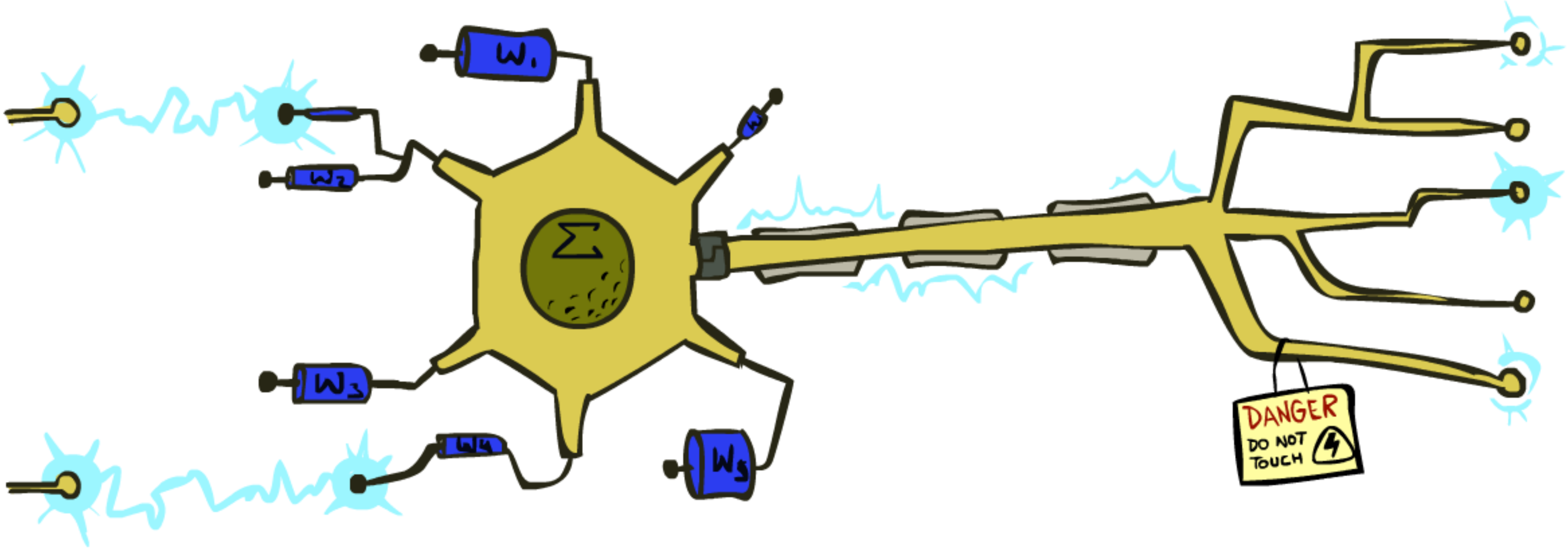
Announcements

- **Homework 8** due **today** (Nov 7) at 11:59pm PT
- **Project 4** extended! Now due **this Friday** (Nov 10) at 11:59pm PT
- **HW 4 part 2** and **HW 5 part 2** regrades at due **this Friday** (Nov 10) at 11:59pm PT

CS 188: Artificial Intelligence

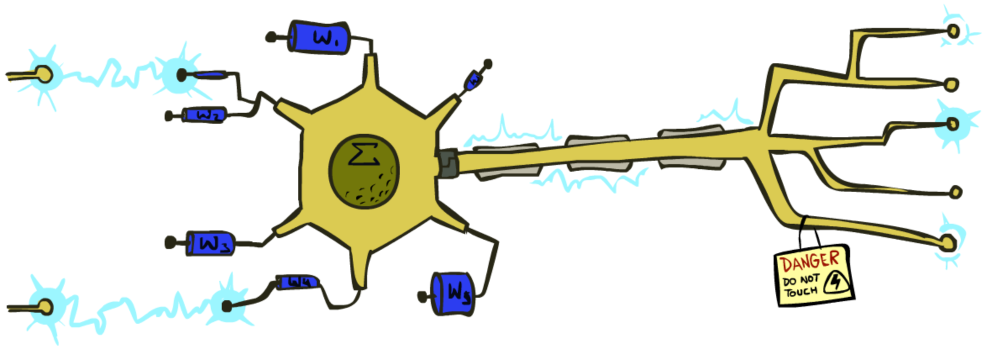
Perceptrons, Logistic Regression and Optimization



[These slides were created by Dan Klein, Pieter Abbeel, Anca Dragan, Sergey Levine. All CS188 materials are at http://ai.berkeley.edu.]

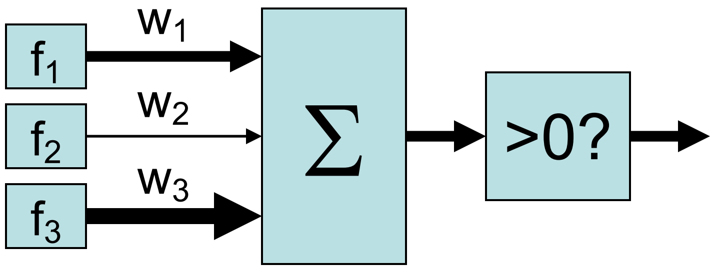
Last Time: Perceptron

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



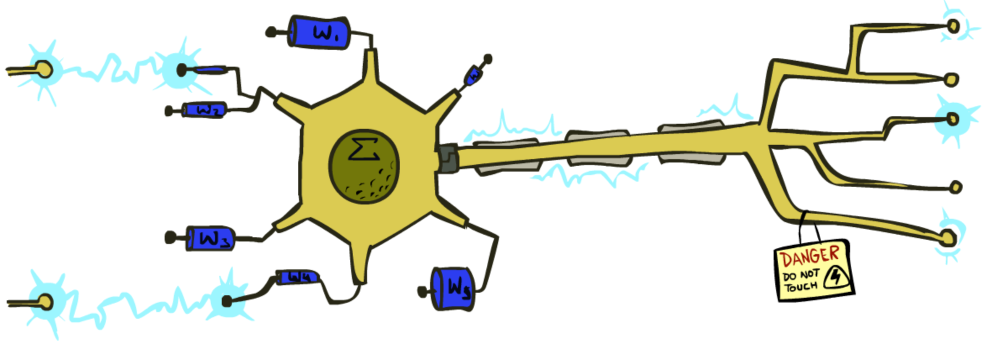
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

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



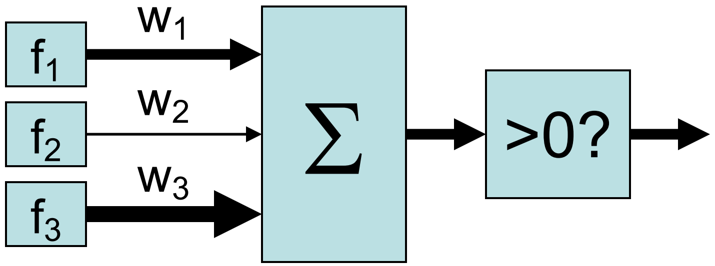
Last Time: Perceptron

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

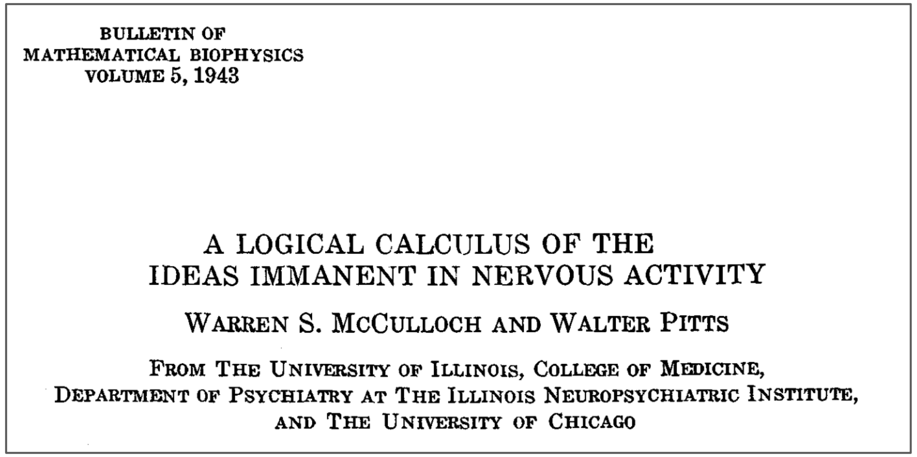


$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

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



Originated from computationally modeling neurons:



BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

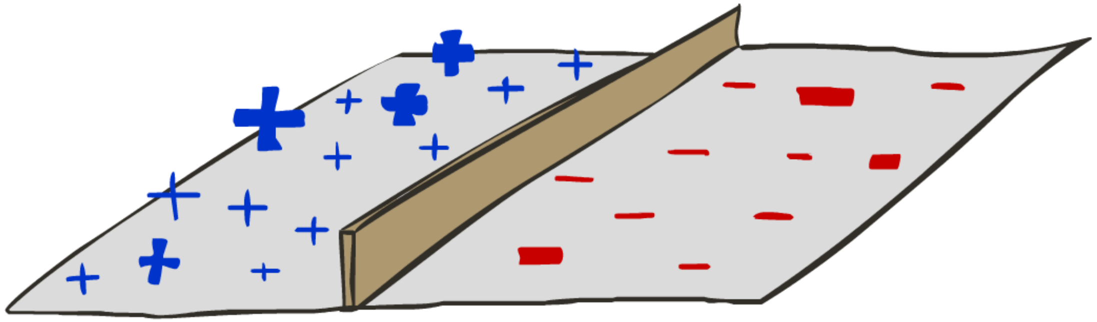
A LOGICAL CALCULUS OF THE
IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

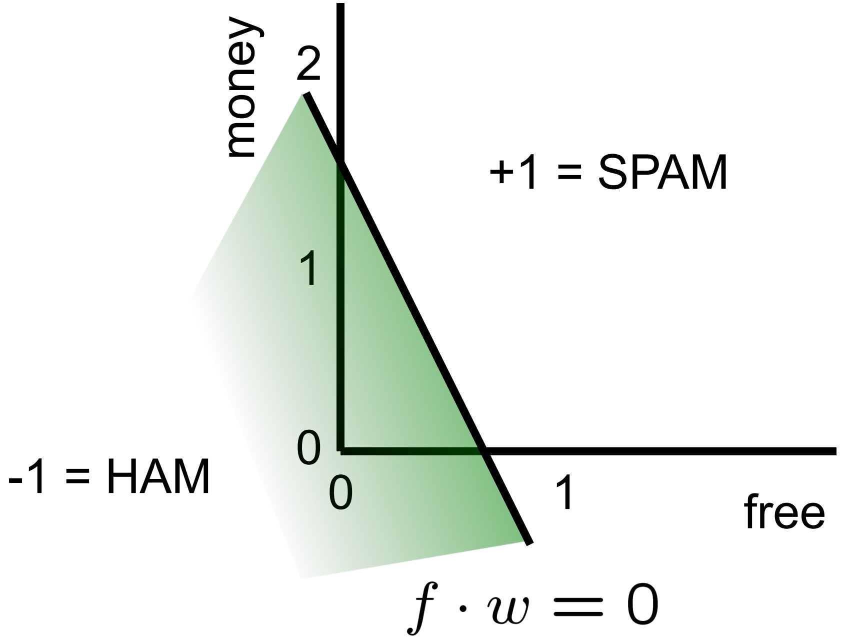
Binary Decision Rule

- 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
...
```



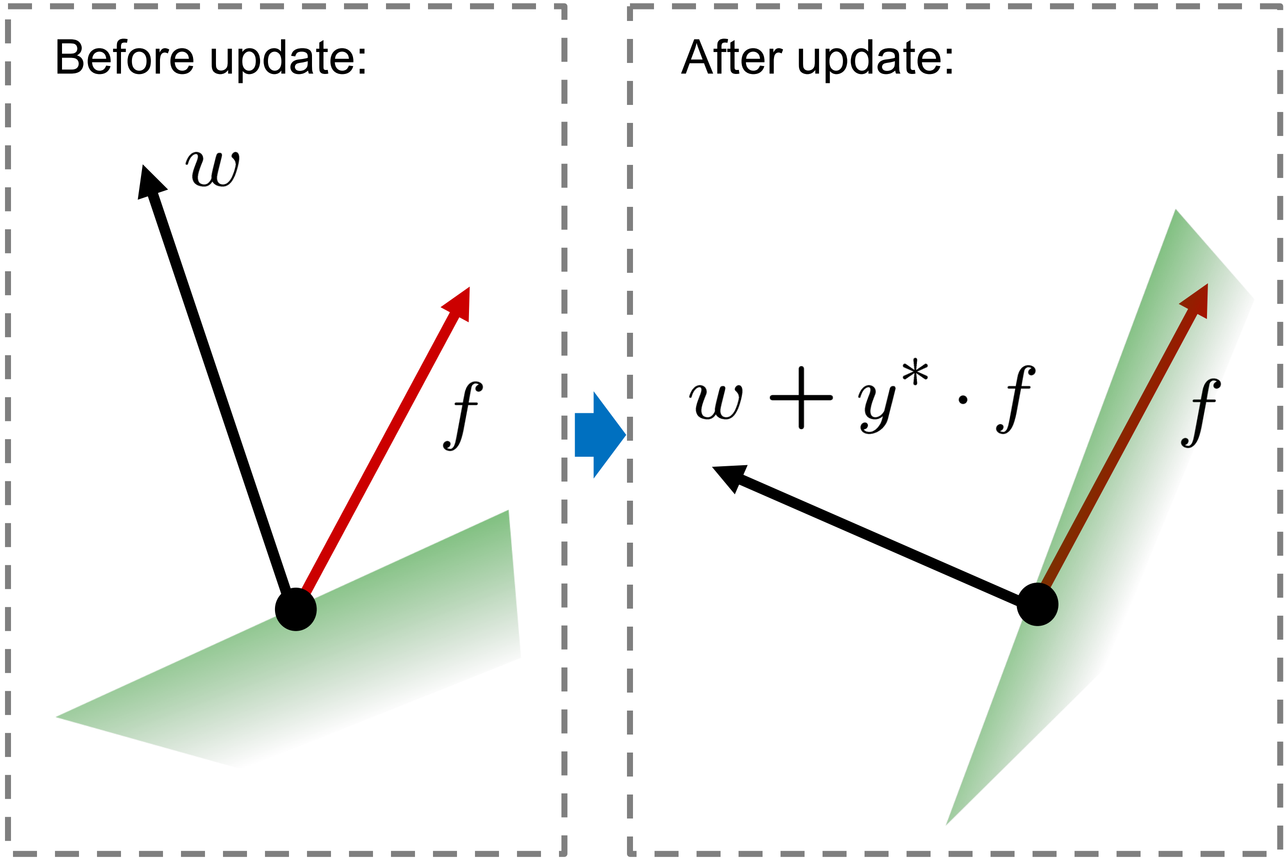
Learning: Binary Perceptron

- Start with weights w = 0
- For each training instance f(x), y*:
  - Classify with current weights

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

  - **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.

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



$$w \cdot f$$

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

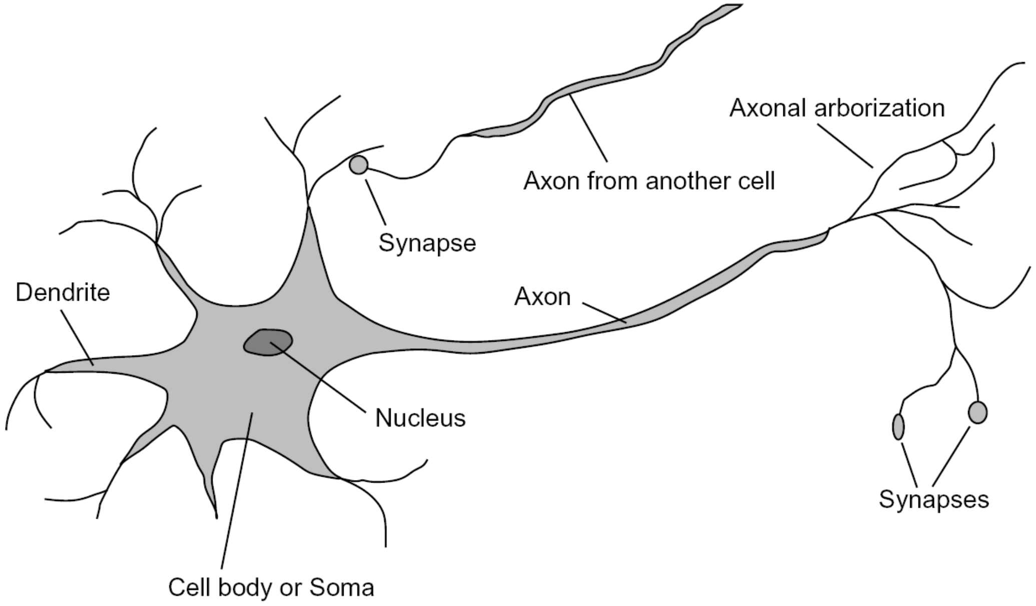
Learning: Binary Perceptron

- Start with weights w = 0
- For each training instance f(x), y*:
  - 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.

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



Inspired by a model of how neural connections develop:

> *"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."*
>
> - Donald Hebb, Organization of Behavior, 1949
>
> TL;DR: *"Neurons that fire together, wire together"*

Learning: Binary Perceptron

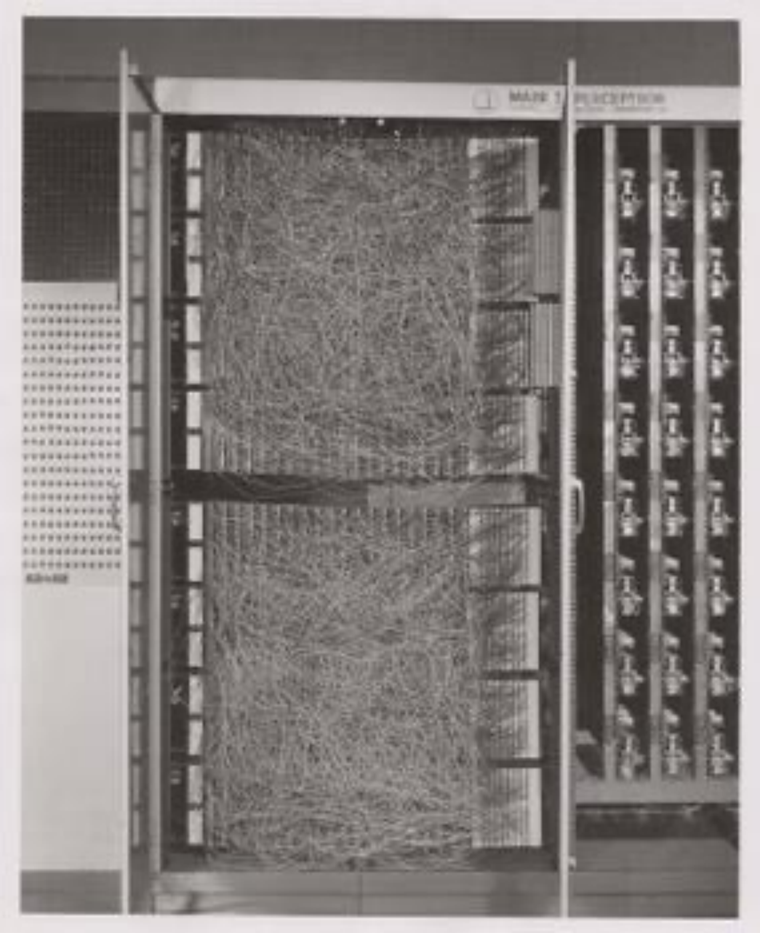
- Start with weights w = 0
- For each training instance f(x), y*:
  - Classify with current weights

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

  - 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.

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

Hardware implementation built by Rosenblatt in 1957:



[Wikipedia]

Multiclass Decision Rule

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

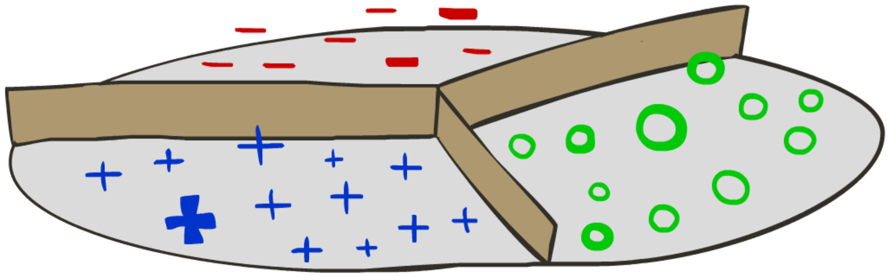
    $$w_y$$

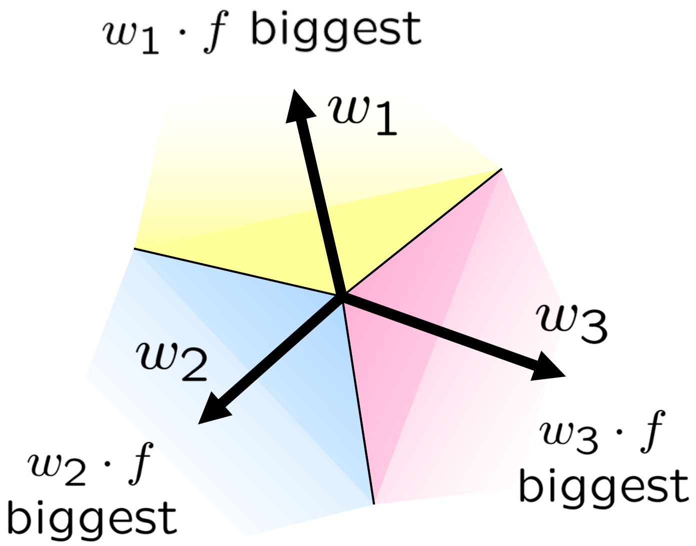
  - Score (activation) of a class y:

    $$w_y \cdot f(x)$$

  - 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

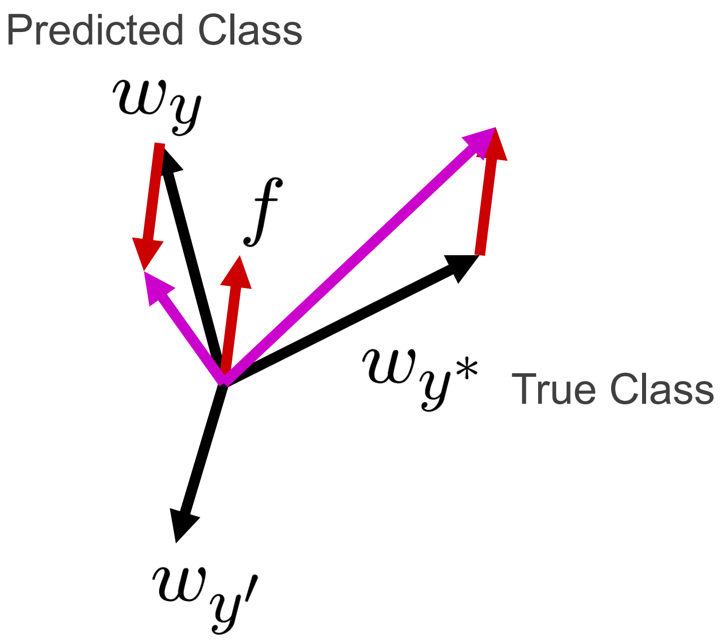
- Start with all weights = 0
- Pick up training examples f(x), y* one by one
- Predict with current weights

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

- **If correct:** no change!
- **If wrong:** lower score of wrong answer, raise score of right answer

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

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



Learning: Multiclass Perceptron

- Start with all weights = 0
- Pick up training examples f(x), y* one by one
- Predict with current weights

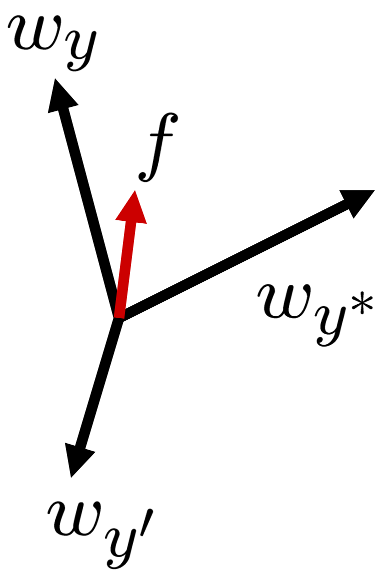
$$y = \arg\max_y \; w_y \cdot f(x)$$

- **If correct:** no change!
- **If wrong:** lower score of wrong answer, raise score of right answer

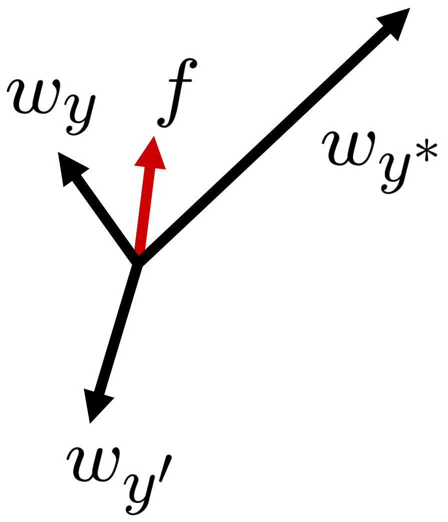
$$w_y = w_y - f(x)$$

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

Before update:



After update:



Score of wrong class:

$$w_y \cdot f$$

Score of right class:

$$w_{y^*} \cdot f$$

Score of wrong class:

$$(w_y - f) \cdot f$$
$$= w_y \cdot f - f \cdot f$$

Score of right class:

$$w_{y^*} \cdot f + f \cdot f$$

Example: Multiclass Perceptron

Iteration 0: x: "*win the vote*" f(x): [1 1 0 1 1] y*: politics

Iteration 1: x: "*win the election*" f(x): [1 1 0 0 1] y*: politics

Iteration 2: x: "*win the game*" f(x): [1 1 1 0 1] y*: sports

w_{SPORTS}

BIAS	1	0	0	1
win	0	-1	-1	0
game	0	0	0	1
vote	0	-1	-1	-1
the	0	-1	-1	0

$w \cdot f(x)$: 1 -2 -2

$w_{POLITICS}$

BIAS	0	1	1	0
win	0	1	1	0
game	0	0	0	-1
vote	0	1	1	1
the	0	1	1	0

$w \cdot f(x)$: 0 3 3

w_{TECH}

BIAS	0	0	0	0
win	0	0	0	0
game	0	0	0	0
vote	0	0	0	0
the	0	0	0	0

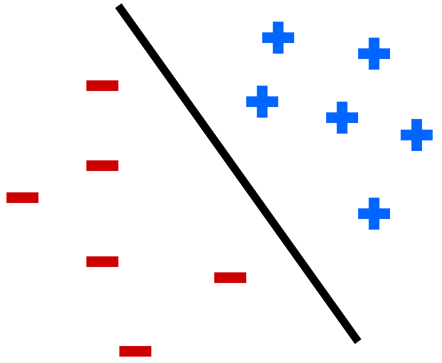
$w \cdot f(x)$: 0 0 0

Properties of Perceptrons

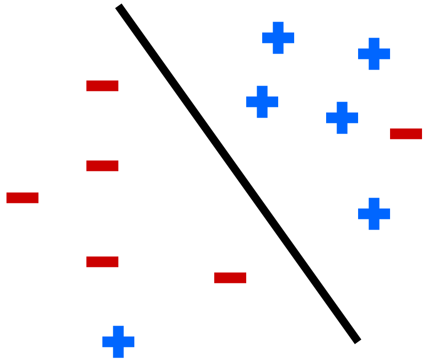
- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\#\text{ of mistakes during training} < \frac{\#\text{ of features}}{(\text{width of margin})^2}$$

Separable

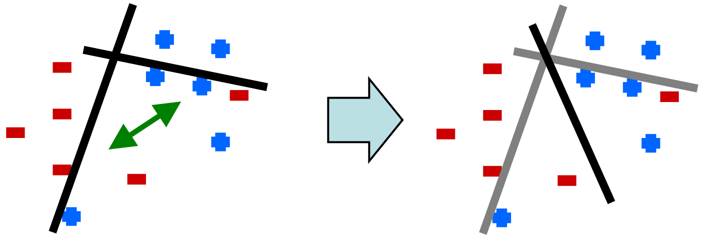


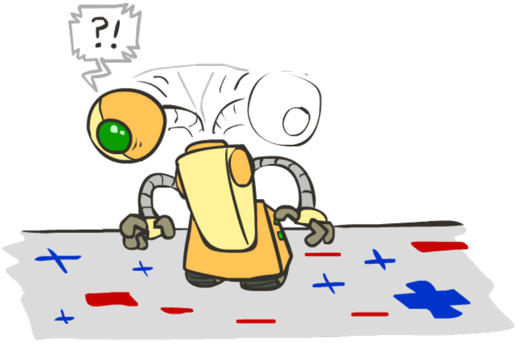
Non-Separable



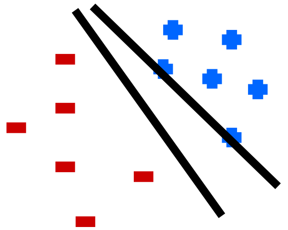
Problems with the Perceptron

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

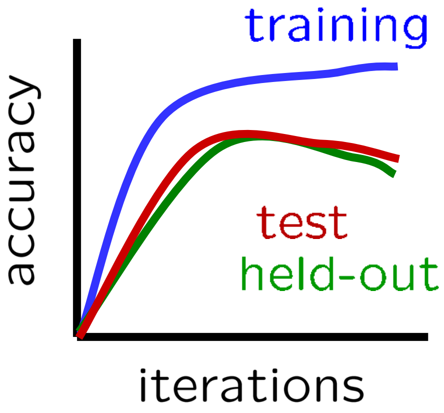


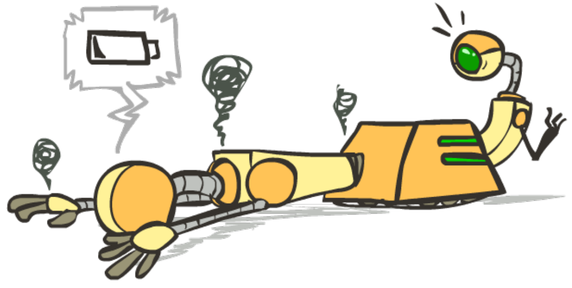


- Mediocre generalization: finds a "barely" separating solution

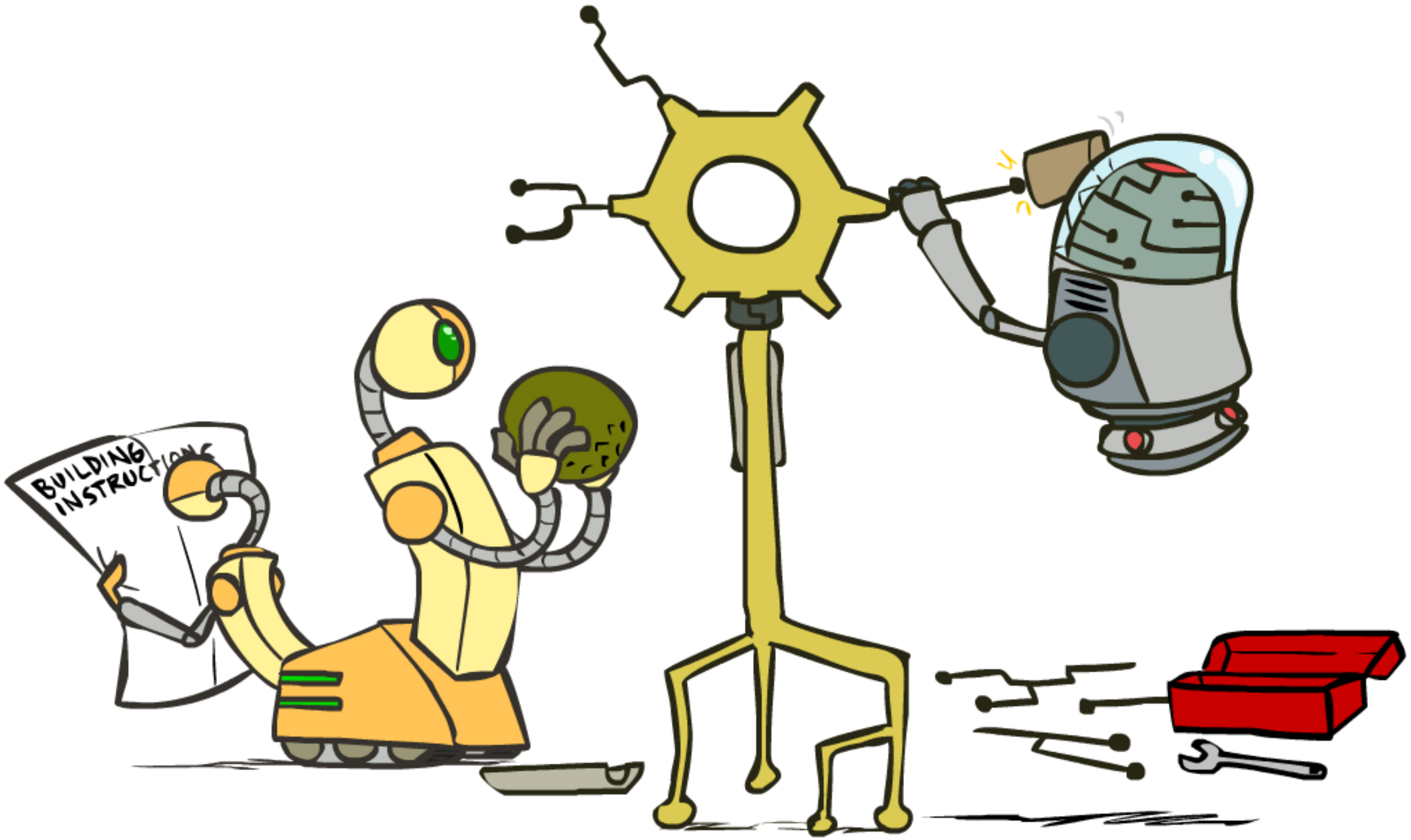


- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



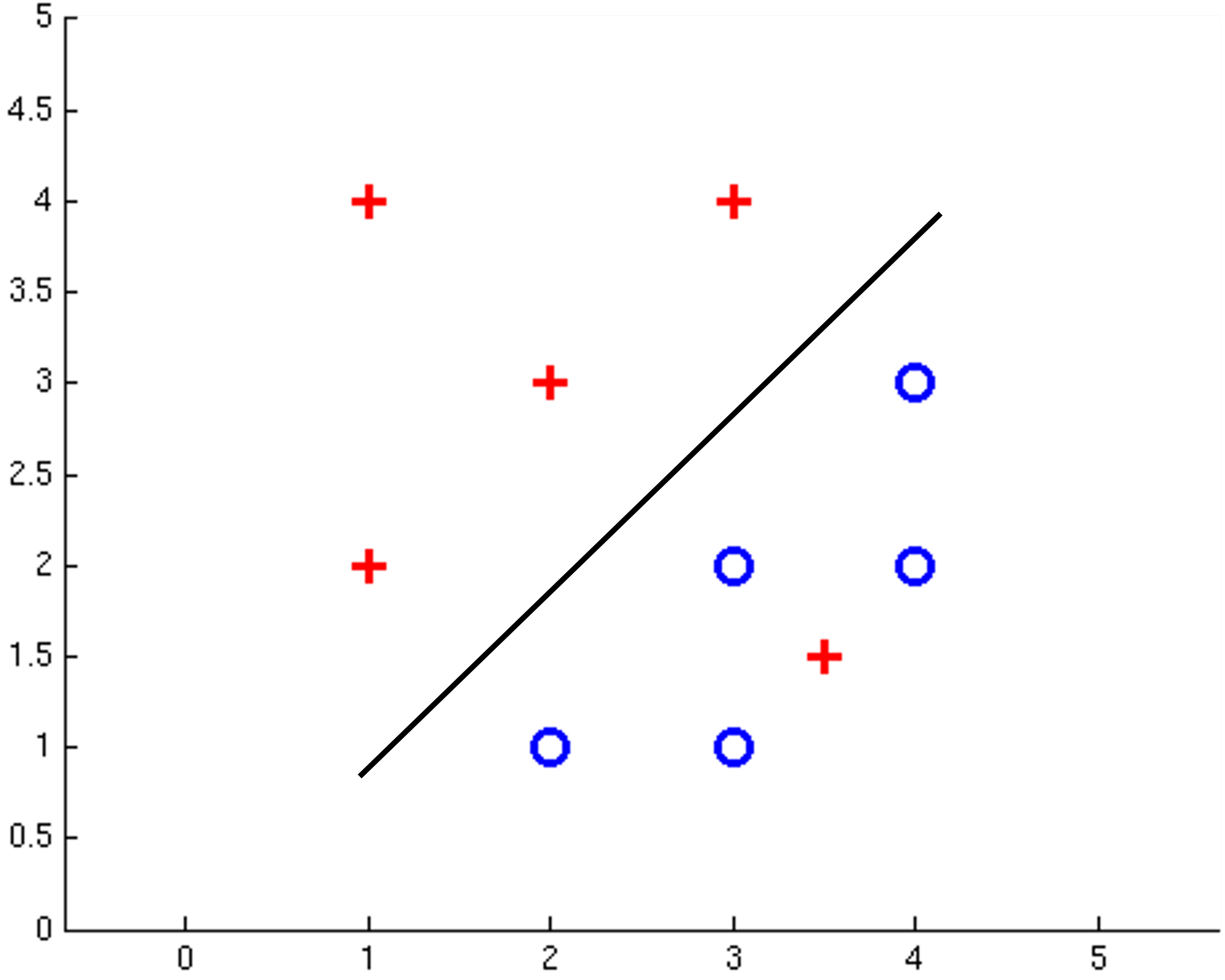


Improving the Perceptron

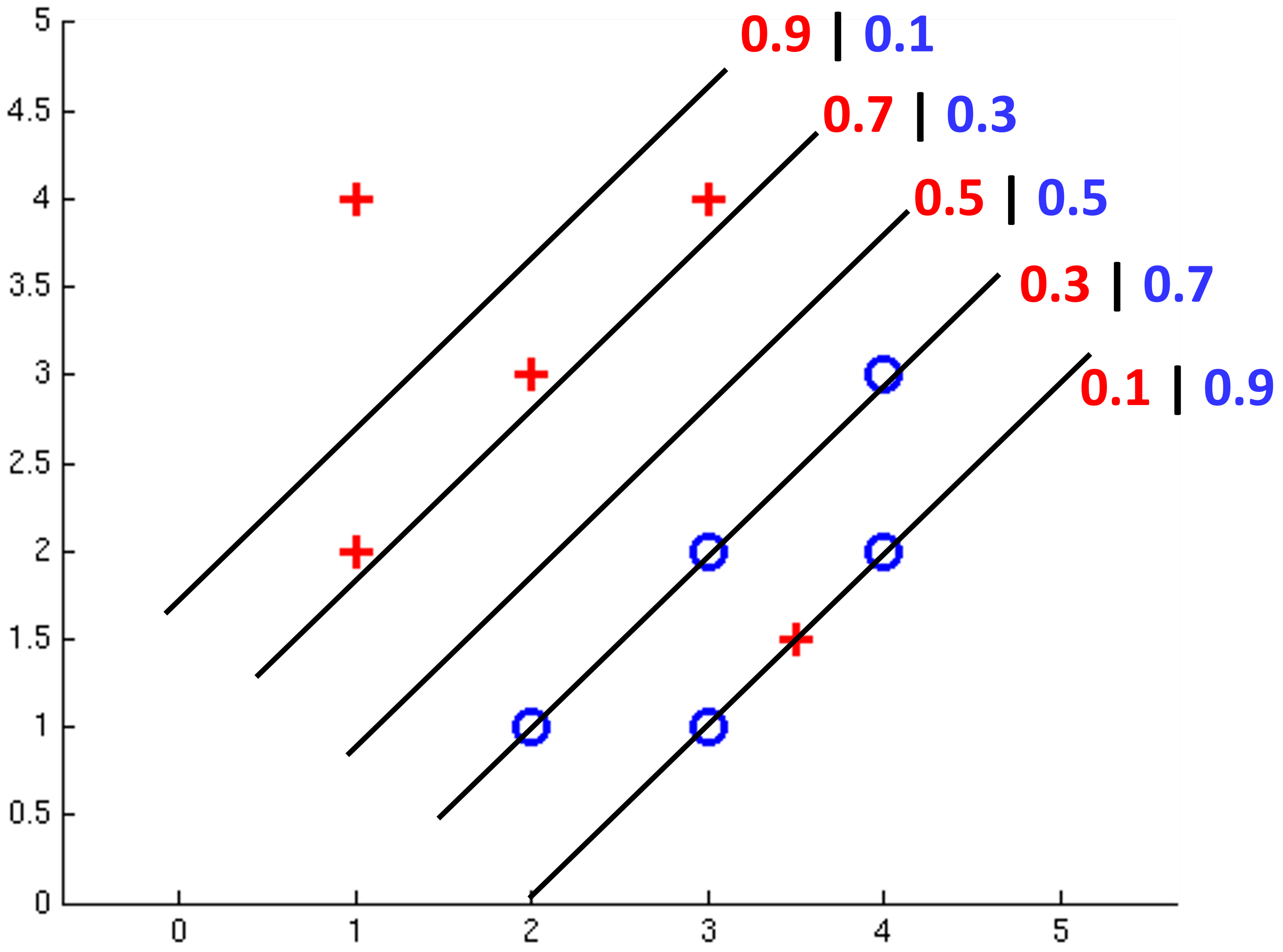


Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake

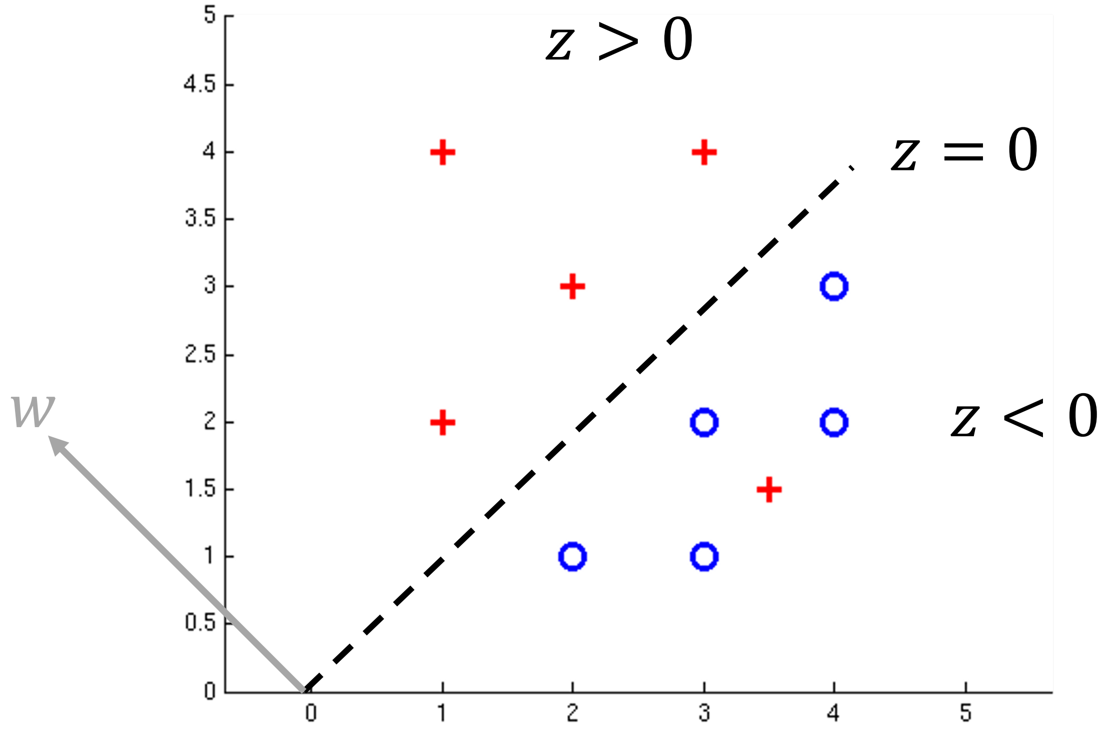


Non-Separable Case: Probabilistic Decision



How to get probabilistic decisions?

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



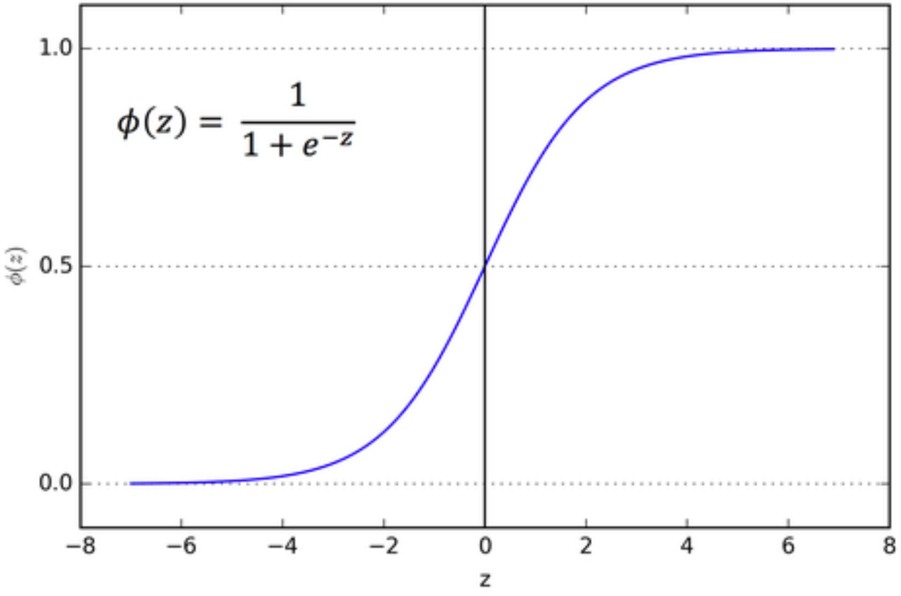
How to get probabilistic decisions?

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

- Sigmoid function

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

$$= \frac{e^z}{e^z+1}$$



How to get probabilistic decisions?

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

- Sigmoid function

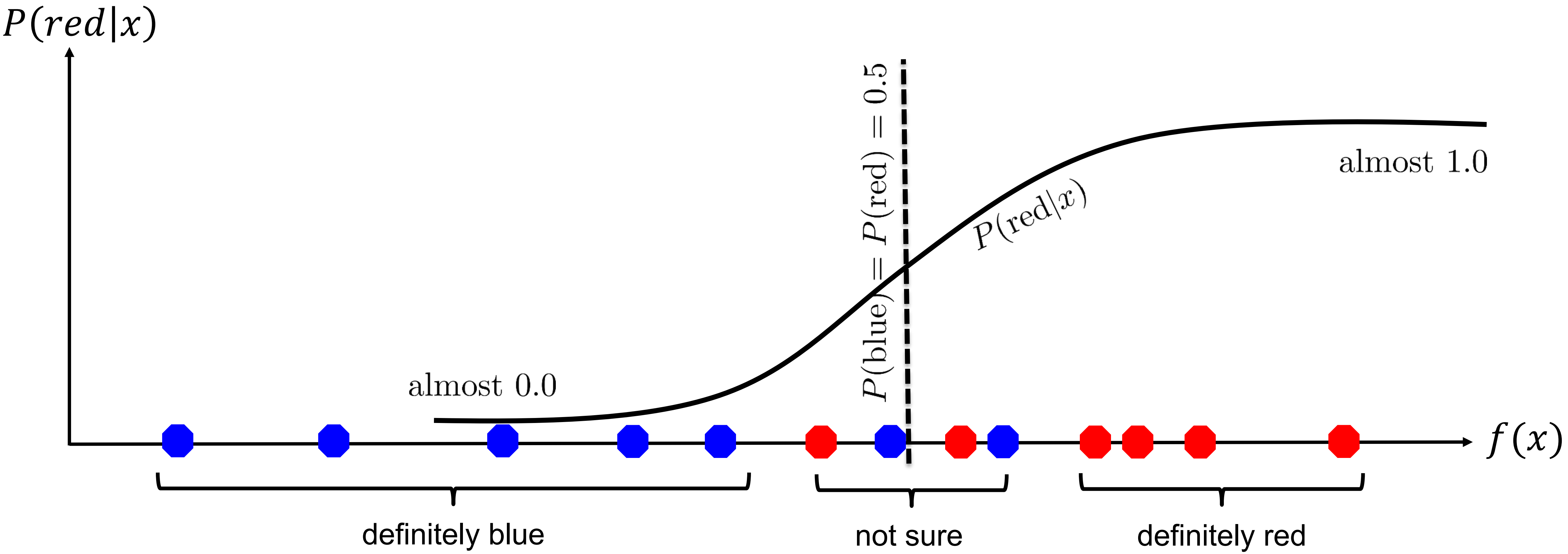
$$\phi(z) = \frac{1}{1 + e^{-z}}$$

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

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

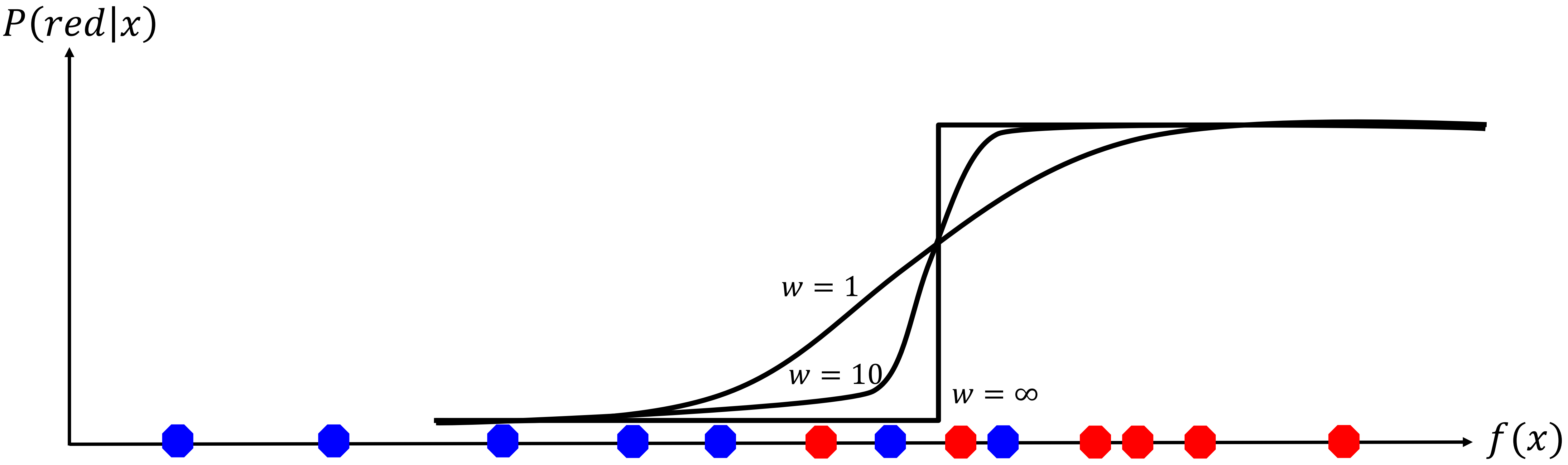
= Logistic Regression

A 1D Example



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

A 1D Example: varying w



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

Best w?

- Recall **maximum likelihood estimation**: Choose the w value that maximizes the probability of the observed (training) data

$$
\begin{aligned}
\text{Likelihood} &= P(\text{training data}|w) \\
&= \prod_i P(\text{training datapoint } i \mid w) \\
&= \prod_i P(\text{point } x^{(i)} \text{ has label } y^{(i)}|w) \\
&= \prod_i P(y^{(i)}|x^{(i)};w) \\
\text{Log Likelihood} &= \sum_i \log P(y^{(i)}|x^{(i)};w)
\end{aligned}
$$

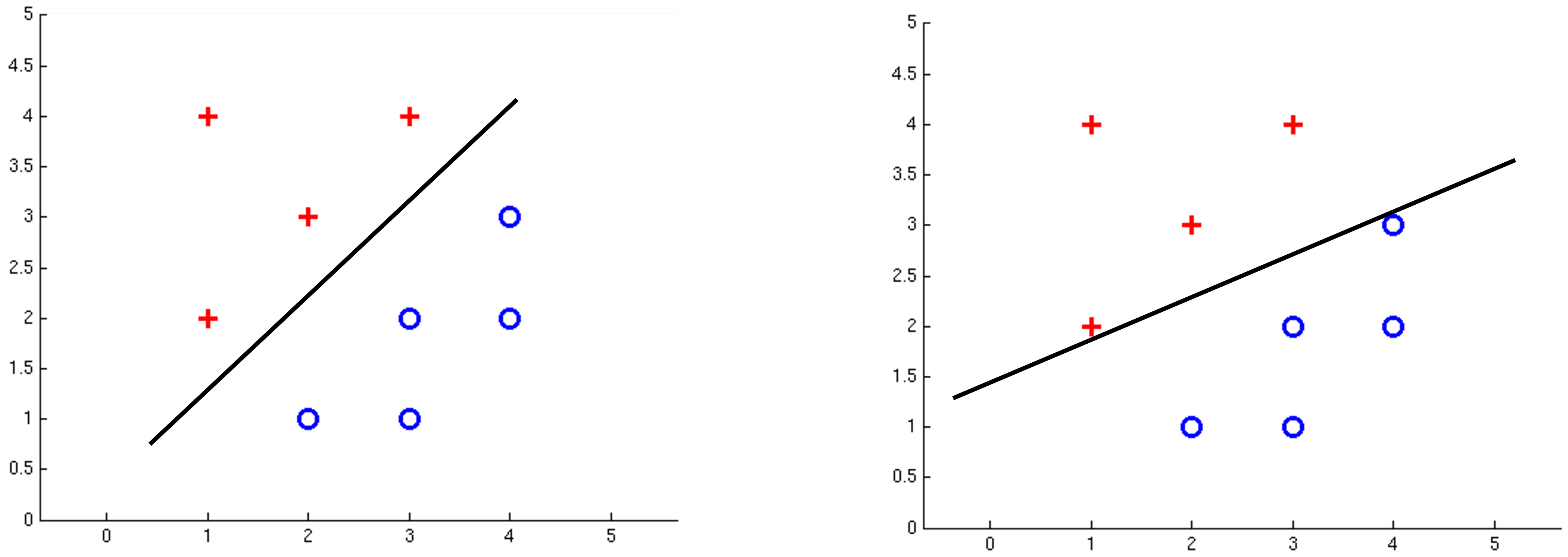
Best w?

- **Recall maximum likelihood estimation**: Choose the w value that maximizes the probability of the observed (training) data

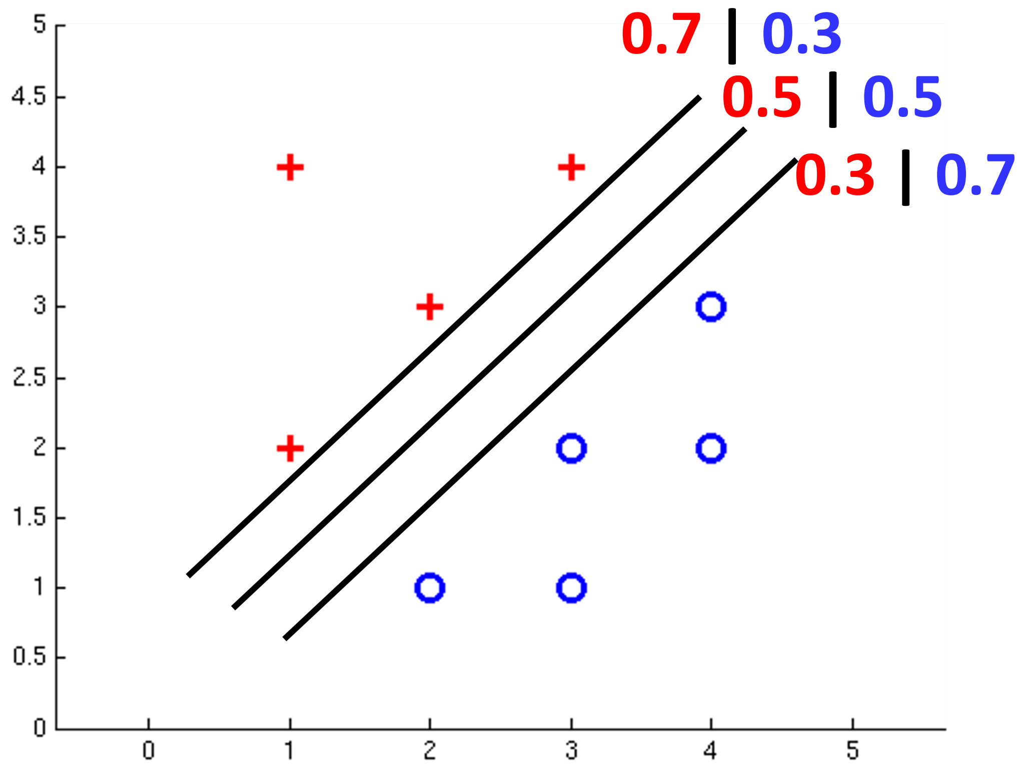
$$\begin{aligned} & P(\text{point } x^{(i)} \text{ has label } y^{(i)} = +1 \mid w) \\ = \ & P(y^{(i)} = +1 \mid x^{(i)}; w) \\ = \ & \frac{1}{1 + e^{-w \cdot x^{(i)}}} \end{aligned}$$

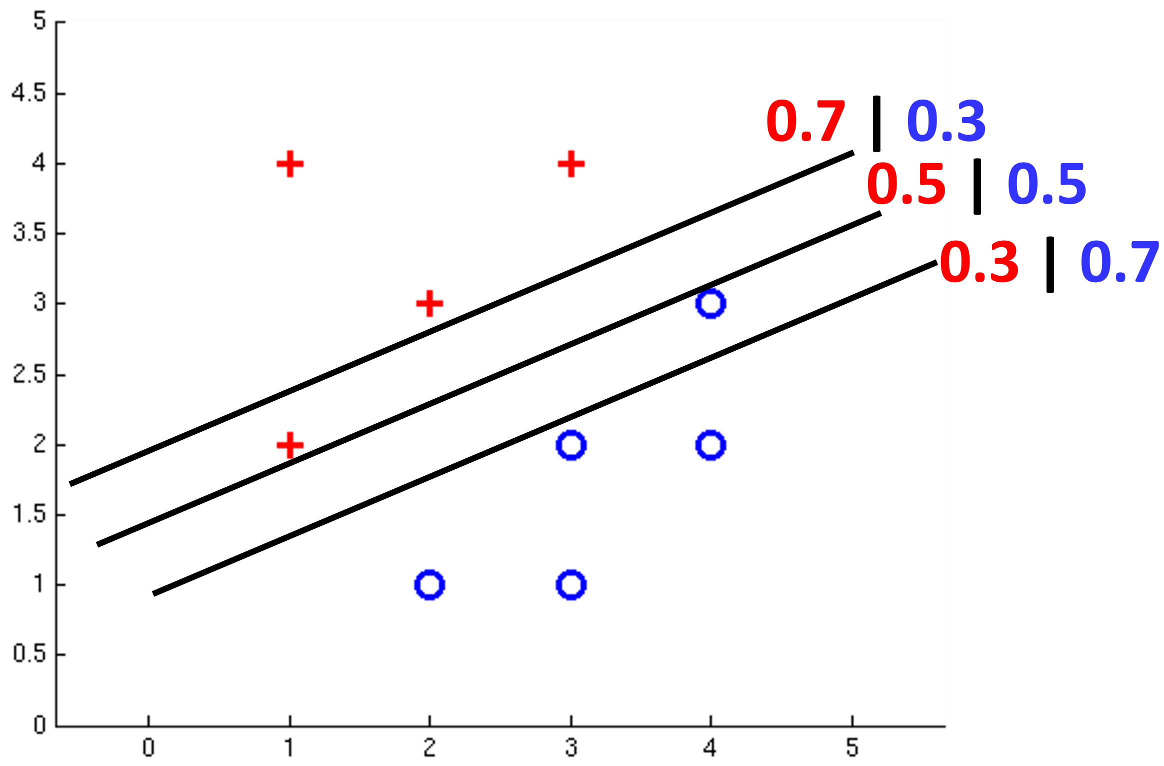
$$\begin{aligned} & P(\text{point } x^{(i)} \text{ has label } y^{(i)} = -1 \mid w) \\ = \ & P(y^{(i)} = -1 \mid x^{(i)}; w) \\ = \ & 1 - \frac{1}{1 + e^{-w \cdot x^{(i)}}} \end{aligned}$$

Separable Case: Deterministic Decision – Many Options



Separable Case: Probabilistic Decision – Clear Preference

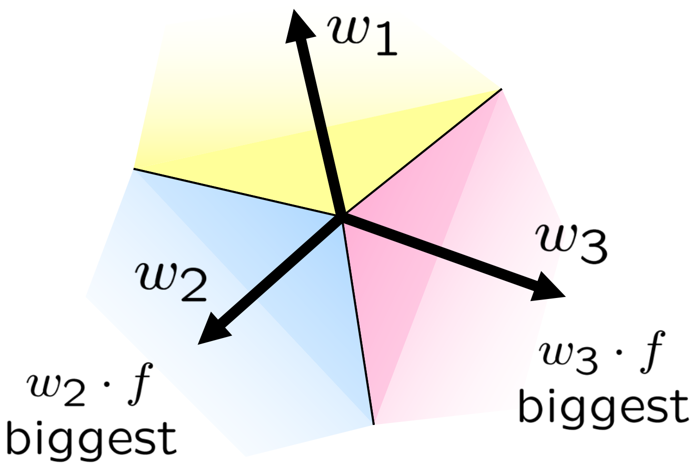




Multiclass Logistic Regression

Multiclass Logistic Regression

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



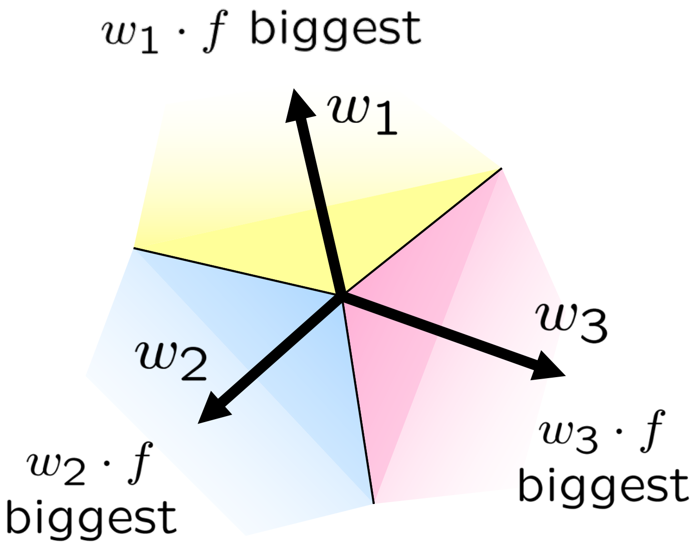
- How to make the scores into probabilities?

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

  - In general: $\mathrm{softmax}(z_1, \ldots, z_n) = [\frac{e^{z_1}}{\sum_i e^{z_i}}, \ldots, \frac{e^{z_n}}{\sum_i e^{z_i}}]$

Multiclass Logistic Regression

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



- How to make the scores into probabilities?

$$P(y \mid x ; w) = \frac{e^{w_y \cdot f(x)}}{\sum_{y'} e^{w_{y'} \cdot f(x)}}$$

= Multi-Class Logistic Regression

Best w?

- Recall **maximum likelihood estimation**: Choose the w value that maximizes the probability of the observed (training) data

$$\begin{aligned}
\text{Likelihood} &= P(\text{training data}|w) \\
&= \prod_i P(\text{training datapoint } i \mid w) \\
&= \prod_i P(\text{point } x^{(i)} \text{ has label } y^{(i)}|w) \\
&= \prod_i P(y^{(i)}|x^{(i)};w) \\
\text{Log Likelihood} &= \sum_i \log P(y^{(i)}|x^{(i)};w)
\end{aligned}$$

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

Softmax and Sigmoid

- Recall: Binary perceptron is a special case of multi-class perceptron
  - Multi-class: Compute $w_y \cdot f(x)$ for each class y, pick class with the highest activation
  - Binary case:
    Let the weight vector of +1 be w (which we learn).
    Let the weight vector of -1 always be 0 (constant).
  - Binary classification as a multi-class problem:
    Activation of negative class is always 0.
    If w · f is positive, then activation of +1 (w · f) is higher than -1 (0).
    If w · f is negative, then activation of -1 (0) is higher than +1 (w · f).

Softmax

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

with $w_{red} = 0$ becomes:

Sigmoid

$$P(\mathrm{red}|x) = \frac{1}{1 + e^{-wx}}$$

Naïve Bayes vs Logistic Regression

	Naïve Bayes	**Logistic Regression**
Model	Joint over all features and label: $P(Y, F_1, F_2, \ldots)$	Conditional: $P(y \mid f_1, f_2, \ldots;\ w)$
Predicted class probabilities	Inference in a Bayes Net: $P(Y \mid f) \propto P(Y)\, P(f_1 \mid Y) \ldots$	Directly output label: $P(y = +1 \mid f;\ w) \ = 1/(1 + e^{-w \cdot f})$
Features	Discrete	Discrete or Continuous
Parameters	Entries of probability tables $P(Y)$ and $P(F_k \mid Y)$	Weight vector w
Learning	Counting occurrences of events	Iterative numerical optimization

How do we maximize functions?

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

In general, cannot always take derivative and set to 0

Use numerical optimization!

Hill Climbing

Recall from CSPs lecture: simple, general idea

Start wherever

Repeat: move to the best neighboring state

If no neighbors better than current, quit

What's particularly tricky when hill-climbing for multiclass logistic regression?

- Optimization over a continuous space
  - Infinitely many neighbors!
  - How to do this efficiently?

Next Time: Optimization and Neural Networks!