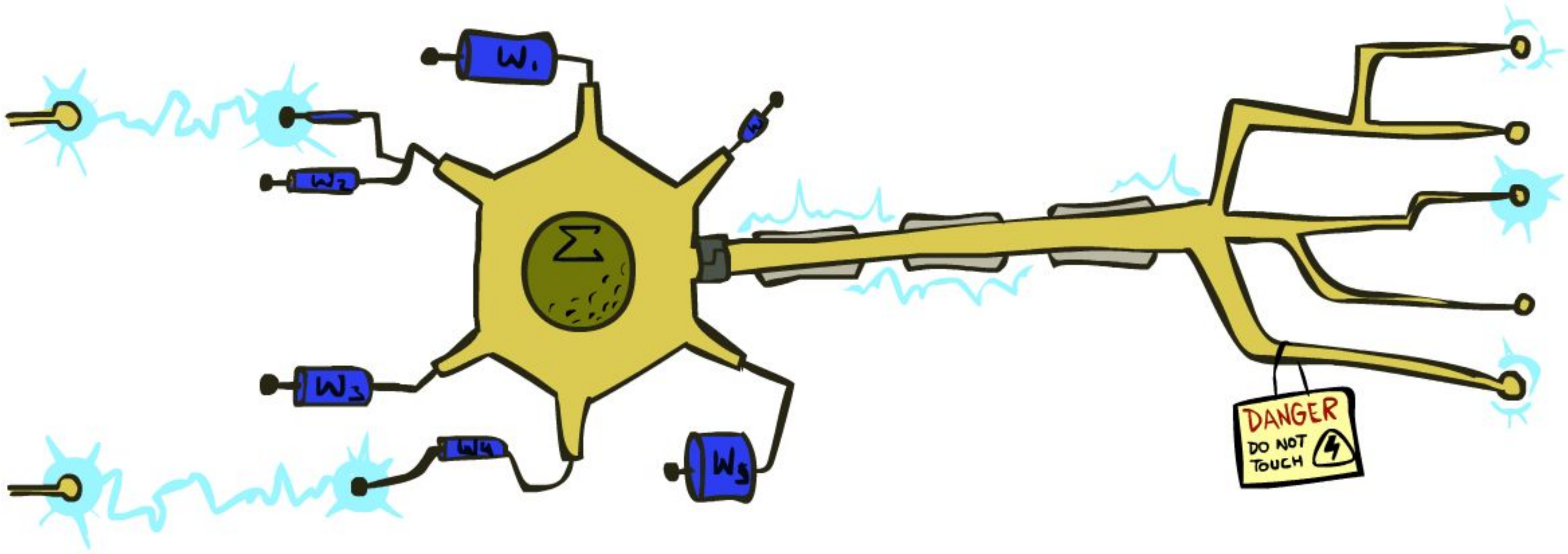
CS 4700: Foundations of Artificial Intelligence

Perceptrons and Logistic Regression



Instructor: Kevin Ellis

These 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.

Announcements

Homework 5 out today:

Due 4/25

First problem is based on today’s lecture

Machine Learning

- Most common approaches:
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

Machine Learning

- Most common approaches:
  - Supervised learning (CS 4780)
  - Unsupervised learning (CS 4786)
  - Reinforcement learning (CS4789)

Machine Learning

- Most common approaches:
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning ✓

Machine Learning

- Most common approaches:
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning ✓ - Chapter 22

Machine Learning

- Most common approaches:
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning ✓ - Chapter 22 (and Chapter 17)

Machine Learning

- Most common approaches:
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning ✓ - Chapter 22 (and Chapter 17)

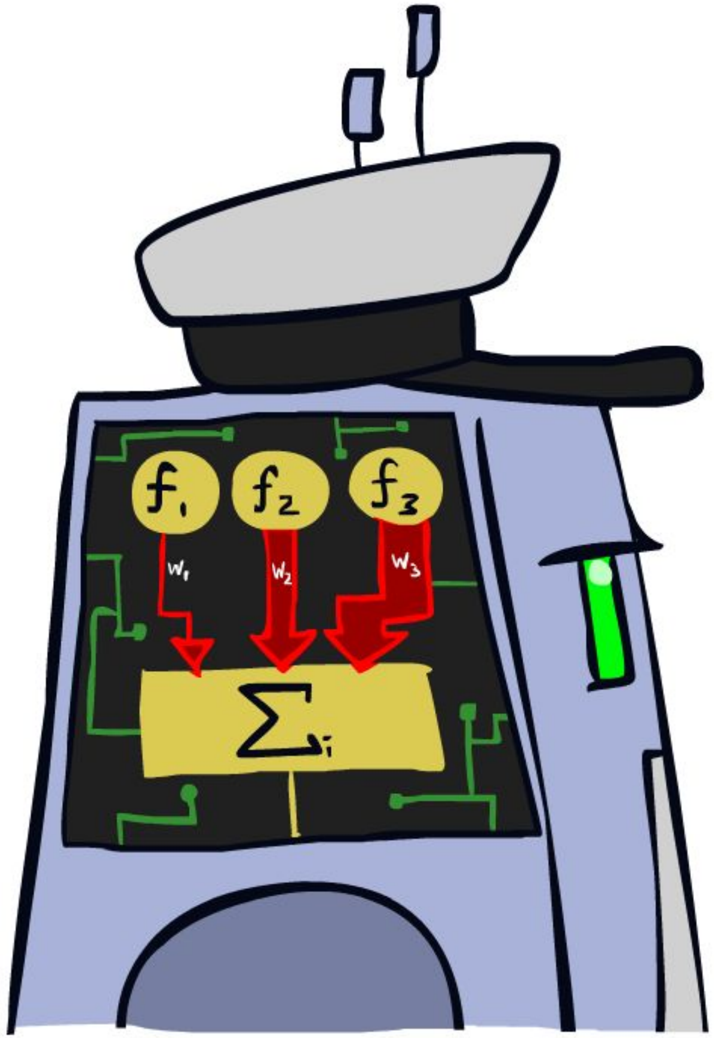
Machine Learning

- Most common approaches:
  - Supervised learning (“Learning from examples”)
  - Unsupervised learning
  - Reinforcement learning ✓ - Chapter 22 (and Chapter 17)

Machine Learning

- Most common approaches:
  - Supervised learning – Chapter 19
  - Unsupervised learning
  - Reinforcement learning ✓ - Chapter 22 (and Chapter 17)

Linear Classifiers



Feature Vectors

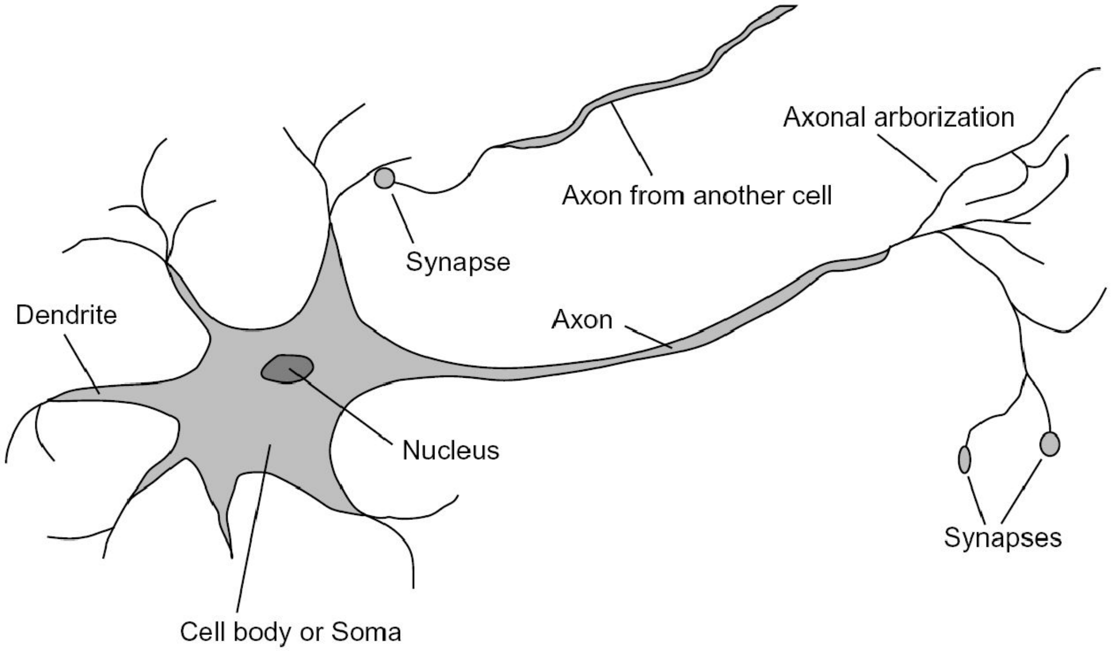
x	$f(x)$	y

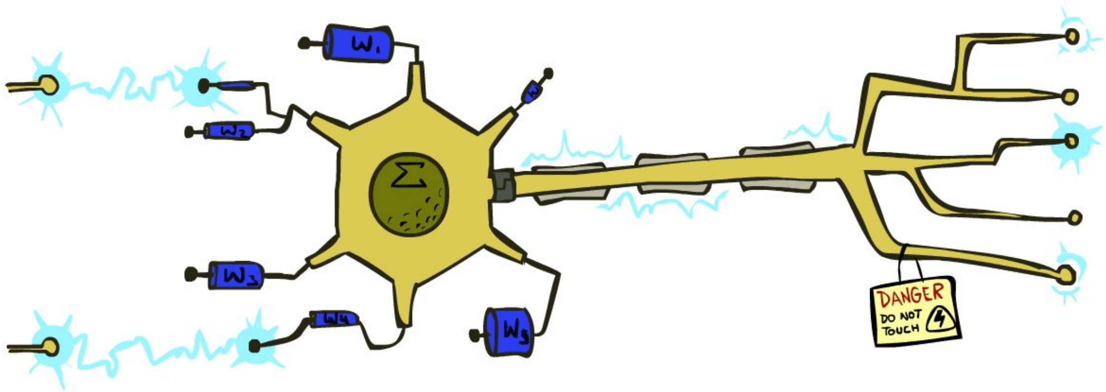
Hello,
 	
Do you want free printr cartriges? Why pay more when you can get them ABSOLUTELY FREE! Just
	`# free : 2`
	`YOUR_NAME : 0`
	`MISSPELLED : 2`
	`FROM_FRIEND : 0`
	`...`
		SPAM
		or
		+

	`PIXEL-7,12 : 1`
	`PIXEL-7,13 : 0`
	`...`
	`NUM_LOOPS : 1`
	`...`
		“2”

Some (Simplified) Biology

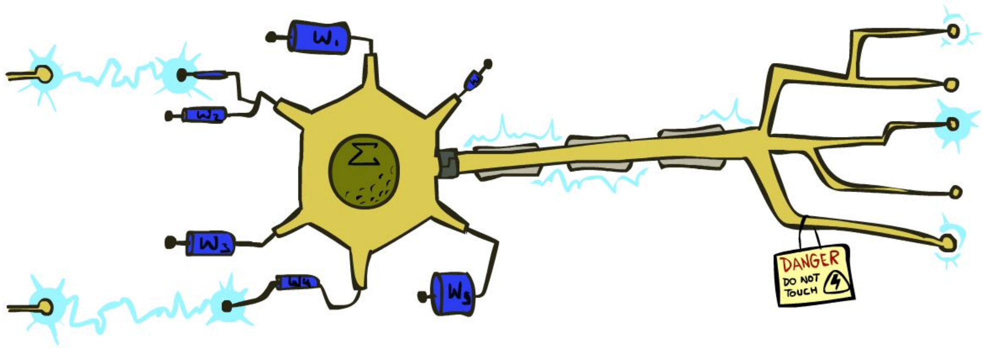
- Very loose inspiration: human neurons





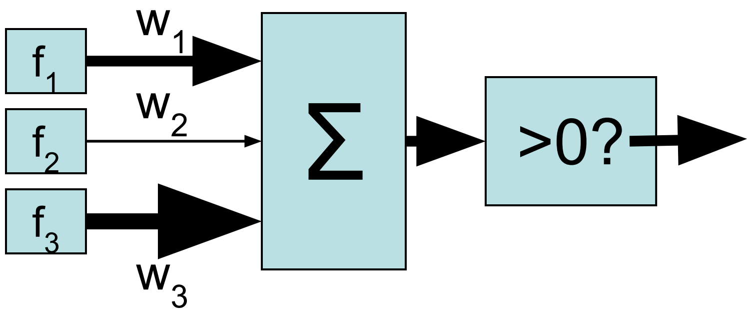
Linear Classifiers

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



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

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



Linear Classifiers: Bias term

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

Activation: Linear function
w: Slope of linear function
bias: intercept of linear function

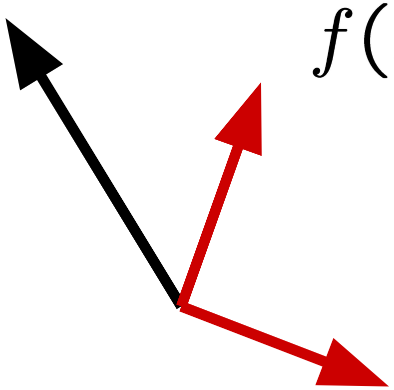
Trick: Add an extra feature which is always 1. Weight on that feature = bias.
Algorithms ignore the concept of “bias” and only have to think about weight w

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
...
```

w



$f(x_1)$

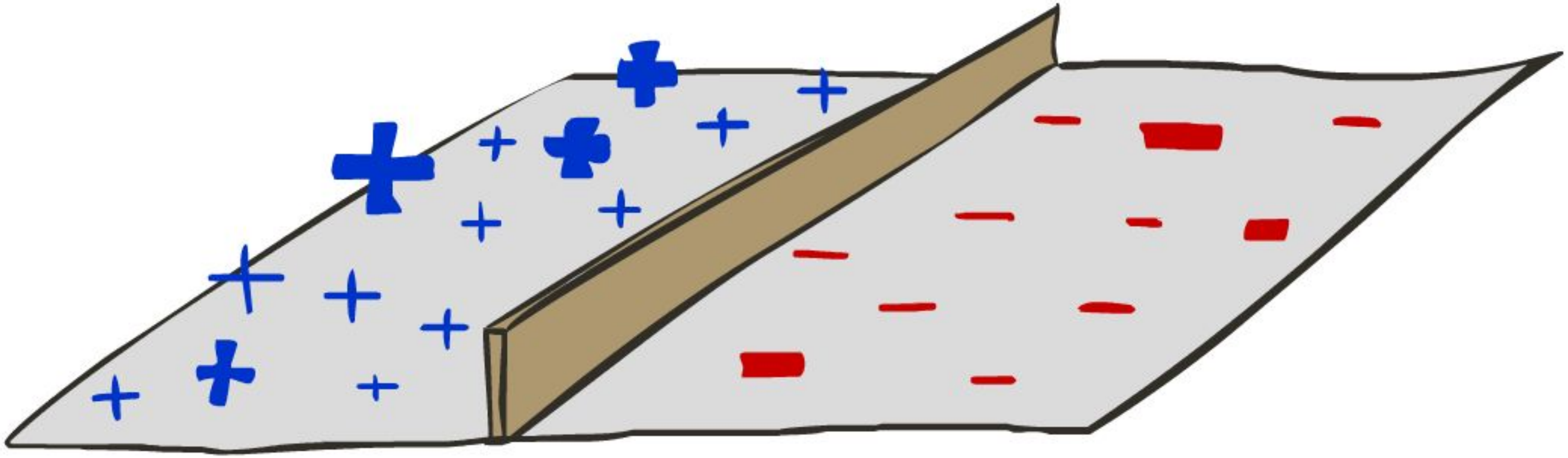
```
# free      : 2
YOUR_NAME   : 0
MISSPELLED  : 2
FROM_FRIEND : 0
...
```

$f(x_2)$

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

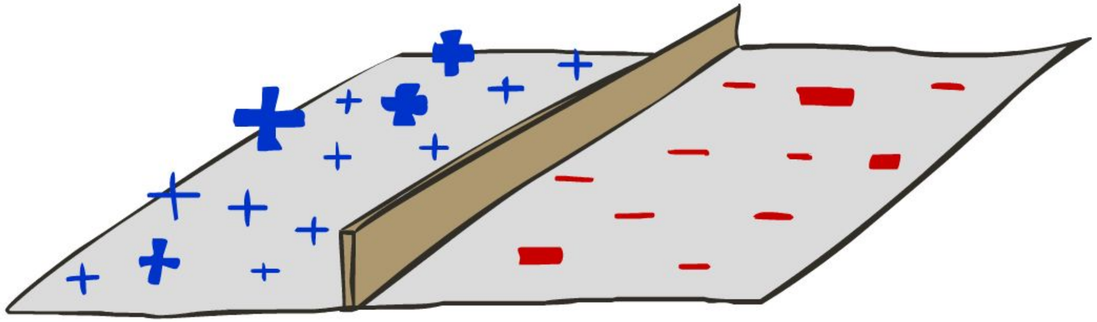
Dot product $w \cdot f$ positive means the positive class

Decision Rules



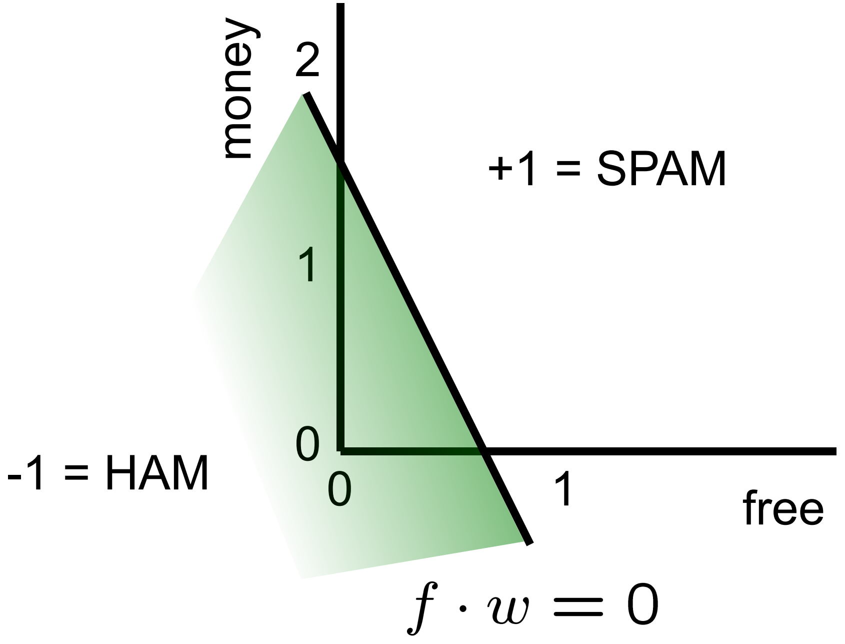
Binary Decision Rule

- 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



$$w$$

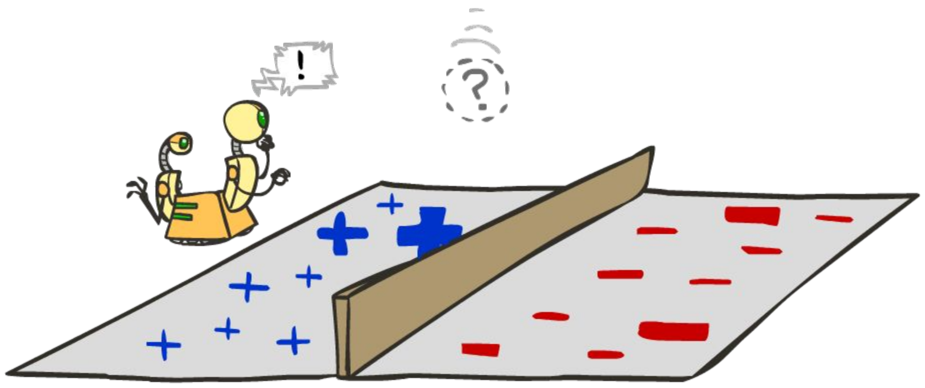
```
BIAS  : -3
free  :  4
money :  2
...
```



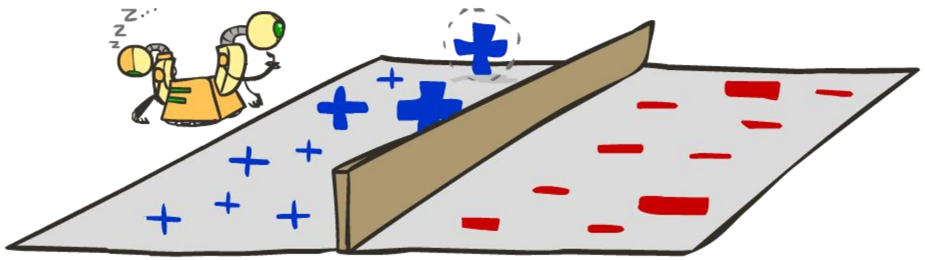
Weight Updates

Learning: Binary Perceptron

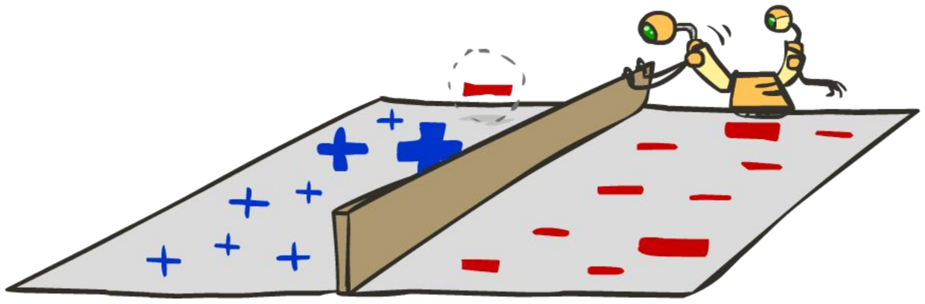
- Start with weights = 0
- For each training instance:
  - Classify with current weights



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



  - If wrong: adjust the weight vector



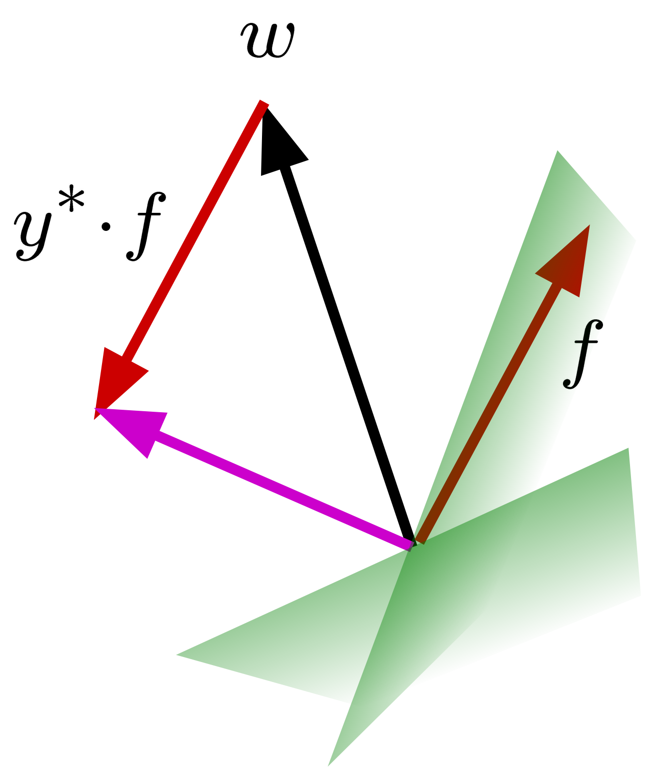
Learning: Binary Perceptron

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

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

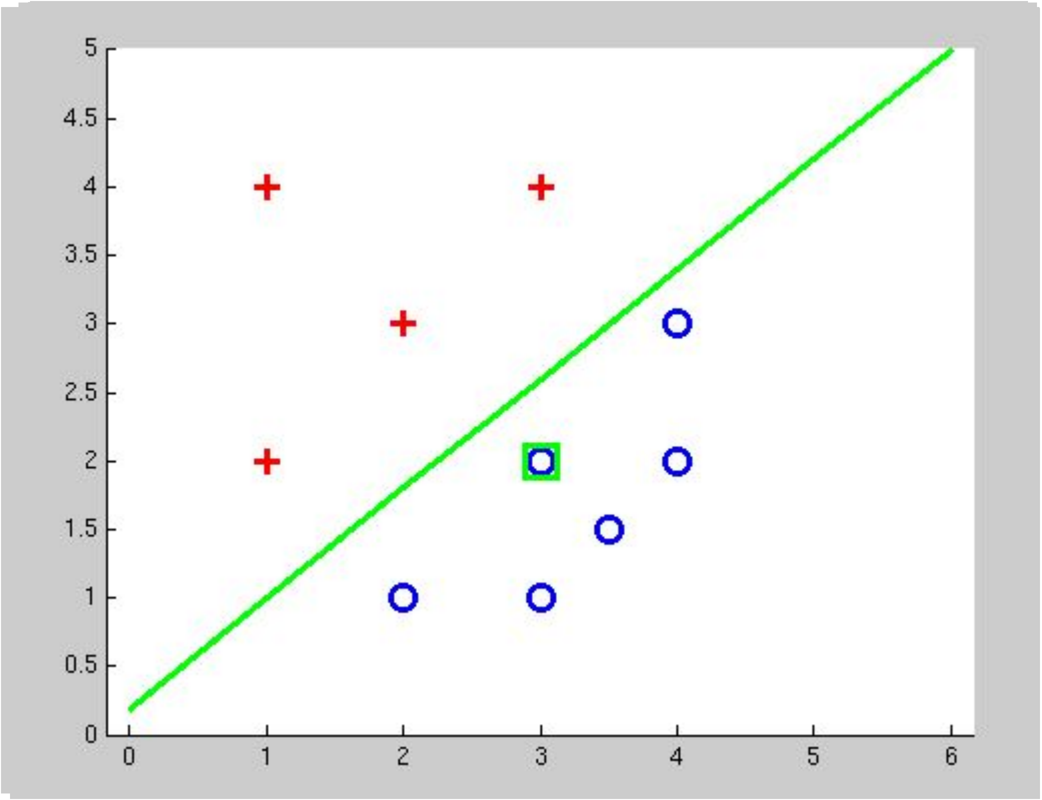
  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

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



Examples: Perceptron

- Separable Case



Multiclass Decision Rule

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

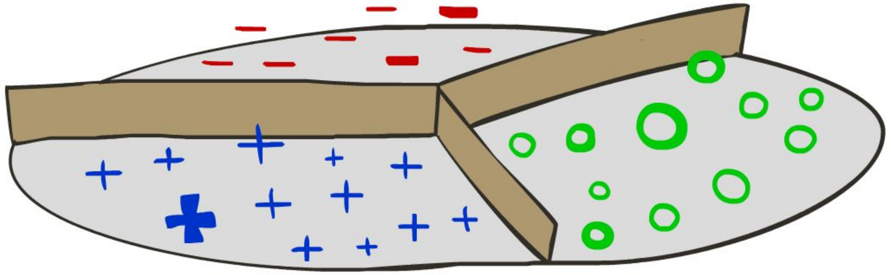
$$w_y$$

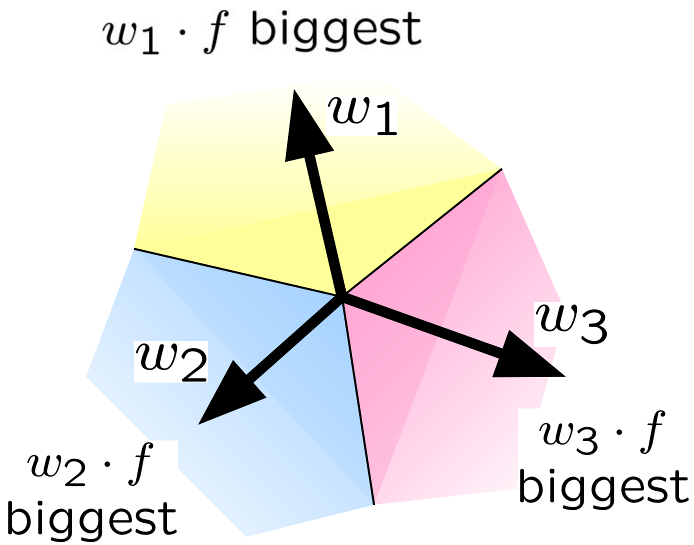
  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - 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

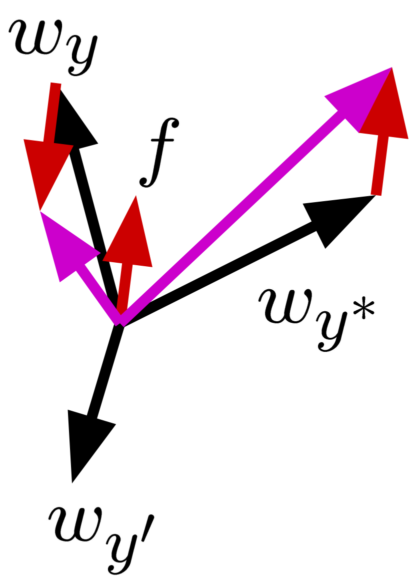
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

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

- 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}

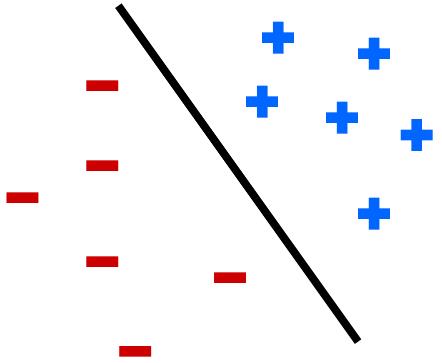
```
BIAS  : 0
win   : 0
game  : 0
vote  : 0
the   : 0
...
```

Properties of Perceptrons

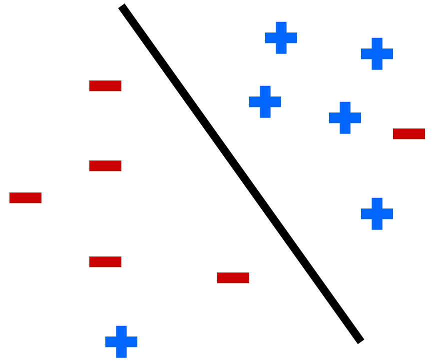
- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

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

Separable

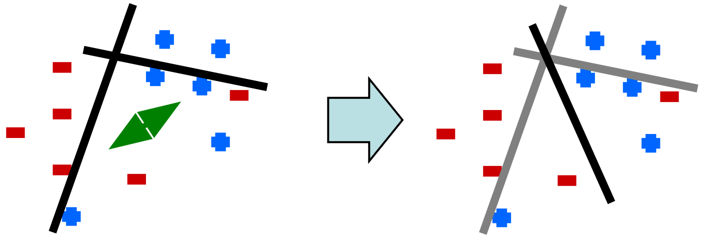


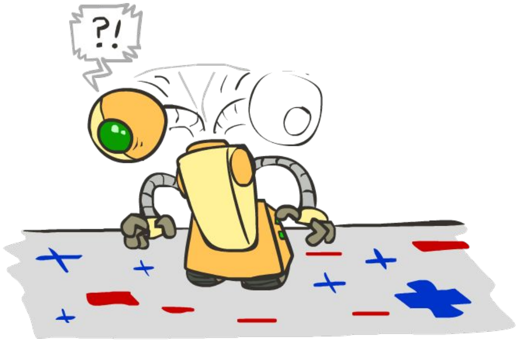
Non-Separable



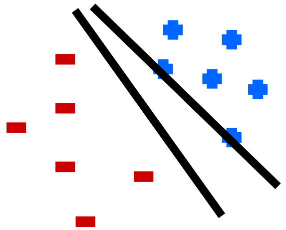
Problems with the Perceptron

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

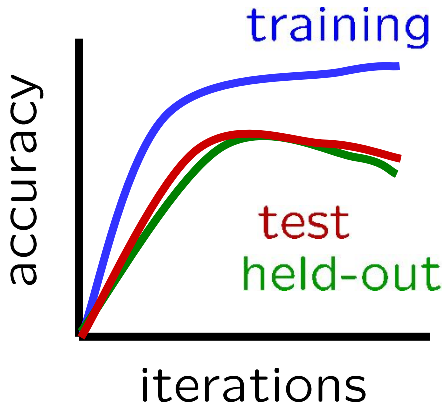


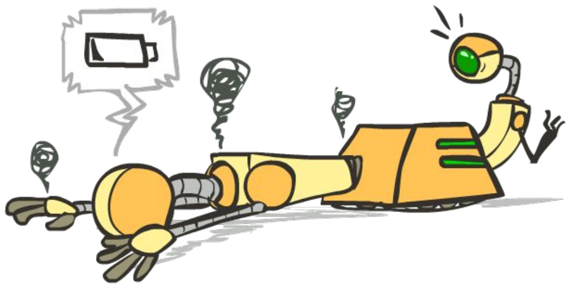


- Mediocre generalization:
  - finds a "barely" separating solution

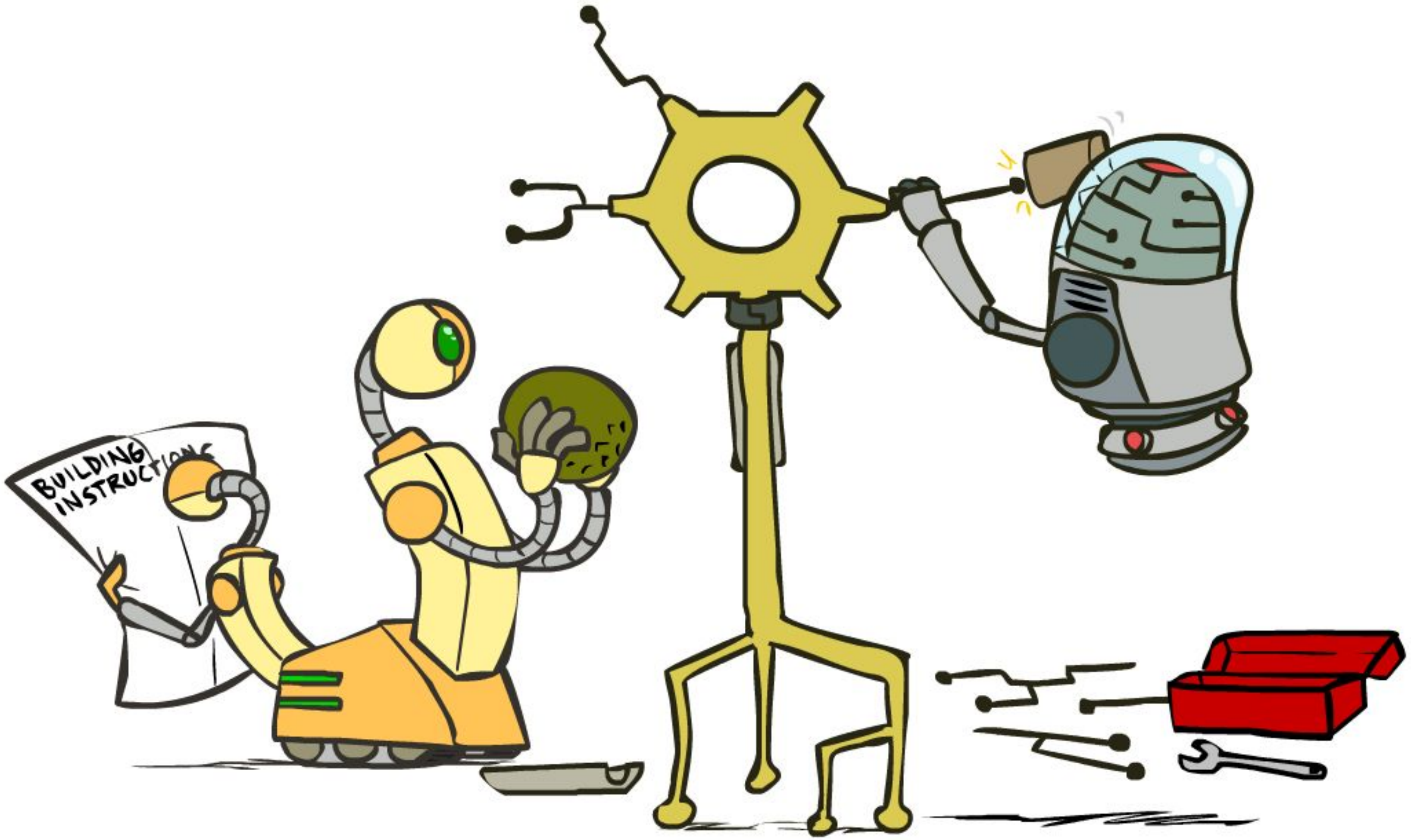


- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



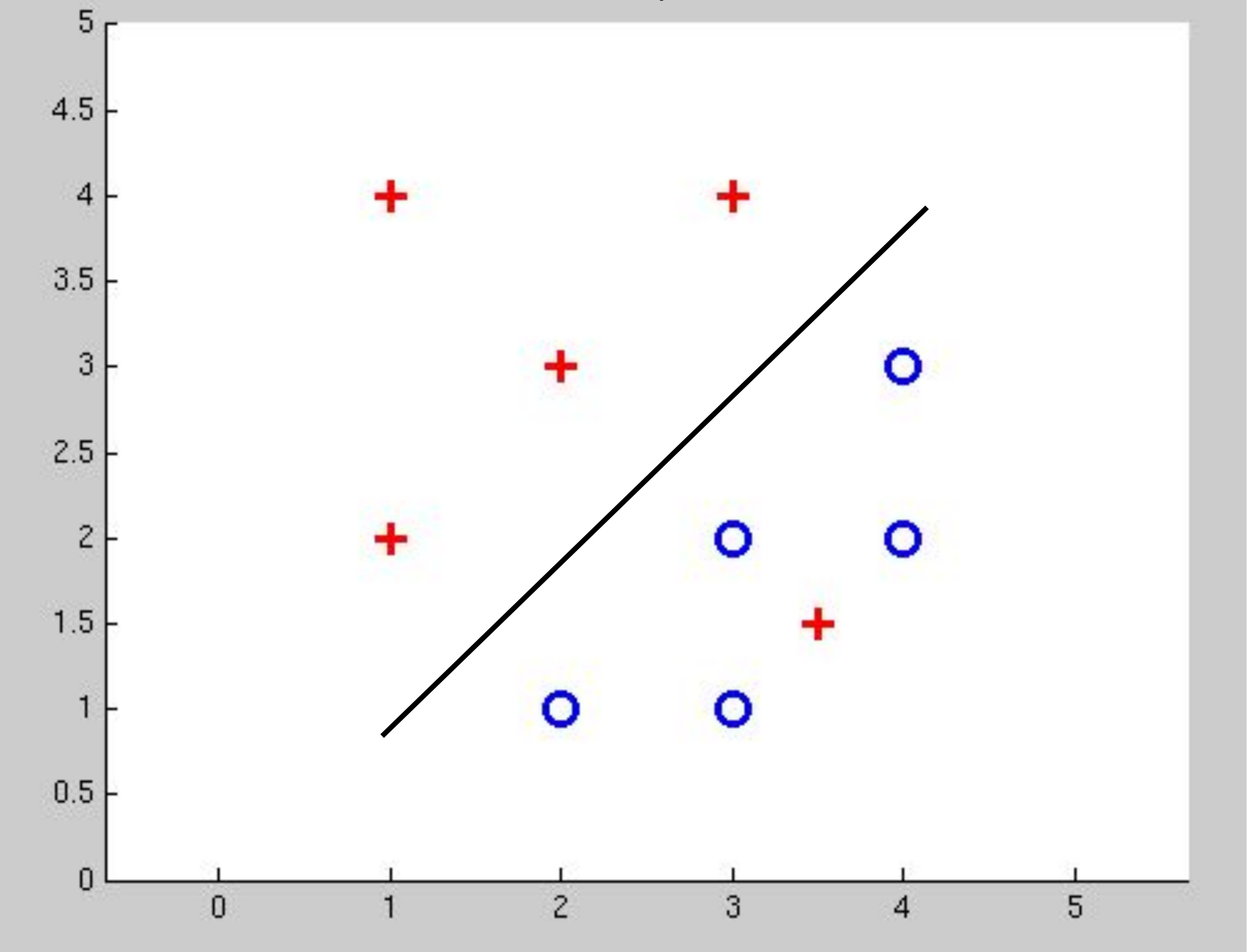


Improving the Perceptron

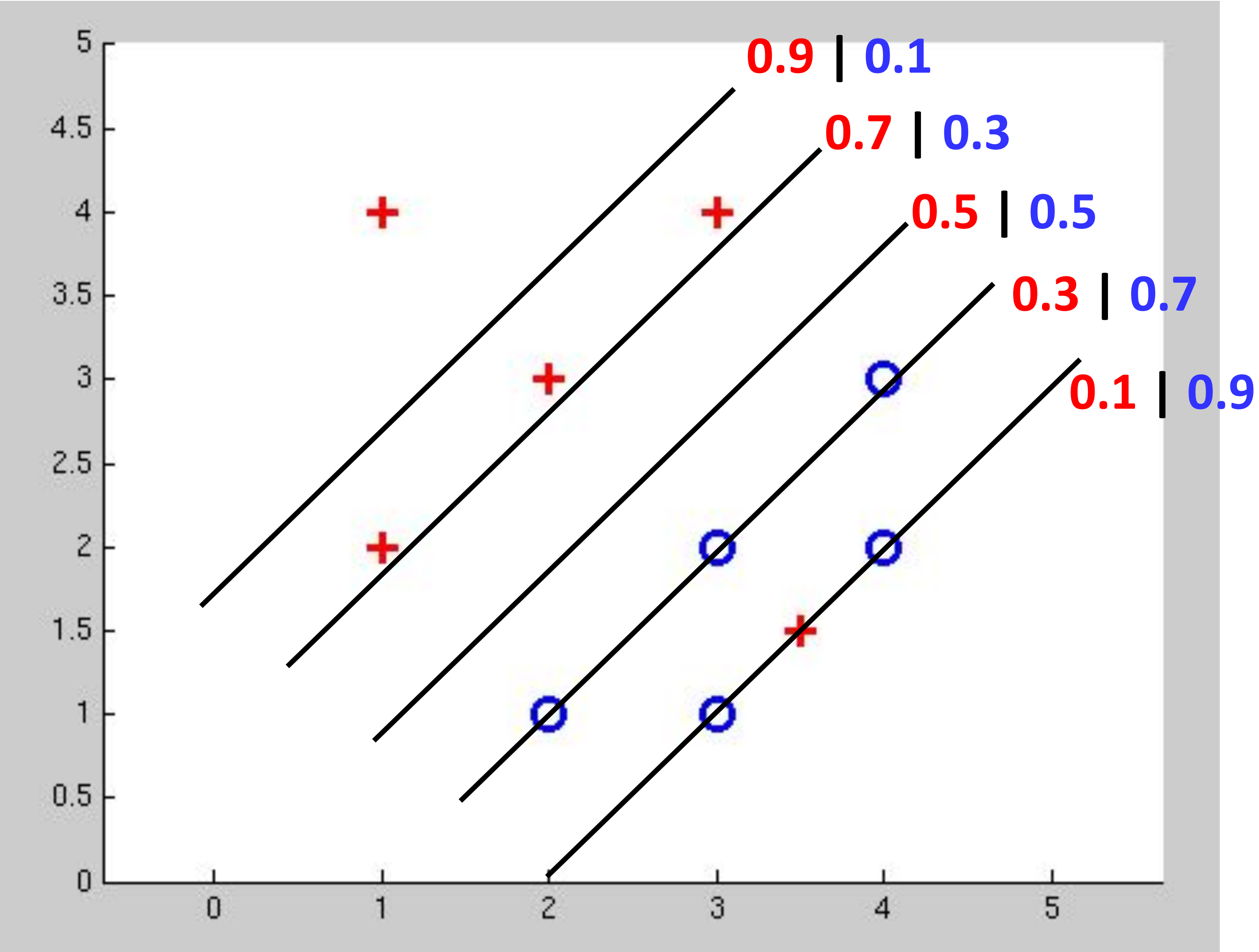


Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake



Non-Separable Case: Probabilistic Decision

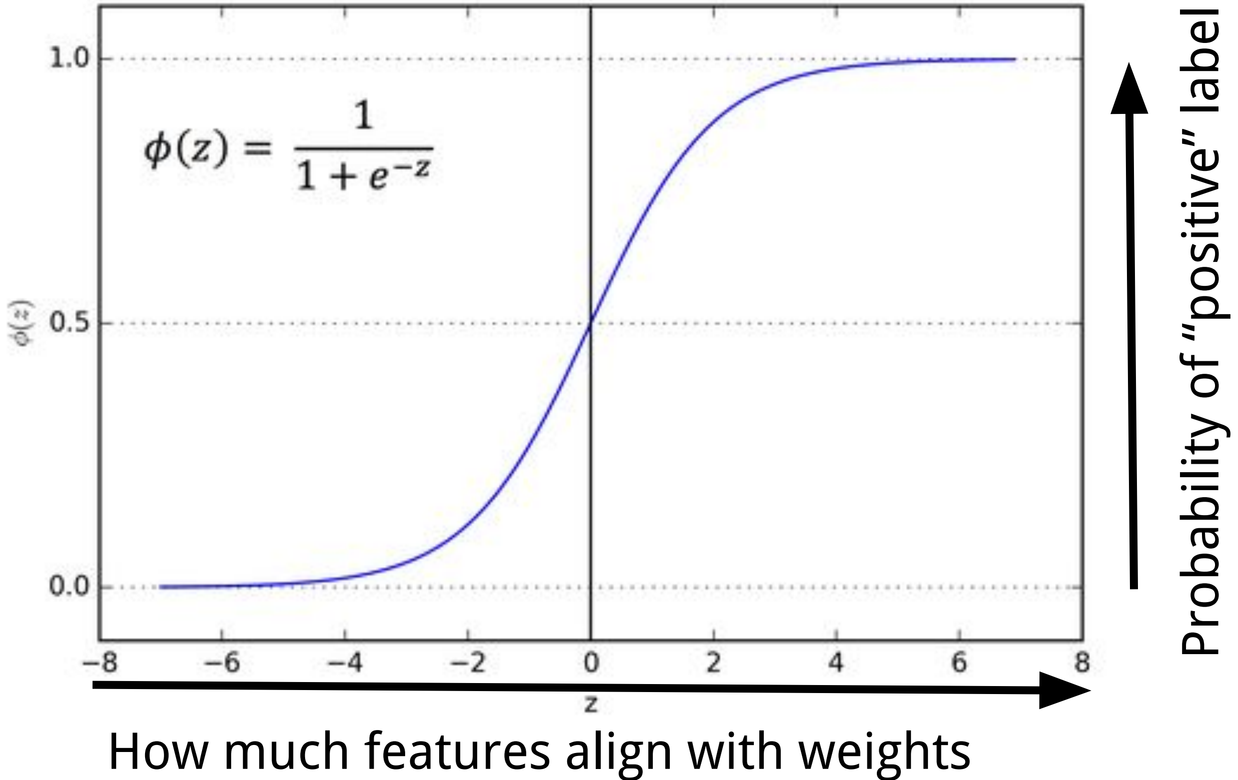


How to get probabilistic decisions?

- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive $\rightarrow$ want probability going to 1
- If $z = w \cdot f(x)$ very negative $\rightarrow$want probability going to 0

- Sigmoid function

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



Best w?

- Maximum likelihood estimation:

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

with:

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

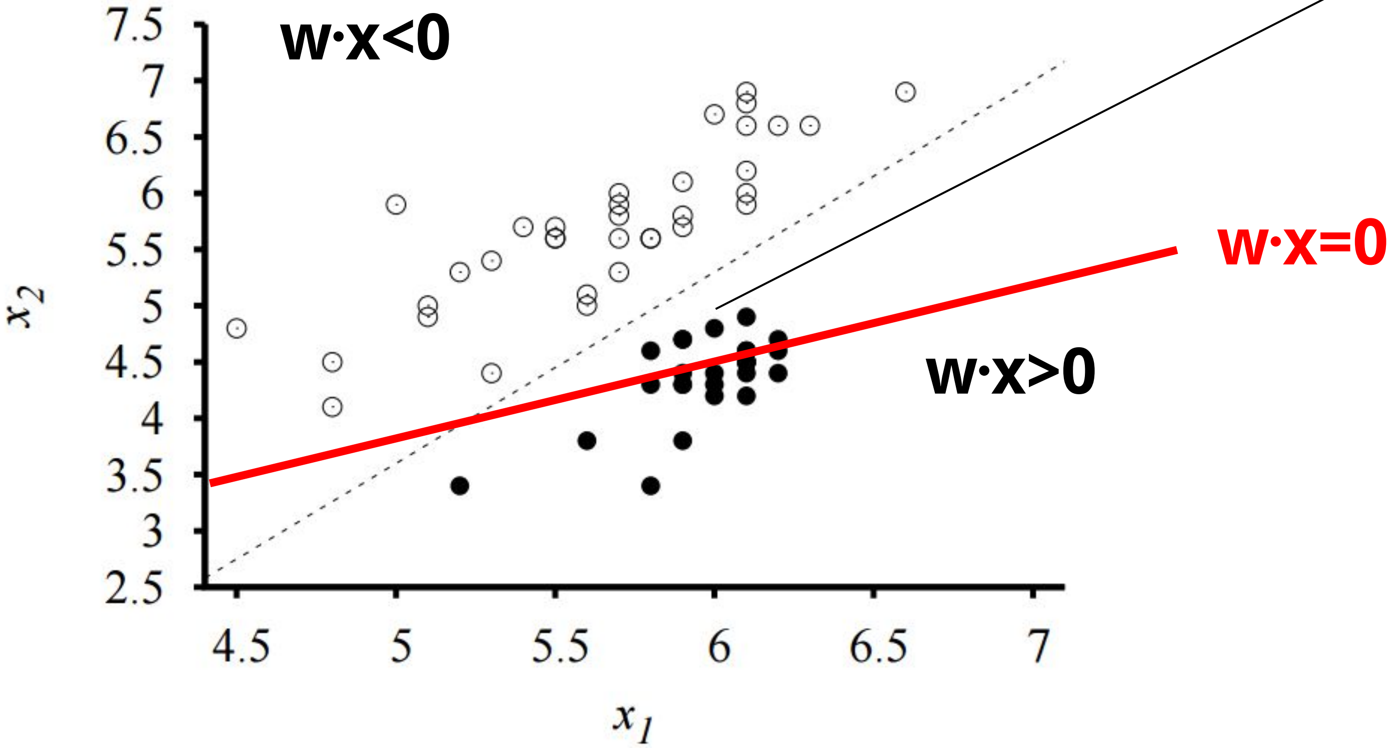
$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

= Logistic Regression

Where does this come from?

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

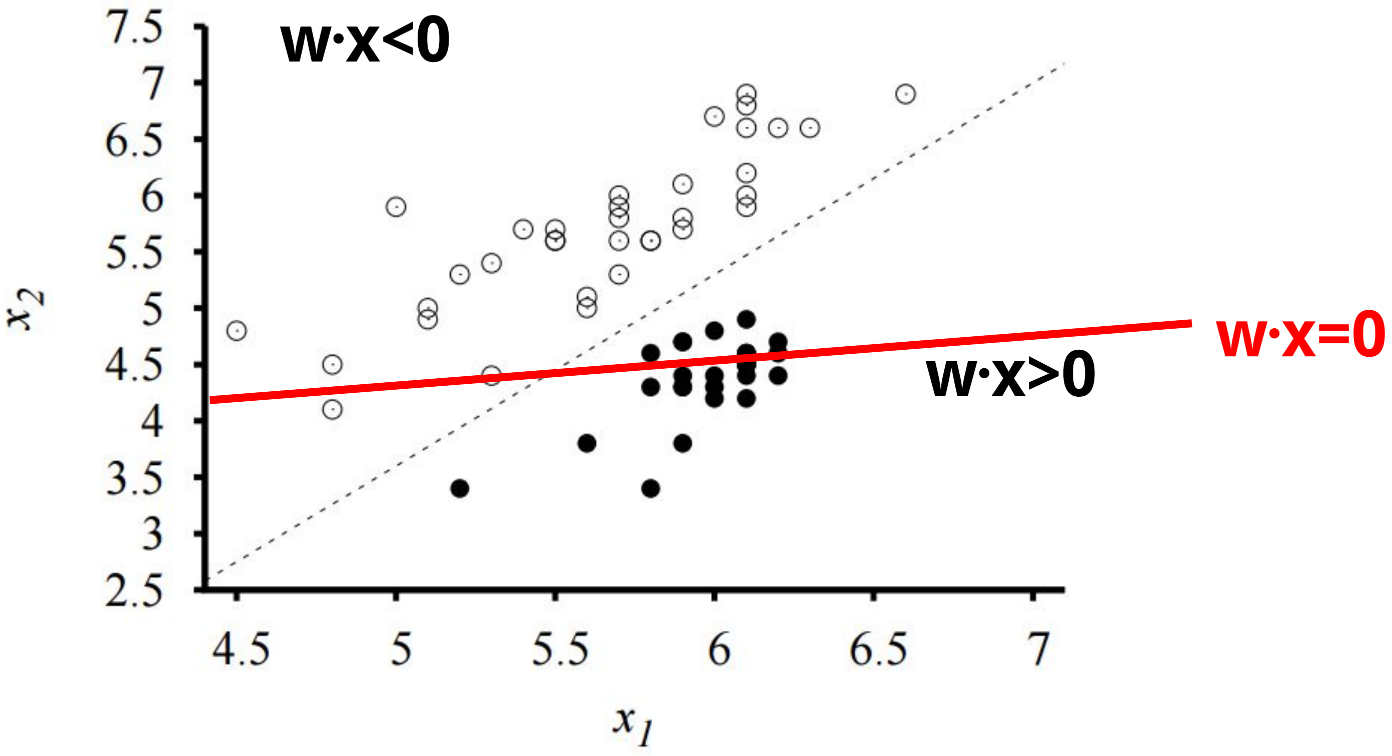
Classification - What are we maximizing?



These misclassified points decrease what we are maximizing

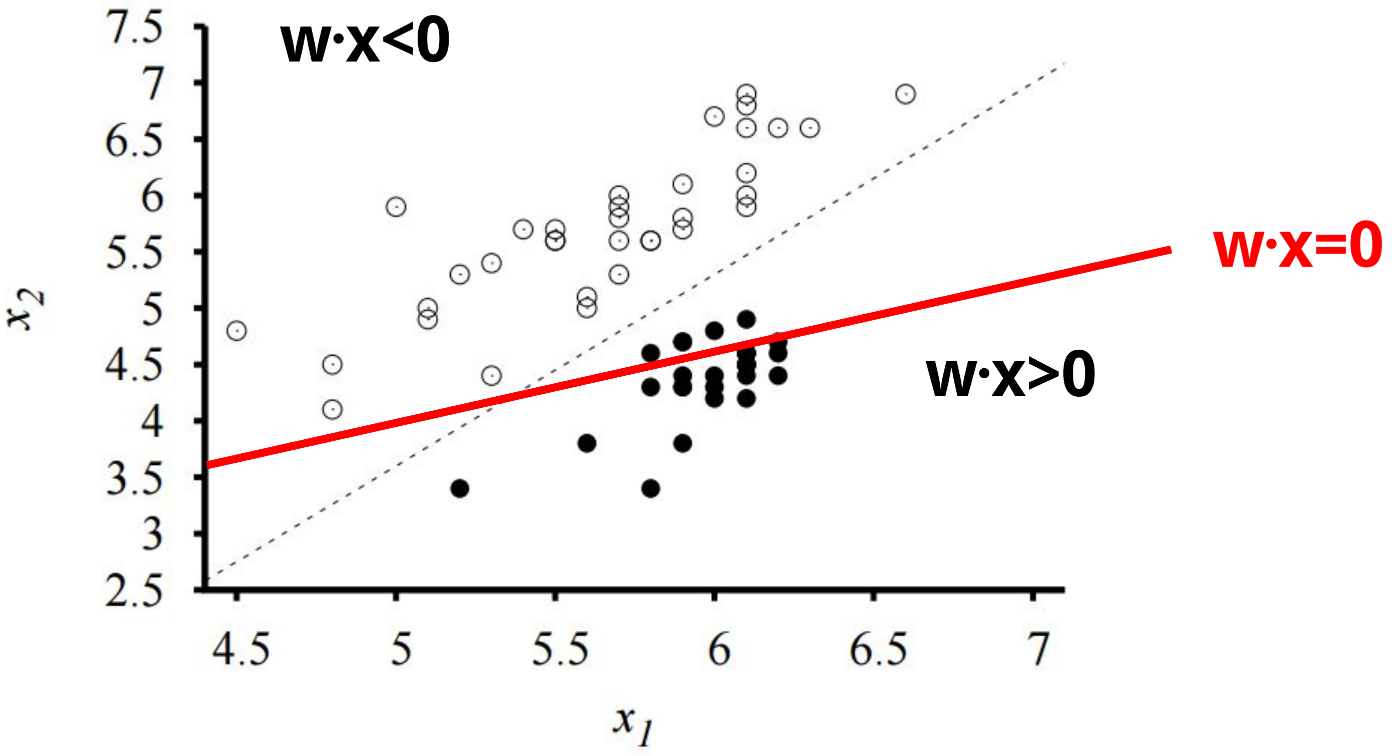
Classification - What are we maximizing?

Yikes, even worse!

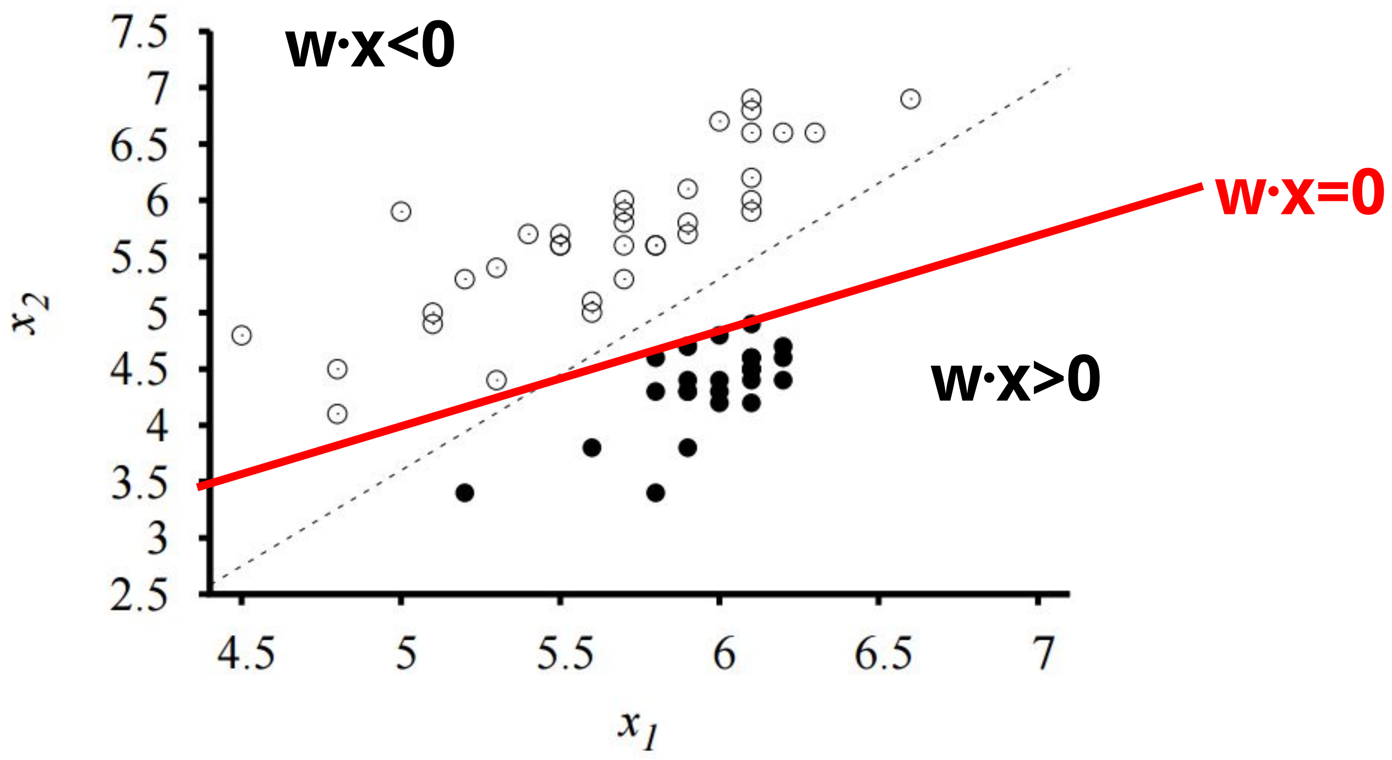


Classification - What are we maximizing?

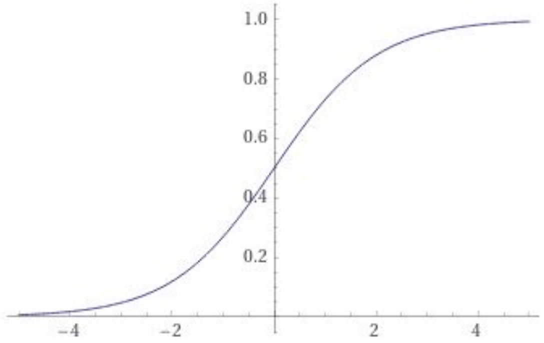
A bit better



Classification - What are we maximizing?

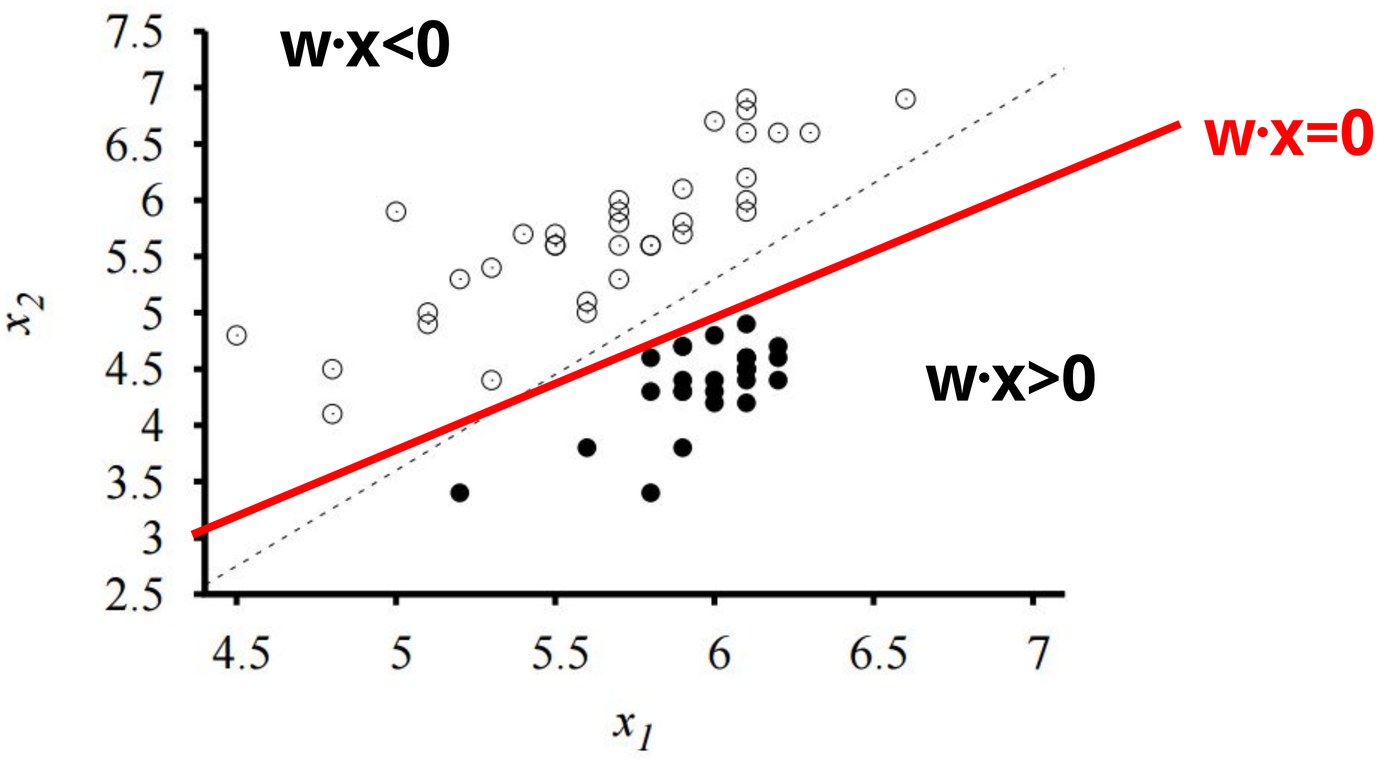


Perfectly classifies, but can it be improved?

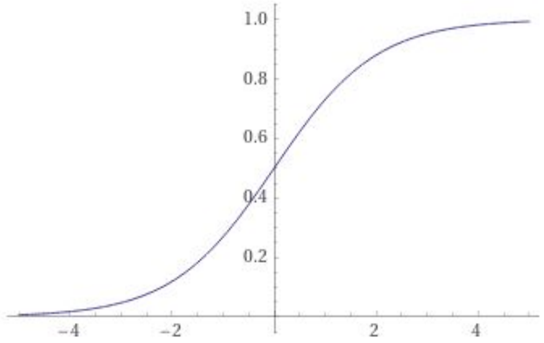


SMOOTH function: Smoothly prefers pushing examples out farther

Classification - What are we maximizing?



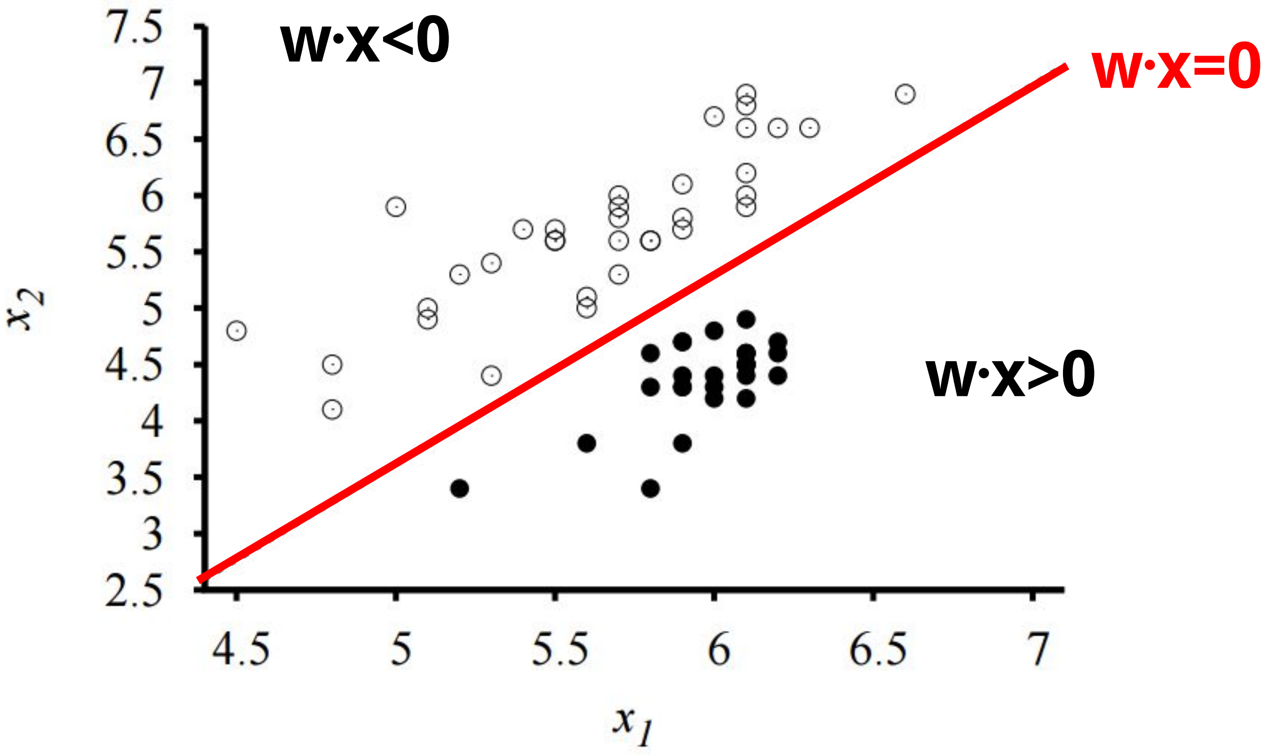
Getting better…

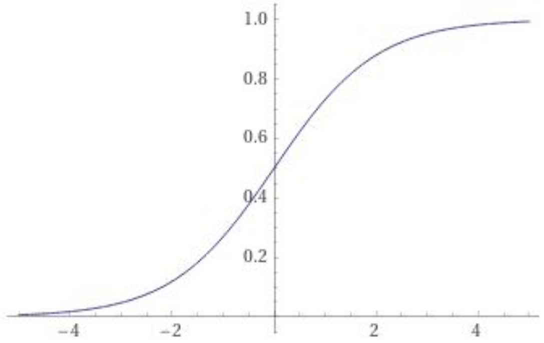


SMOOTH function:
Smoothly prefers pushing examples out farther

Classification - What are we maximizing?

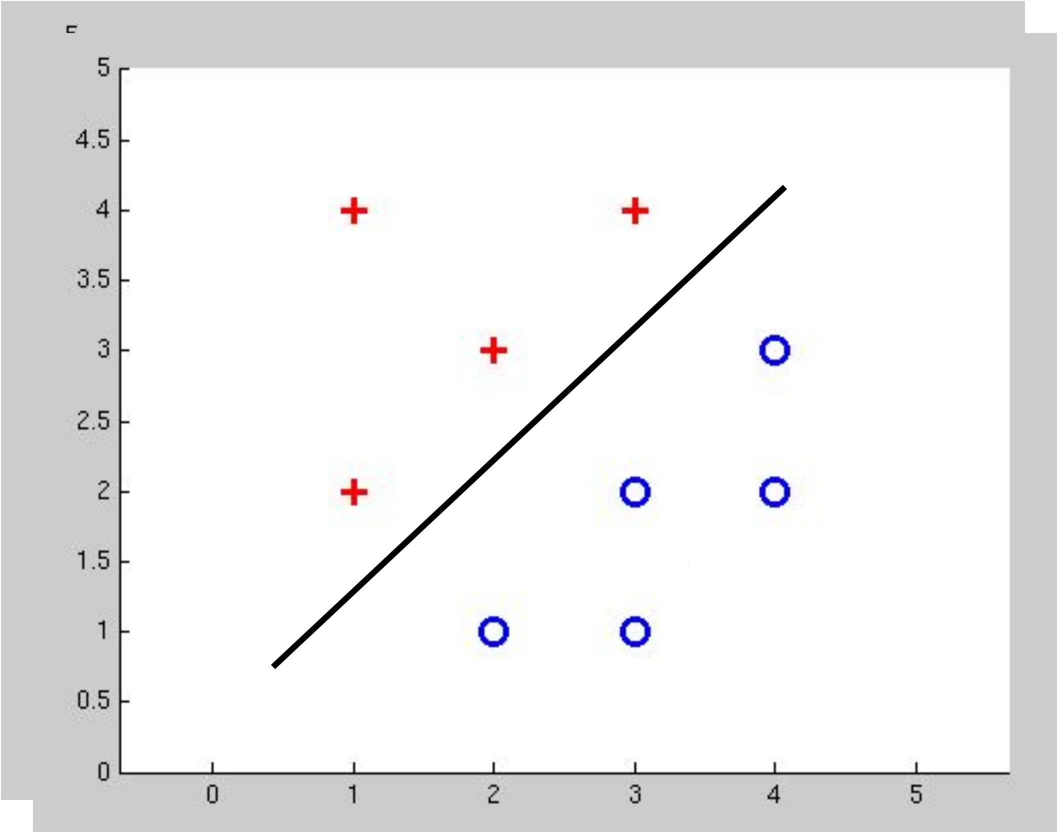
Perfect!

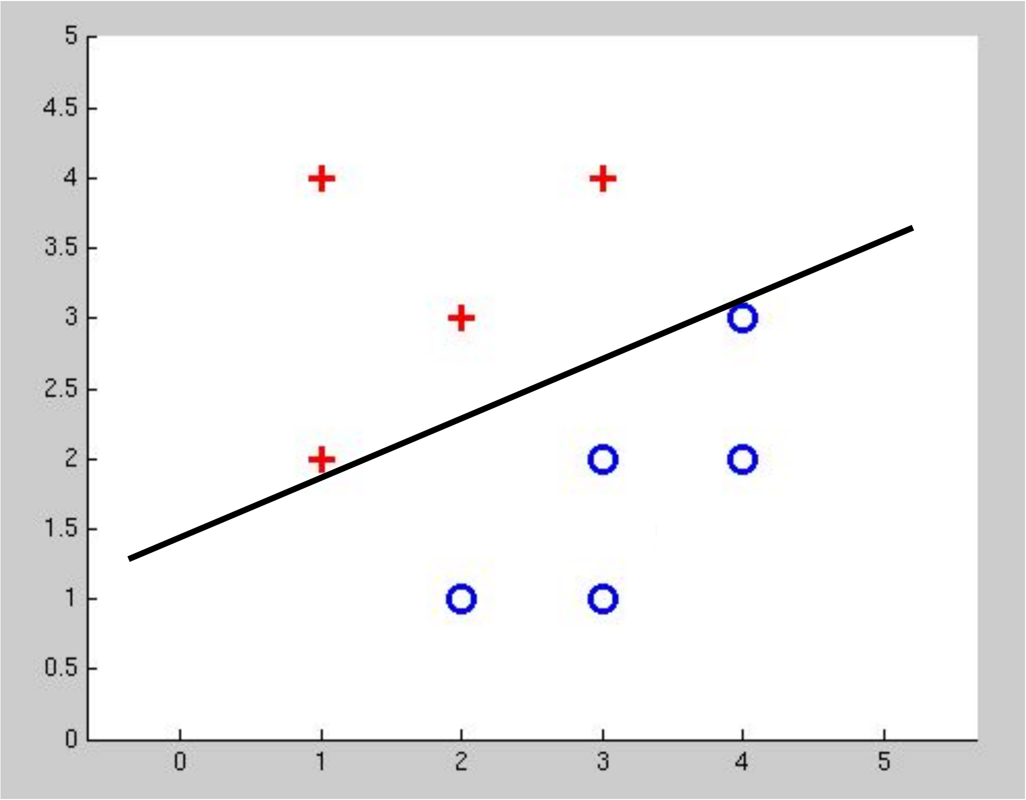




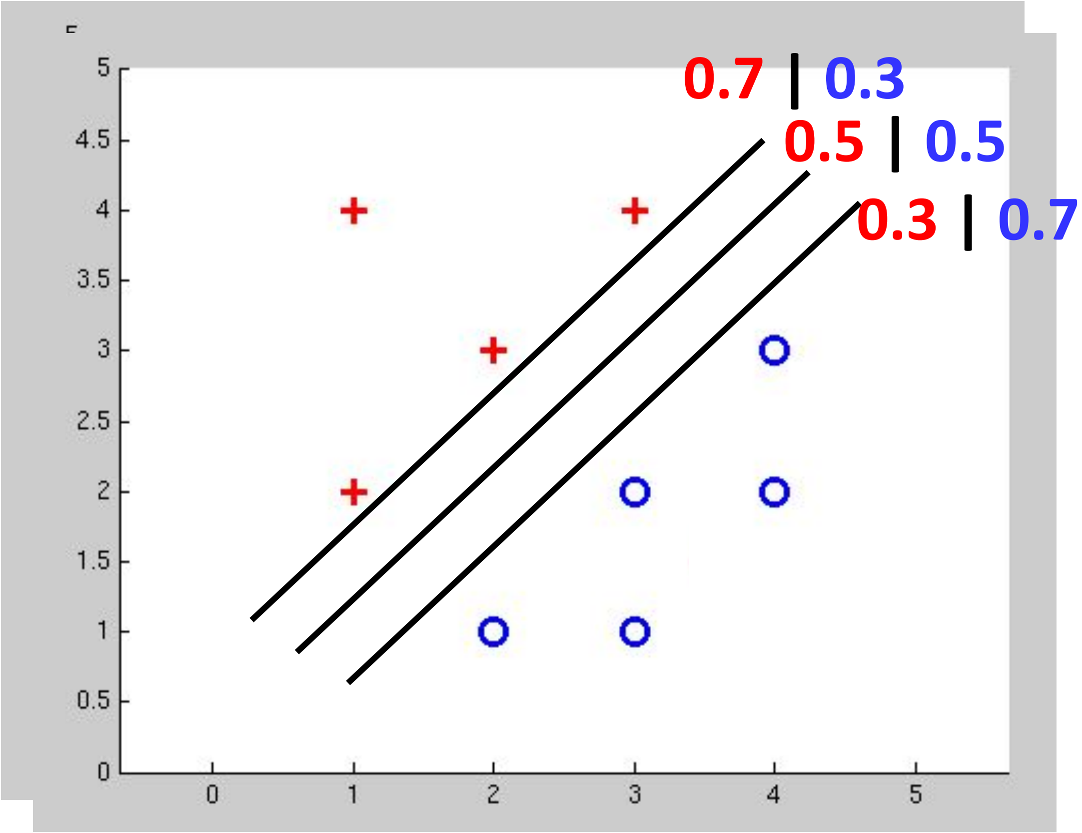
SMOOTH function:
Smoothly prefers pushing examples out farther

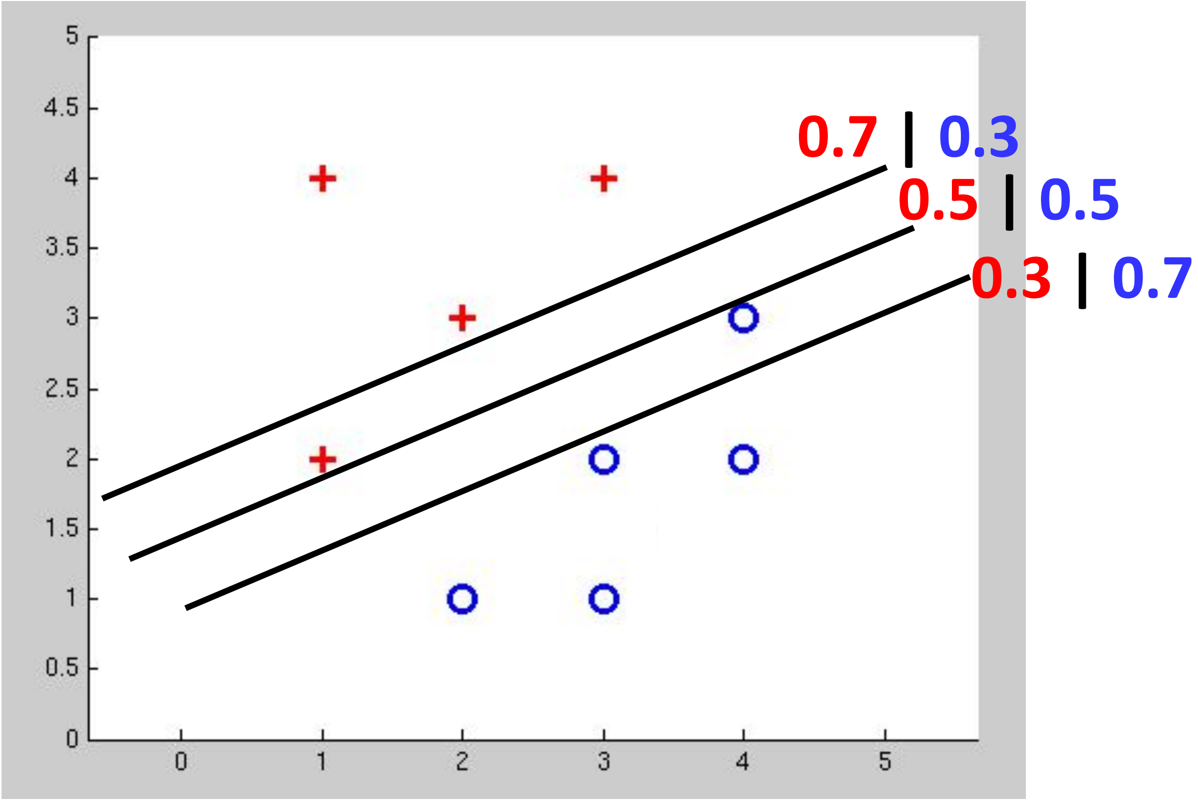
Separable Case: Deterministic Decision – Many Options





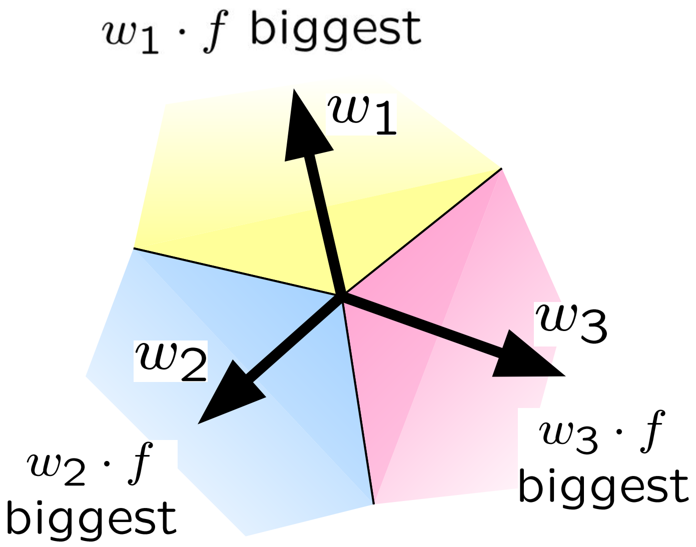
Separable Case: Probabilistic Decision – Clear Preference





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:

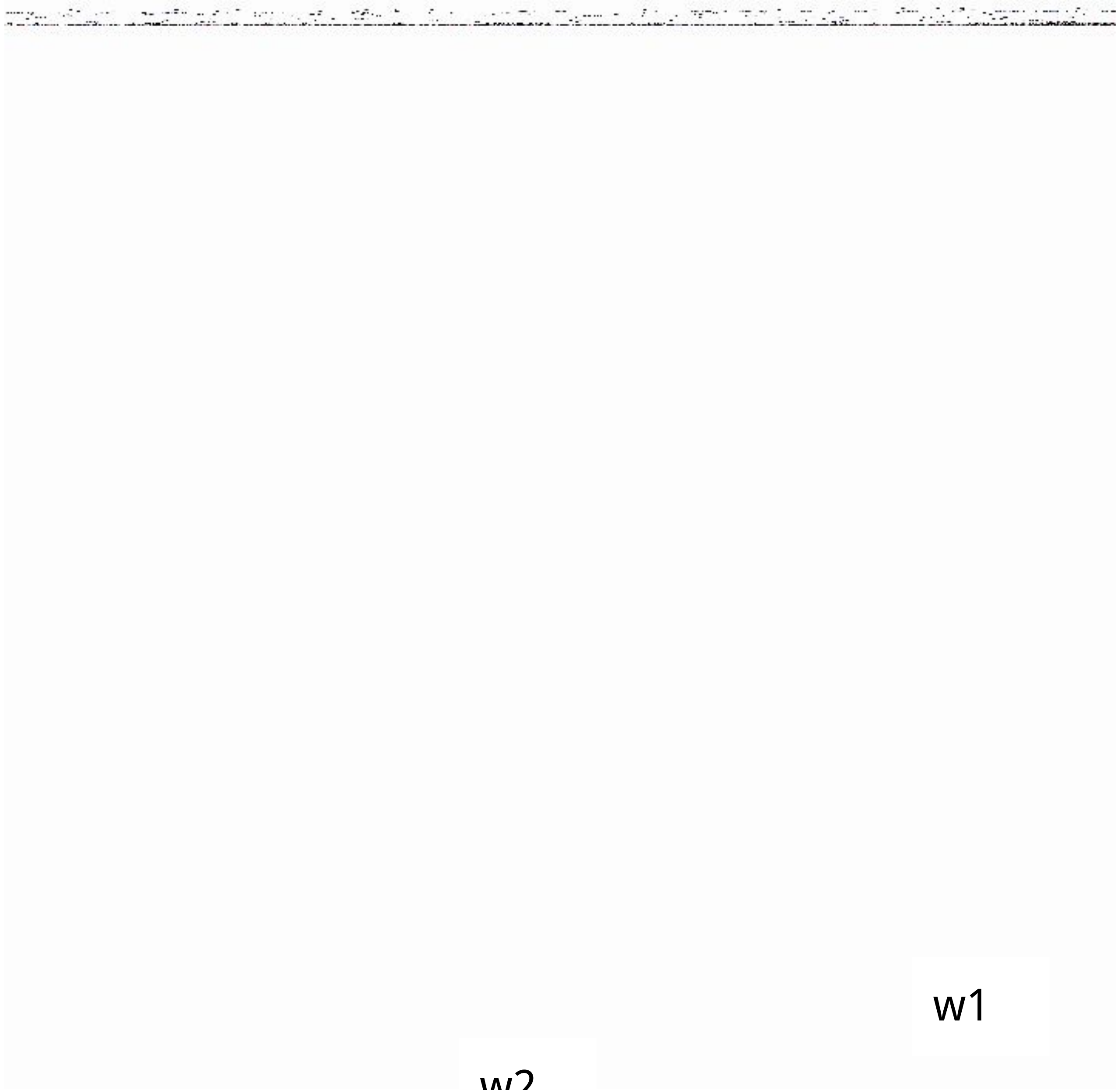
$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

= Multi-Class Logistic Regression

Next Lecture

- Optimization
  - i.e., how do we solve:

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



Source

Source

Reminder

Homework 5 out today:

- Due 4/25
- First problem is based on today's lecture