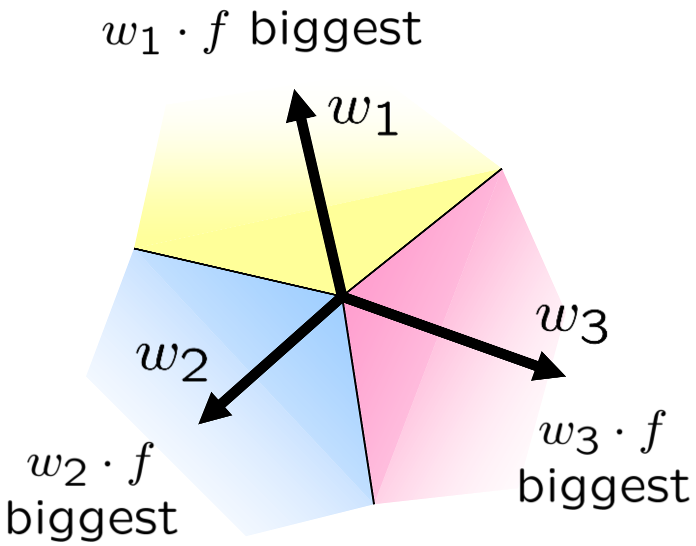
  - Score (activation) of a class y:

    $$w_y \cdot f(x)$$

  - 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

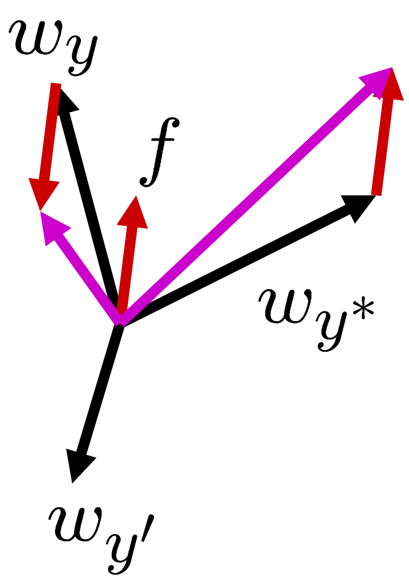
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

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

- 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}

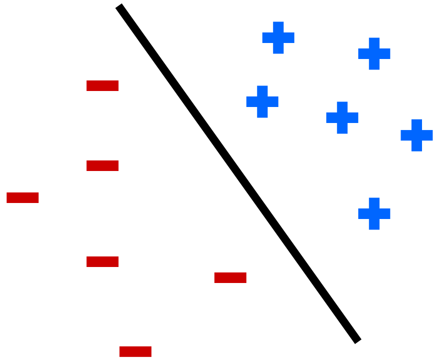
```
BIAS  : 0
win   : 0
game  : 0
vote  : 0
the   : 0
...
```

Properties of Perceptrons

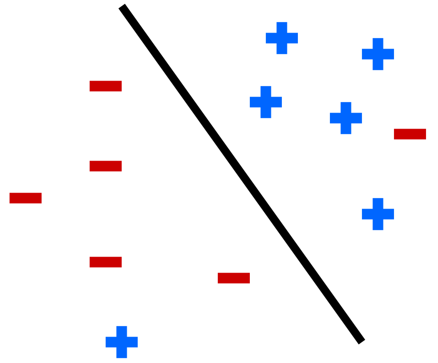
- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

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

Separable

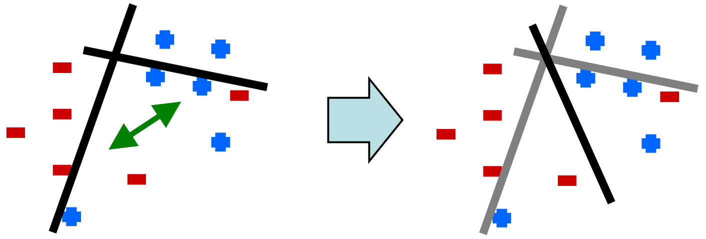


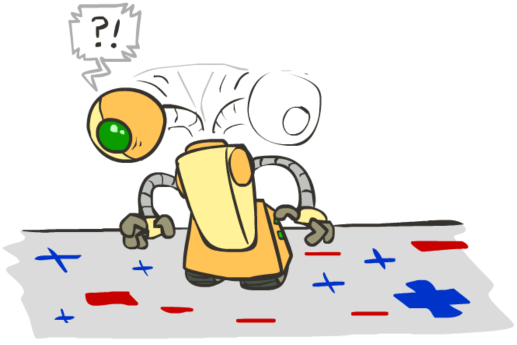
Non-Separable



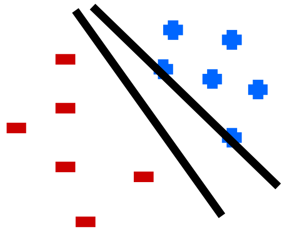
Problems with the Perceptron

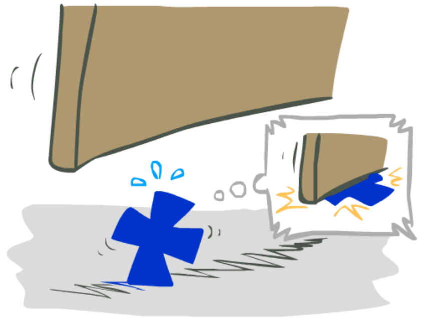
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



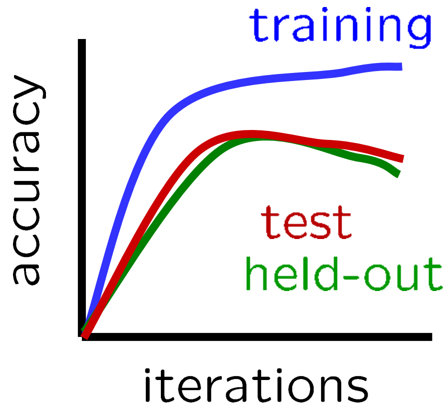


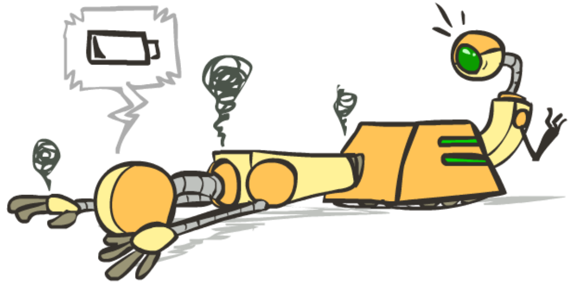
- Mediocre generalization: finds a "barely" separating solution



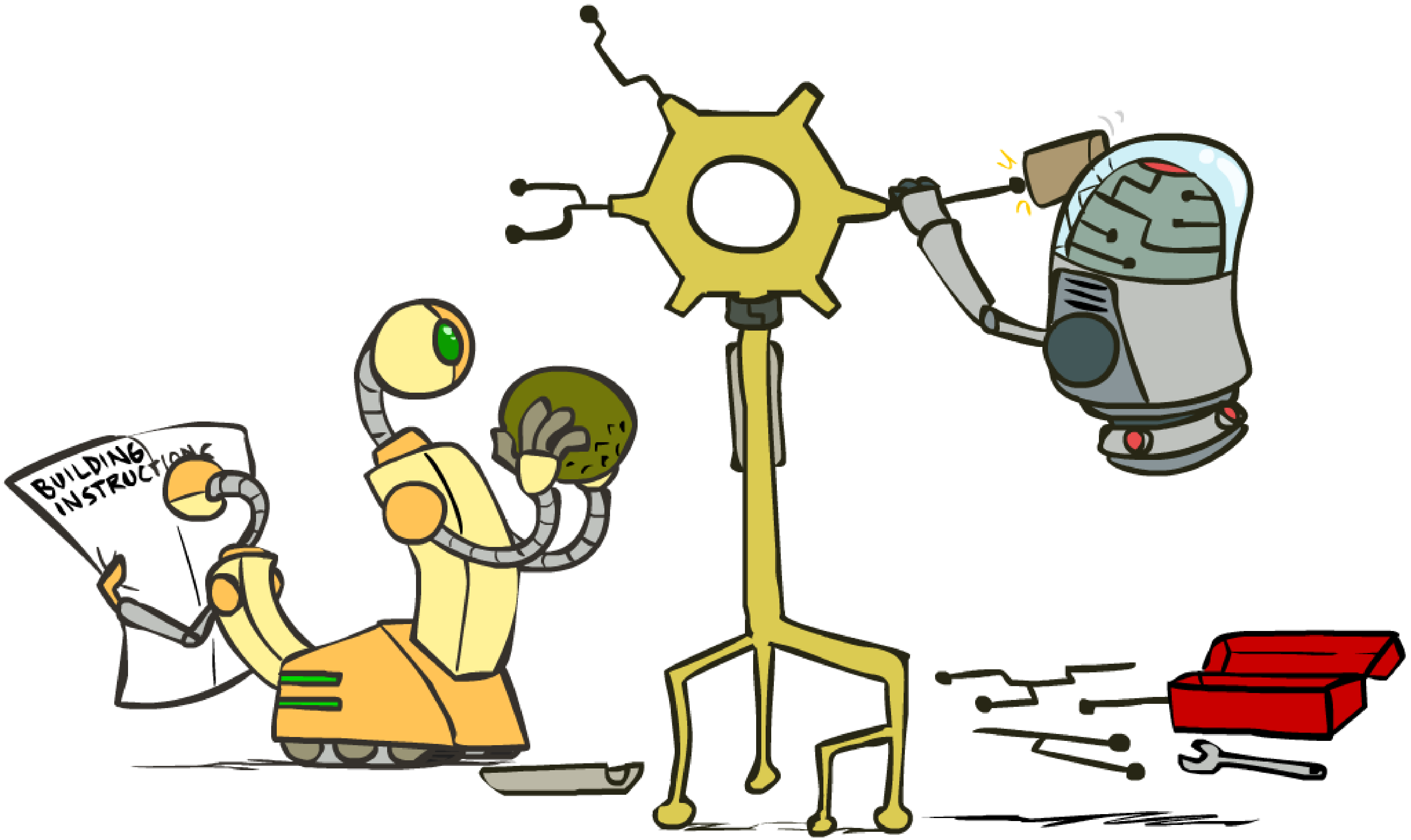


- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting





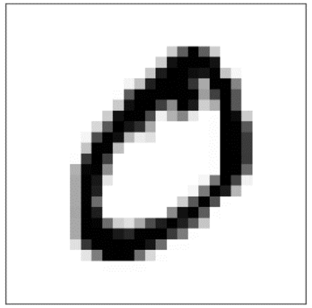
Improving the Perceptron



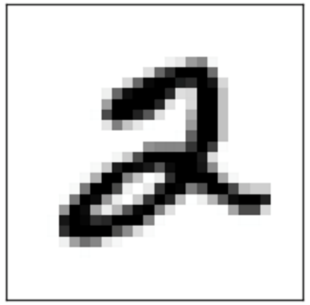
Probabilistic Classification

- Naïve Bayes provides *probabilistic* classification

$$\text{Answers the query: } P(Y = y_i | x_1, \ldots, x_n)$$



1: 0.001
2: 0.001
…
0: 0.991

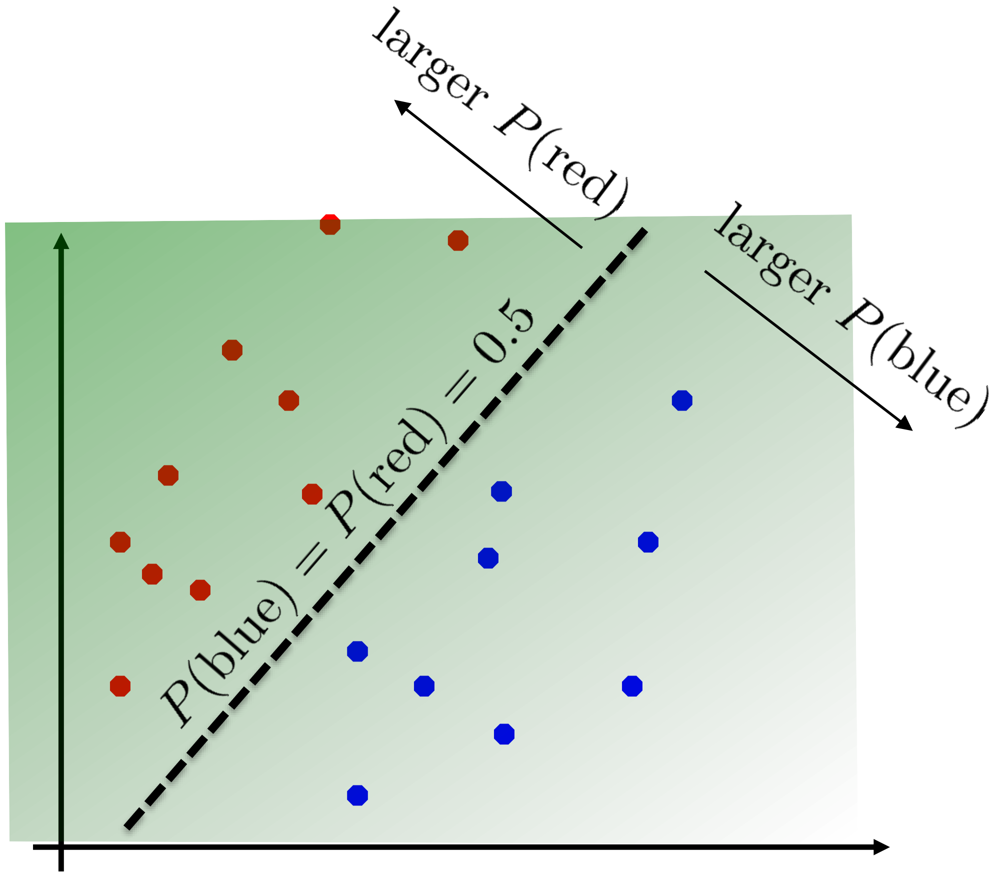


1: 0.001
2: 0.703
…
6: 0.264
…
0: 0.001

- Perceptron just gives us a class prediction
  - Can we get it to give us probabilities?
  - Turns out it also makes it easier to train!

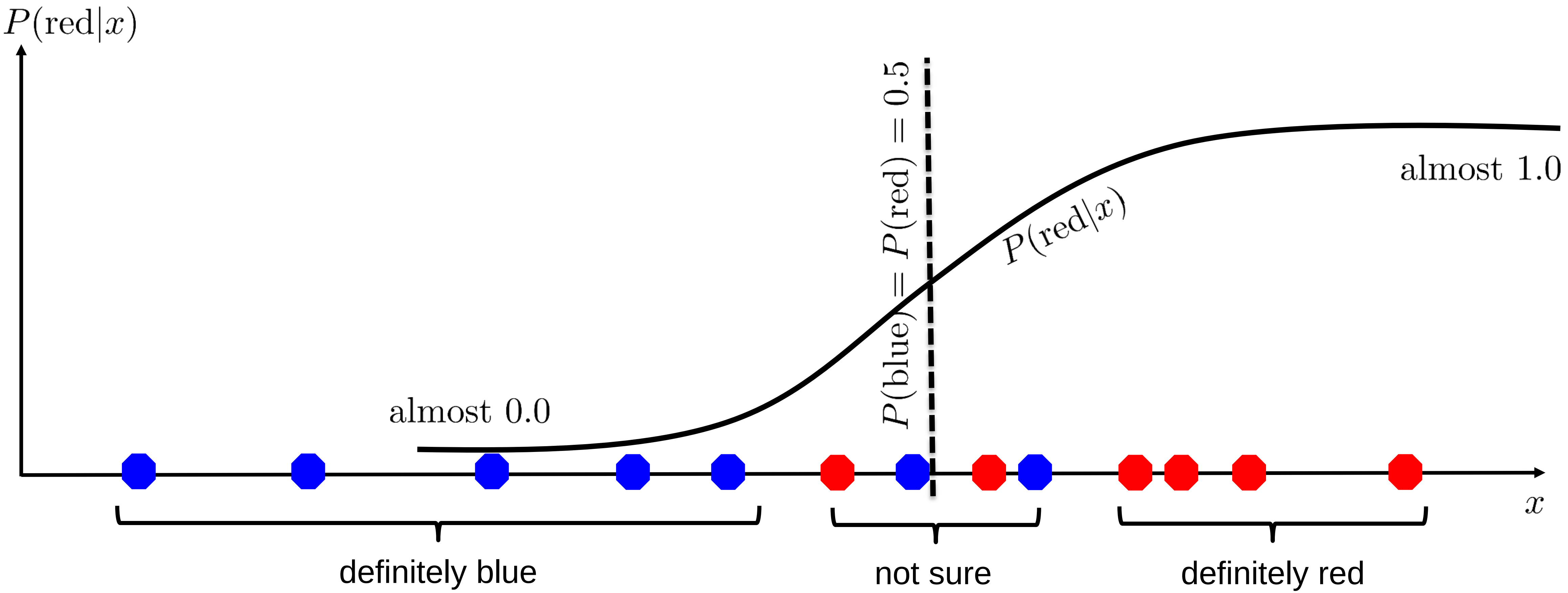
Note: I'm going to be lazy and use "x" in place of "f(x)" here – this is just for notational convenience!

A *Probabilistic* Perceptron



As $w_y \cdot x$ gets bigger, $P(y|x)$ gets bigger

A 1D Example

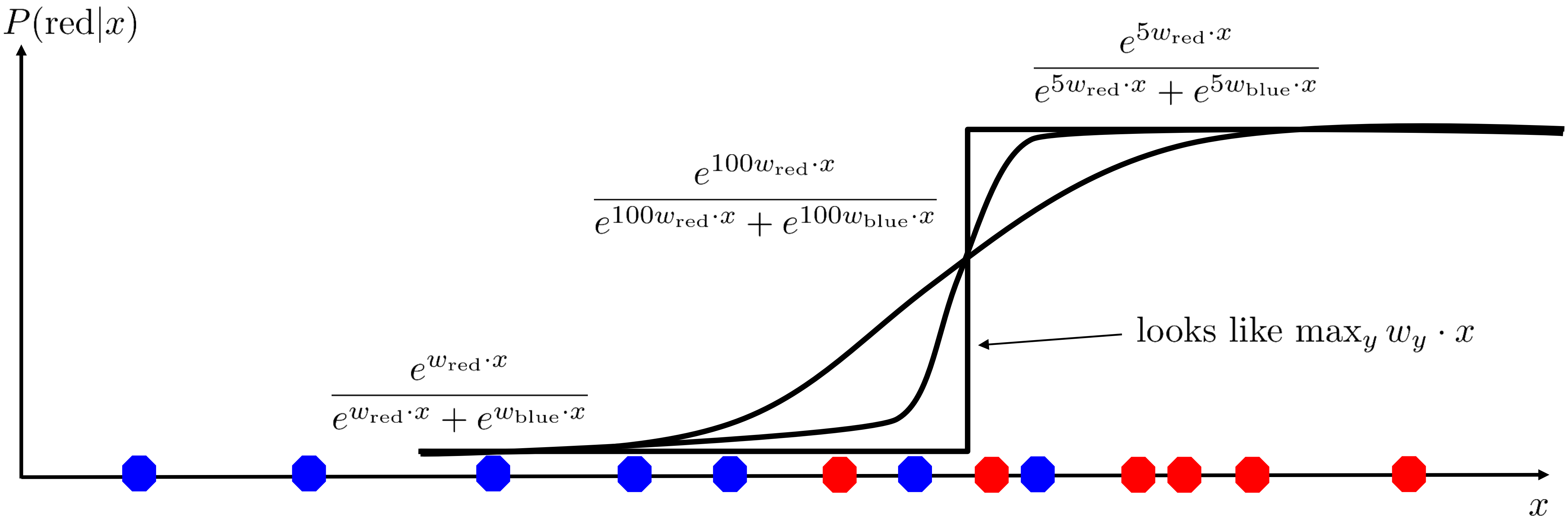


$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$

probability increases exponentially as we move away from boundary

normalizer

The *Soft* Max

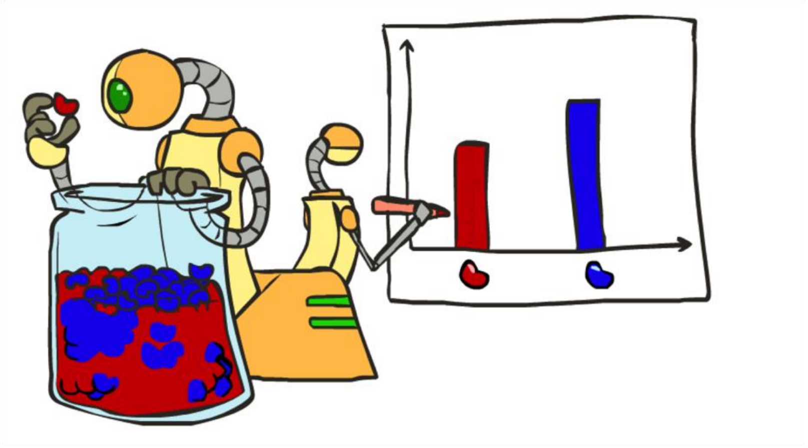


$$P(\text{red}|x) = \frac{e^{w_{\text{red}}\cdot x}}{e^{w_{\text{red}}\cdot x} + e^{w_{\text{blue}}\cdot x}}$$

How to Learn?

- Maximum likelihood estimation

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



- Maximum *conditional* likelihood estimation

$$\theta^\star = \arg\max_\theta P(\mathbf{Y}|\mathbf{X},\theta)$$
$$= \arg\max_\theta \prod_i \underbrace{P_\theta(y_i|x_i)}$$

$$\ell(w) = \prod_i \frac{e^{w_{y_i}\cdot x_i}}{\sum_y e^{w_y \cdot x_i}}$$

$$\ell\ell(w) = \sum_i \log P_w(y_i|x_i)$$
$$= \sum_i w_{y_i}\cdot x_i - \log \sum_y e^{w_y\cdot x_i}$$

Local Search

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
  - Neighbors = small perturbations of w

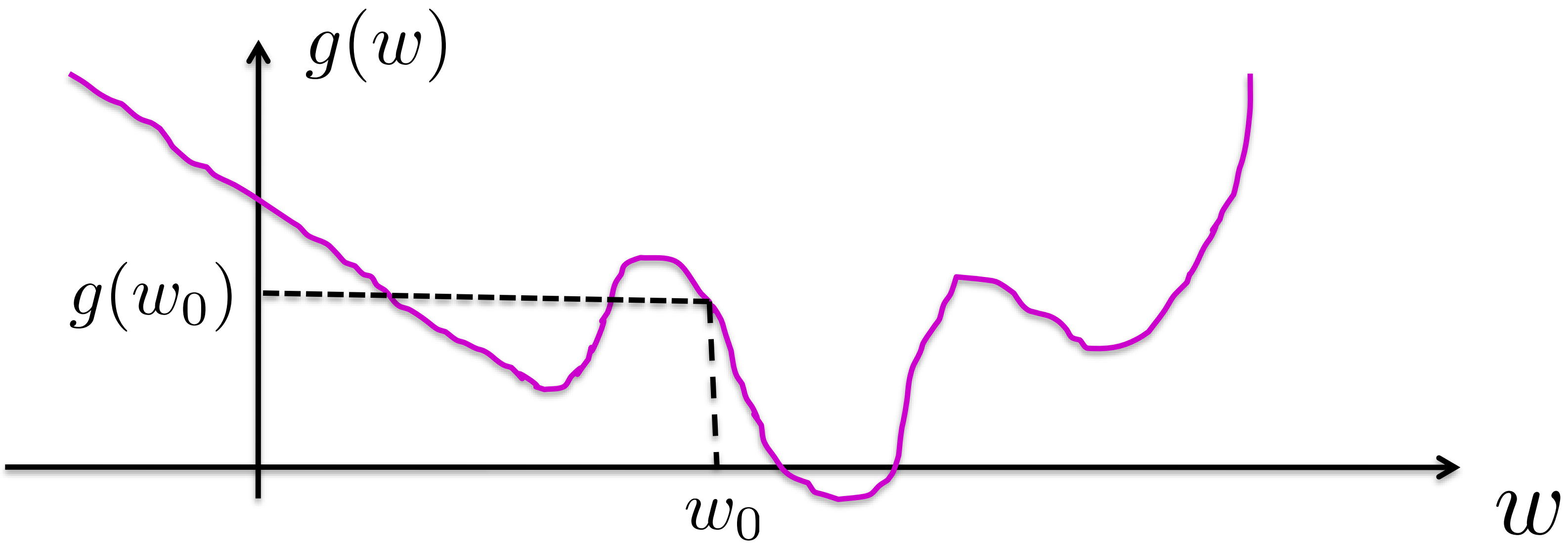
Our Status

- Our objective $ll(w)$
- Challenge: how to find a good *w* ?
- Equivalently:

$$\max_w ll(w)$$

$$\min_w -ll(w)$$

1D optimization

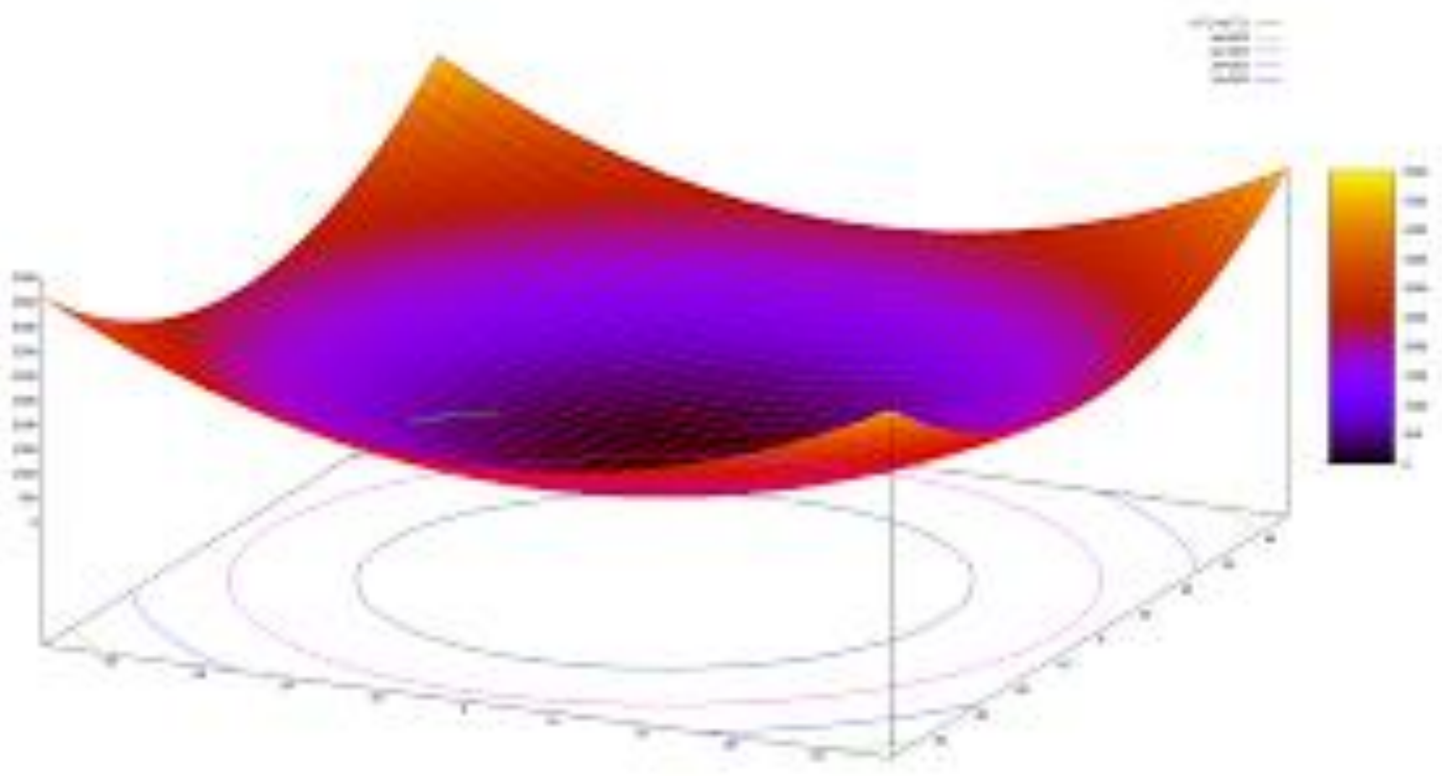


- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction

- Or, evaluate derivative: $\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$

  - Which tells which direction to step into

2-D Optimization



Source: Thomas Jungblut’s Blog

Steepest Descent

- Idea:
  - Start somewhere
  - Repeat: Take a step in the steepest descent direction

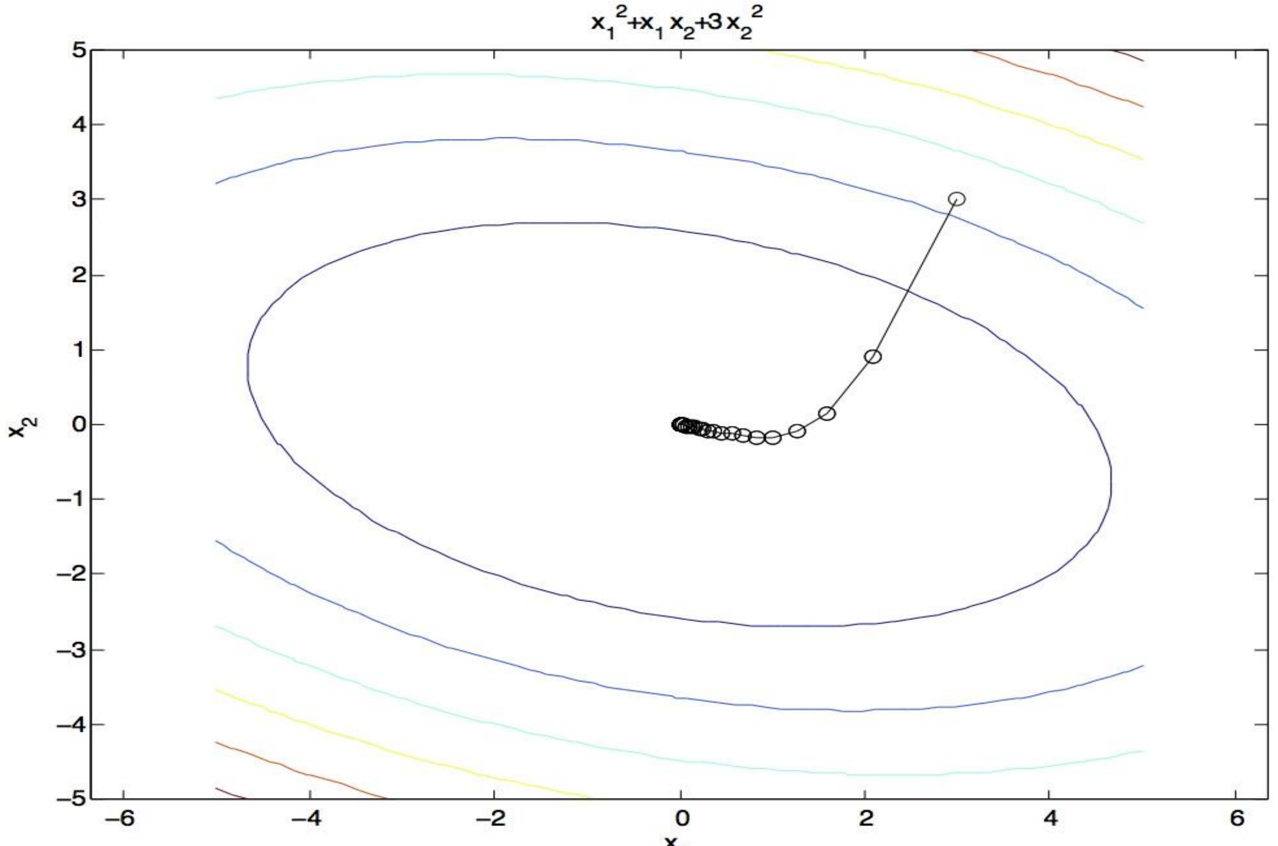


Figure source: Mathworks

Steepest Direction

- Steepest Direction = direction of the gradient

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \cdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

How to Learn?

$$\ell\ell(w) = \sum_i \log P_w(y_i|x_i)$$

$$= \sum_i w_{y_i} \cdot x_i - \log \sum_y e^{w_y \cdot x_i}$$

$$\frac{d}{dw_y} \log P_w(y_i|x_i) = \begin{cases} x_i - x_i \frac{e^{w_y \cdot x_i}}{\sum_{y'} e^{w_{y'} \cdot x_i}} & \text{if } y = y_i \\ -x_i \frac{e^{w_y \cdot x_i}}{\sum_{y'} e^{w_{y'} \cdot x_i}} & \text{otherwise} \end{cases}$$

$$= x_i(I(y = y_i) - P(y|x_i))$$

Optimization Procedure: Gradient Descent

initialize w (e.g., randomly)

repeat for K iterations:

for each example (x_i, y_i):

compute gradient $\Delta_i = -\nabla_w \log P_w(y_i|x_i)$

compute gradient $\nabla_w \mathcal{L} = \sum_i \Delta_i$

$w \leftarrow w - \alpha \nabla_w \mathcal{L}$

$$\frac{d}{dw_y} \log P_w(y_i|x_i) = x_i(I(y = y_i) - P(y|x_i))$$

- α: learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update should change *w* by about 0.1 – 1 %

Stochastic Gradient Descent

initialize w (e.g., randomly)

repeat for K iterations:

  for each example (x_i, y_i):

    compute gradient $\Delta_i = -\nabla_w \log P_w(y_i|x_i)$

    $w \leftarrow w - \alpha \Delta_i$

$$\frac{d}{dw_y} \log P_w(y_i|x_i) = x_i(I(y = y_i) - P(y|x_i))$$

if $y_i = y$, move w_y toward x_i

with weight $\underbrace{1 - P(y_i|x_i)}_{\text{probability of \textit{incorrect} answer}}$

if $y_i \neq y$, move w_y away from x_i

with weight $\underbrace{P(y|x_i)}_{\text{probability of \textit{incorrect} answer}}$

compare this to the multiclass perceptron: probabilistic weighting!

Logistic Regression Demo!

https://playground.tensorflow.org/