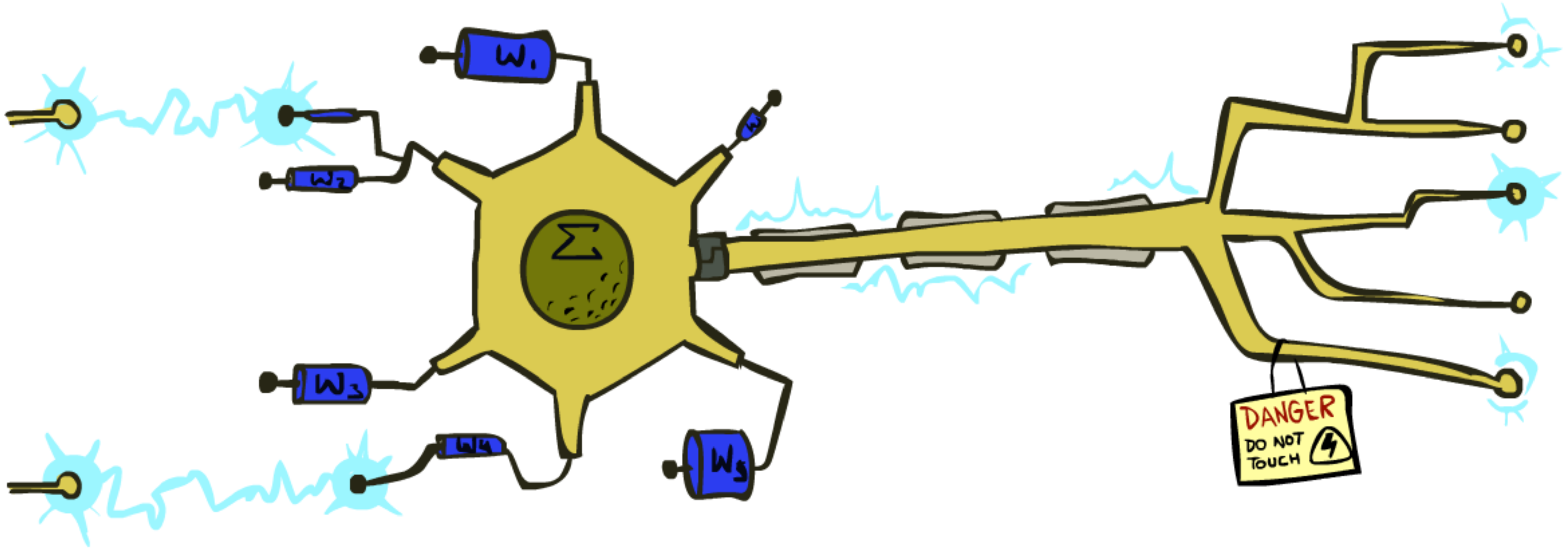
CS 188: Artificial Intelligence

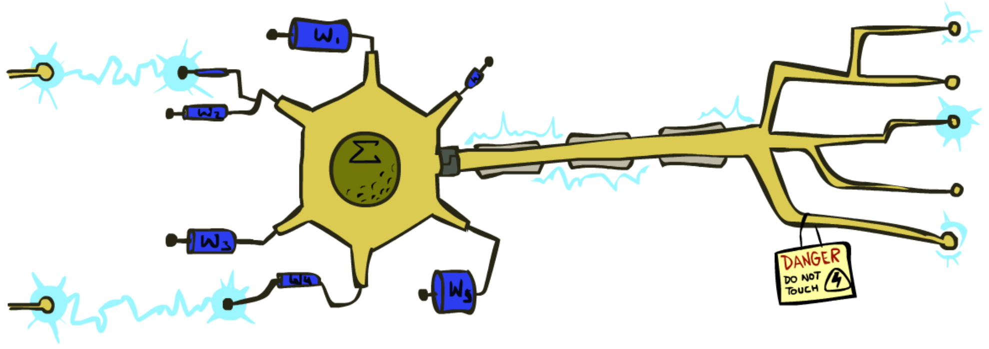
Perceptrons, Logistic Regression and Optimization



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

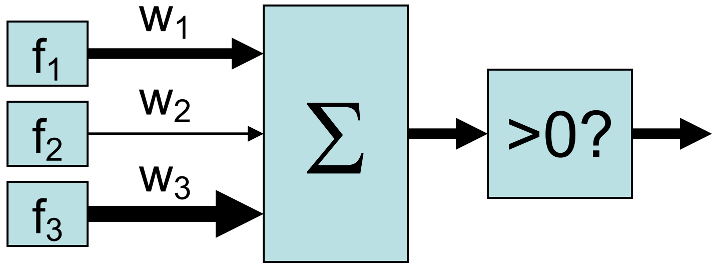
Linear Classifiers

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



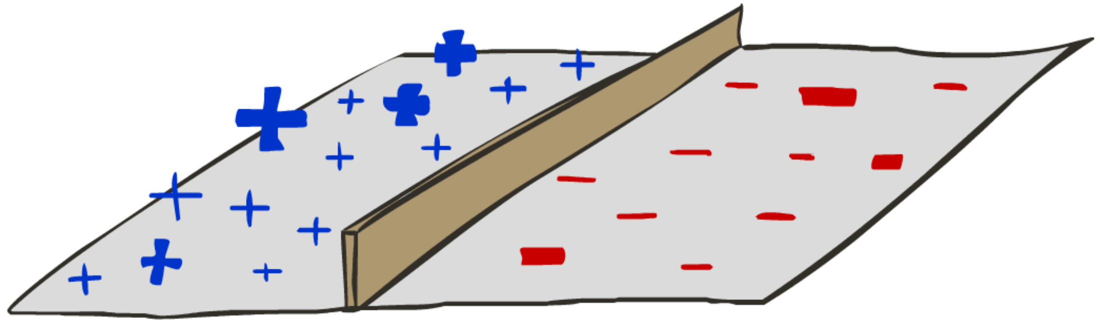
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

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



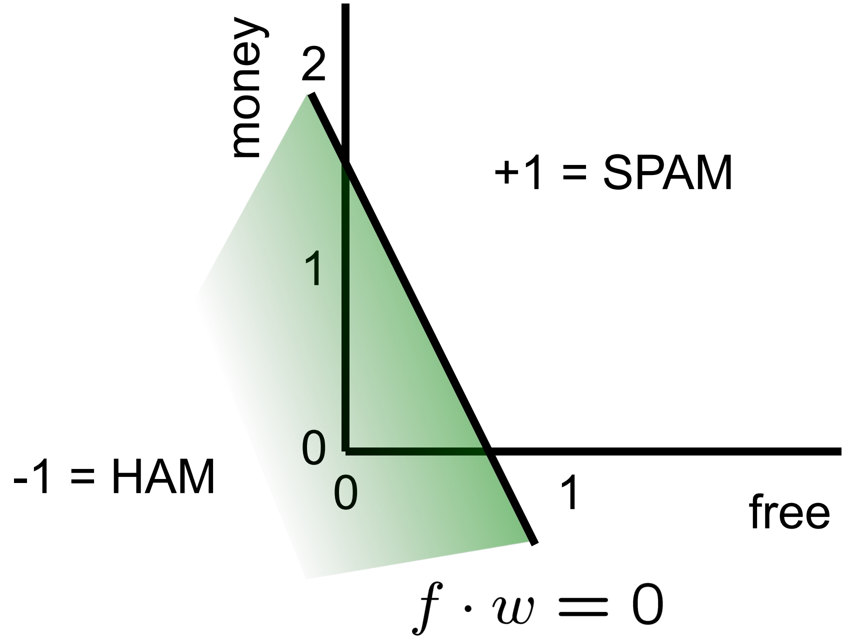
Binary Decision Rule

- 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
...
```



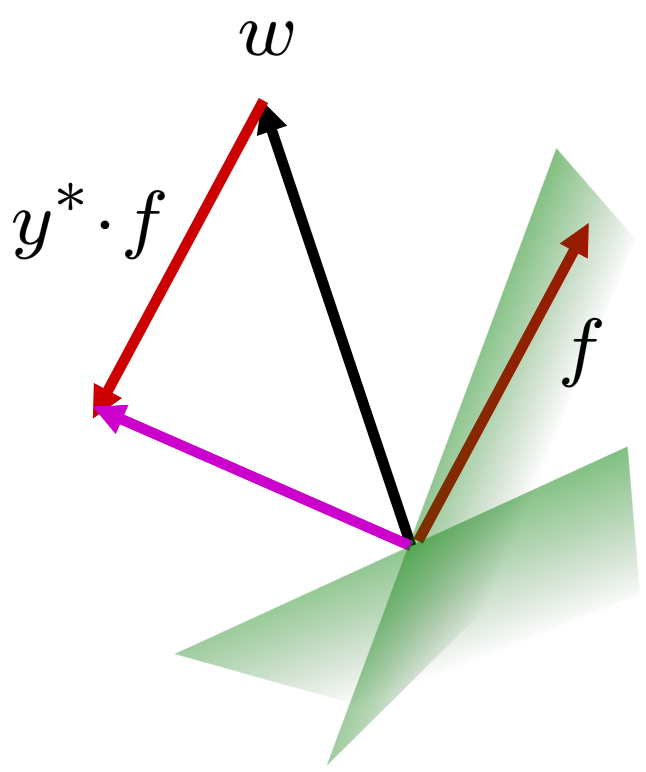
Learning: Binary Perceptron

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

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

  - 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.

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



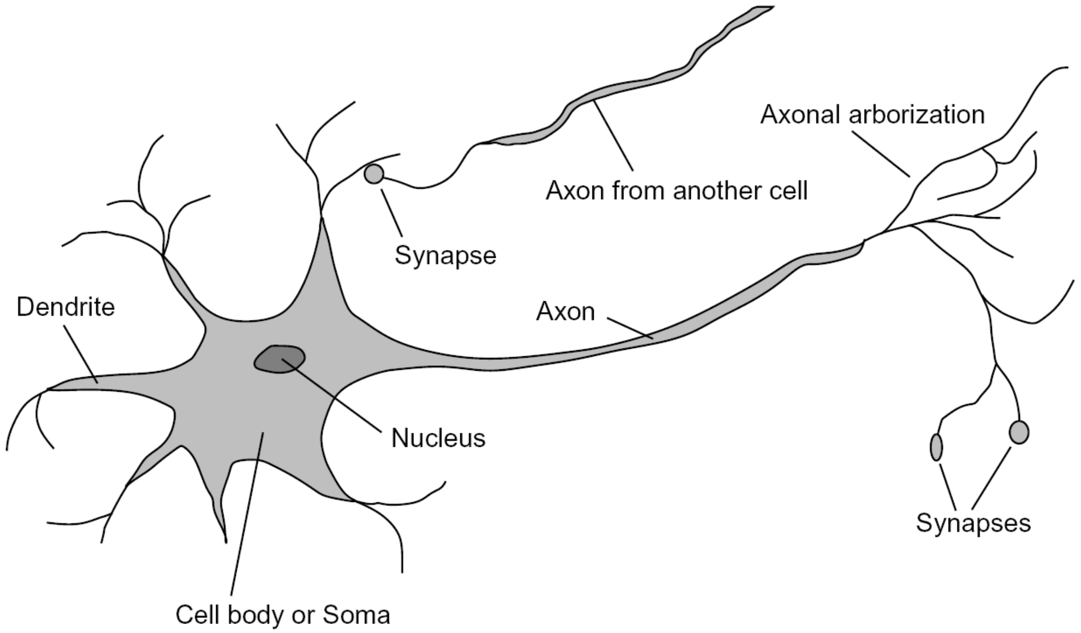
Learning: Binary Perceptron

- 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.

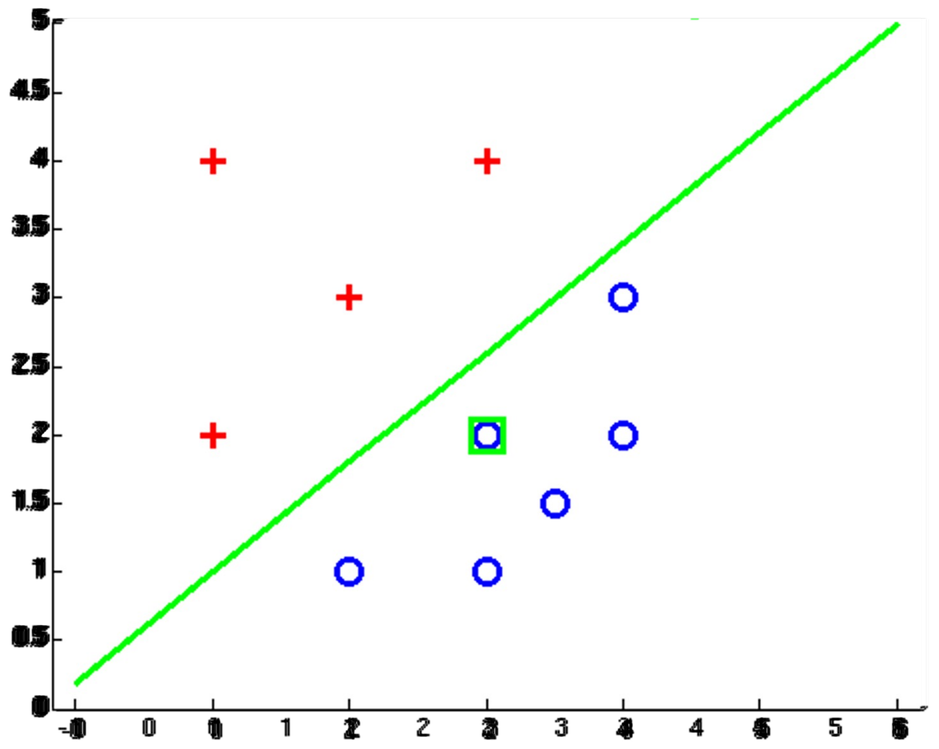
$$w = w + y^* \cdot f$$



"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."
Hebb (1949)

Example: Perceptron

- Separable Case



Multiclass Decision Rule

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

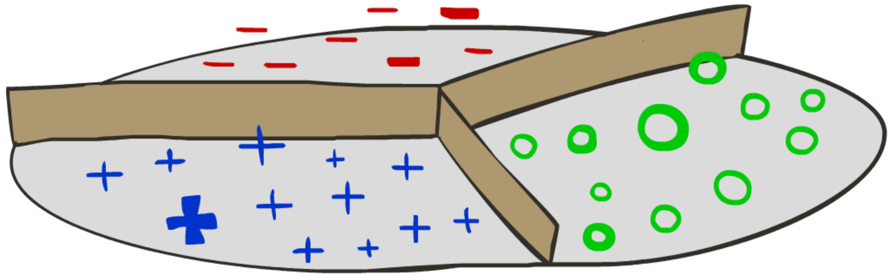
    $$w_y$$

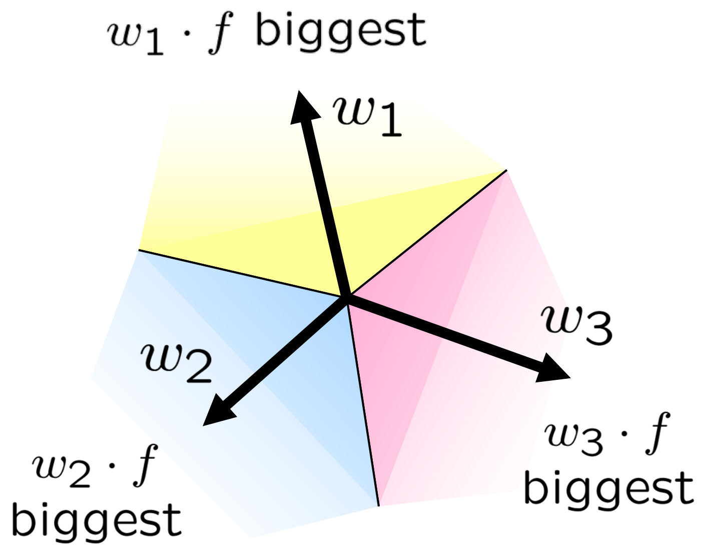
  - Score (activation) of a class y:

    $$w_y \cdot f(x)$$

  - Prediction highest score wins

    $$y = \underset{y}{\arg\max}\ w_y \cdot f(x)$$





Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

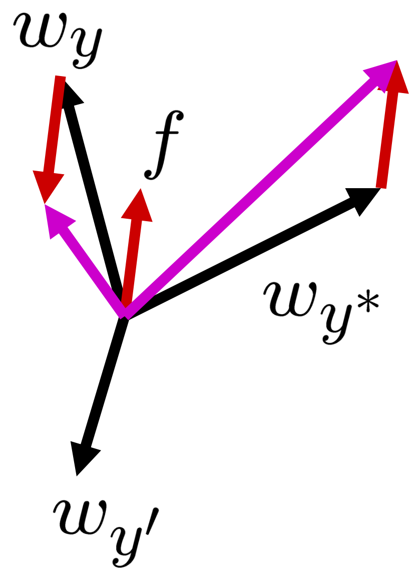
- Start with all weights = 0
- Pick up training examples f(x), y* one by one
- Predict with current weights

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

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

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

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



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	

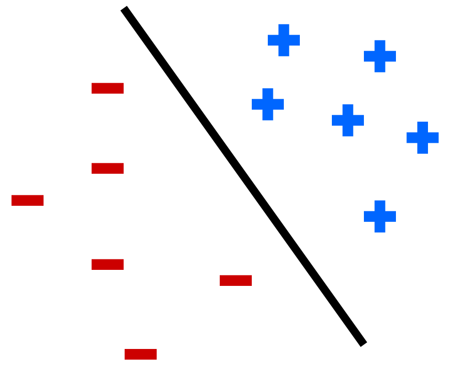
Properties of Perceptrons

- Separability: true if some parameters get the training set perfectly correct

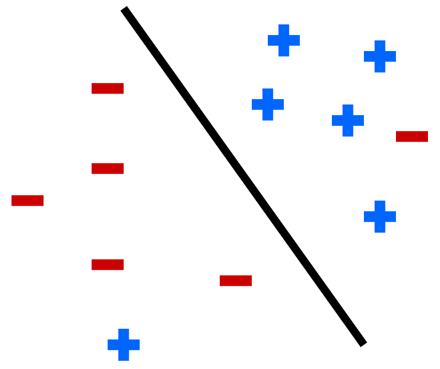
- Convergence: if the training is separable, perceptron will eventually converge (binary case)

- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

Separable

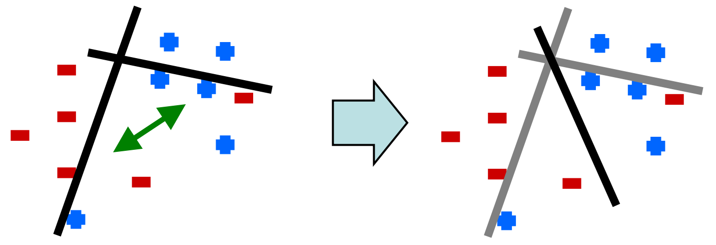


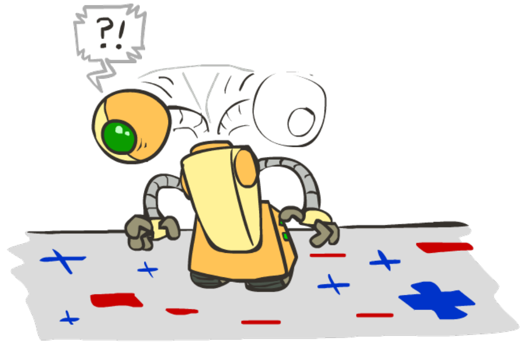
Non-Separable



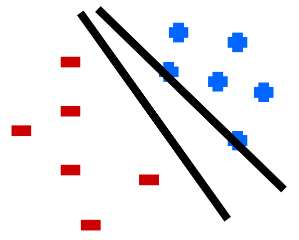
Problems with the Perceptron

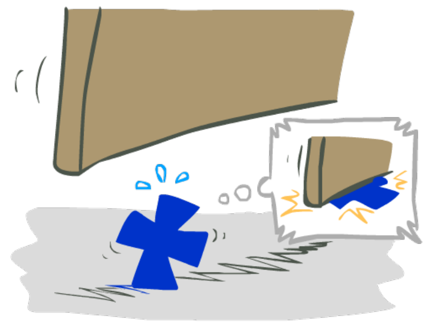
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



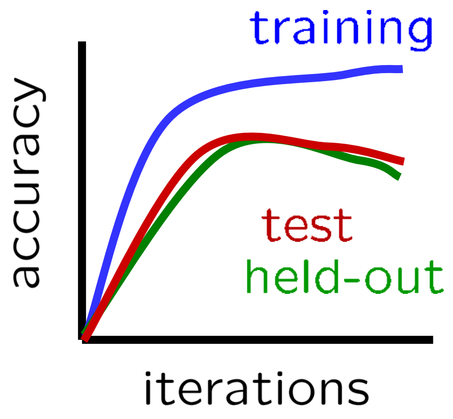


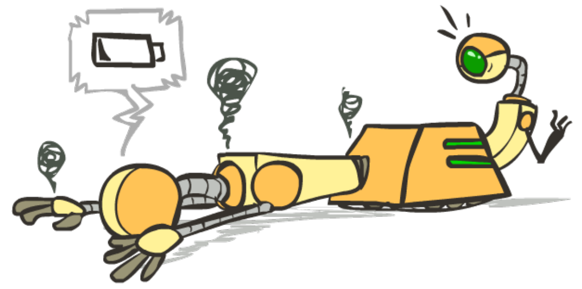
- Mediocre generalization: finds a "barely" separating solution



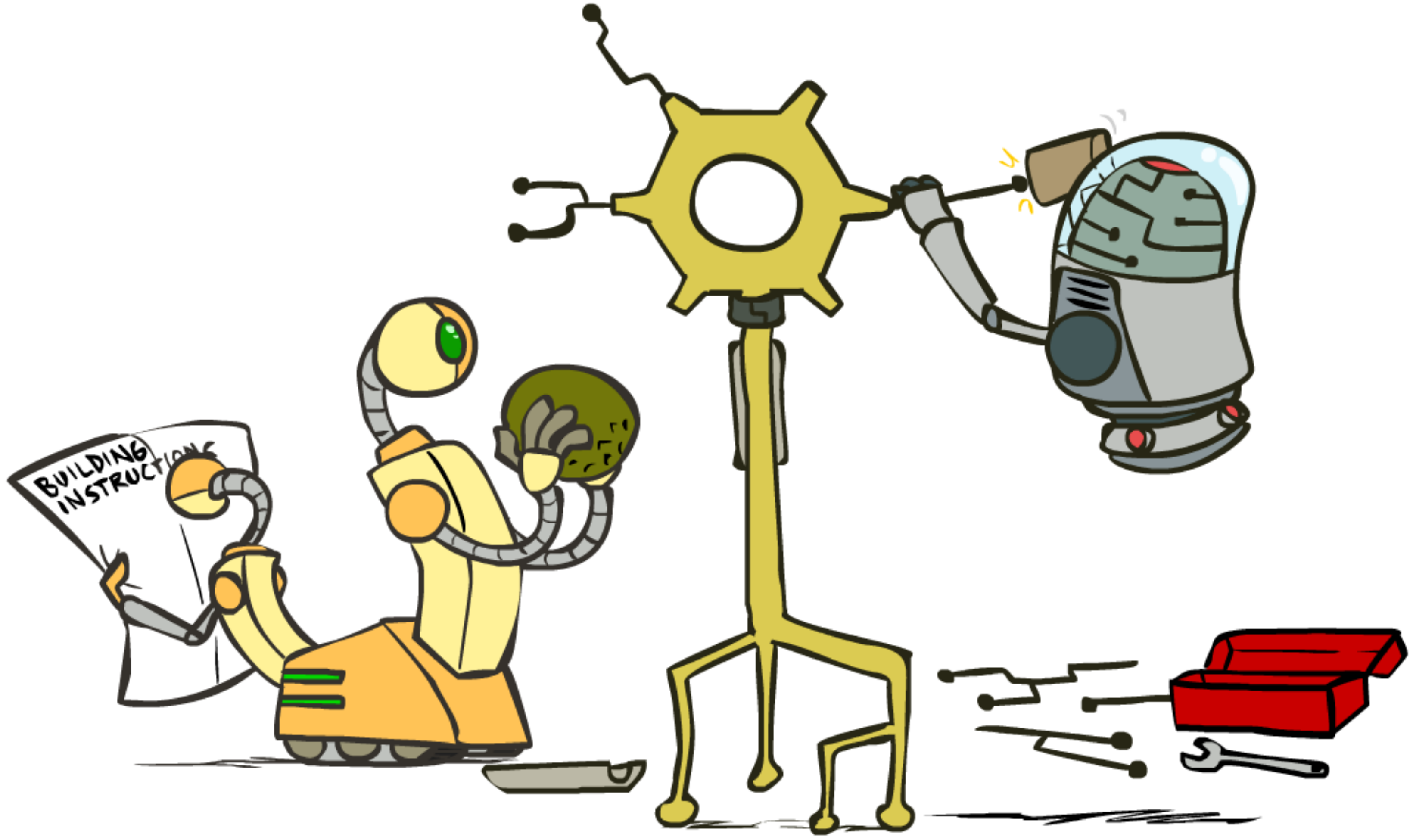


- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



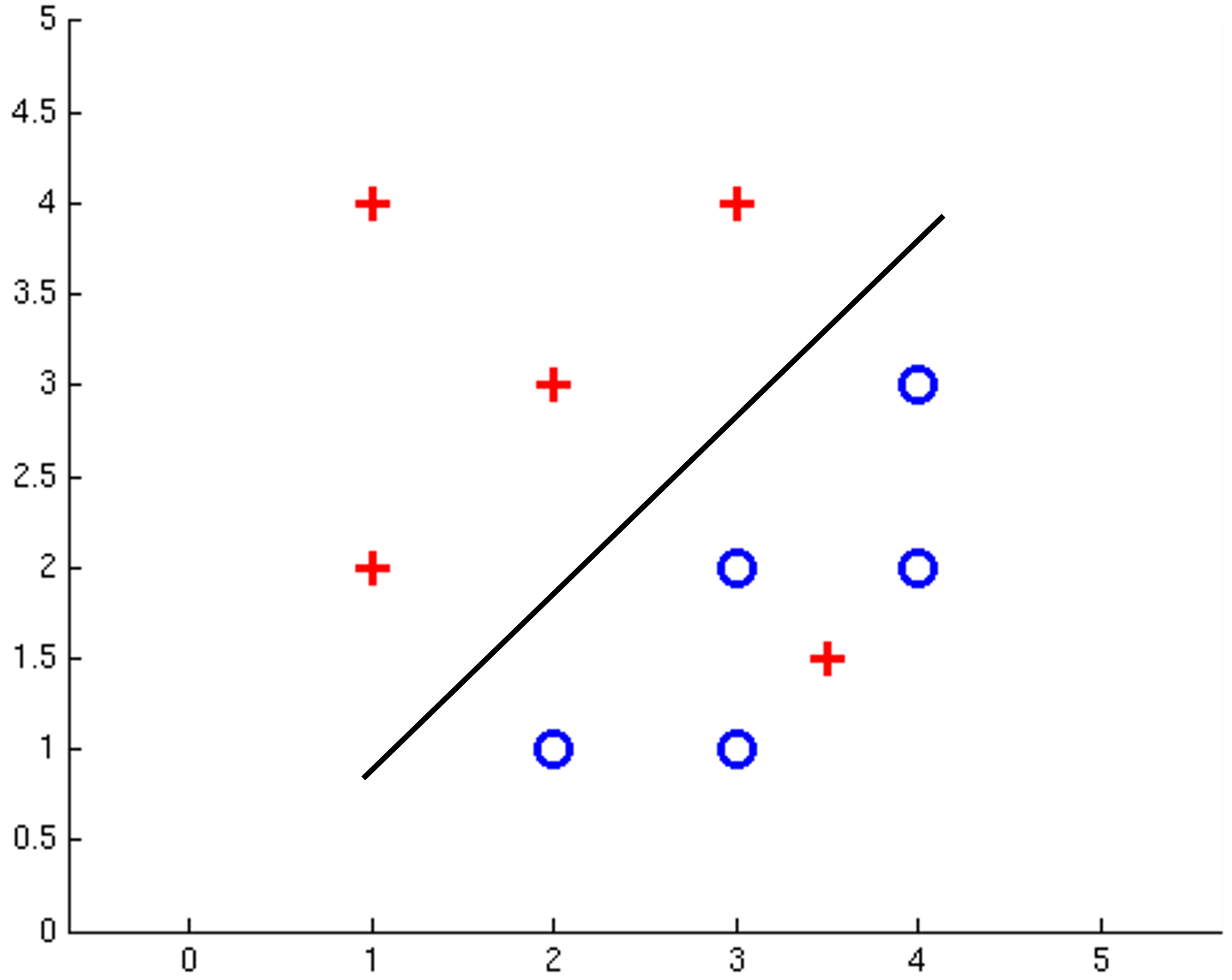


Improving the Perceptron

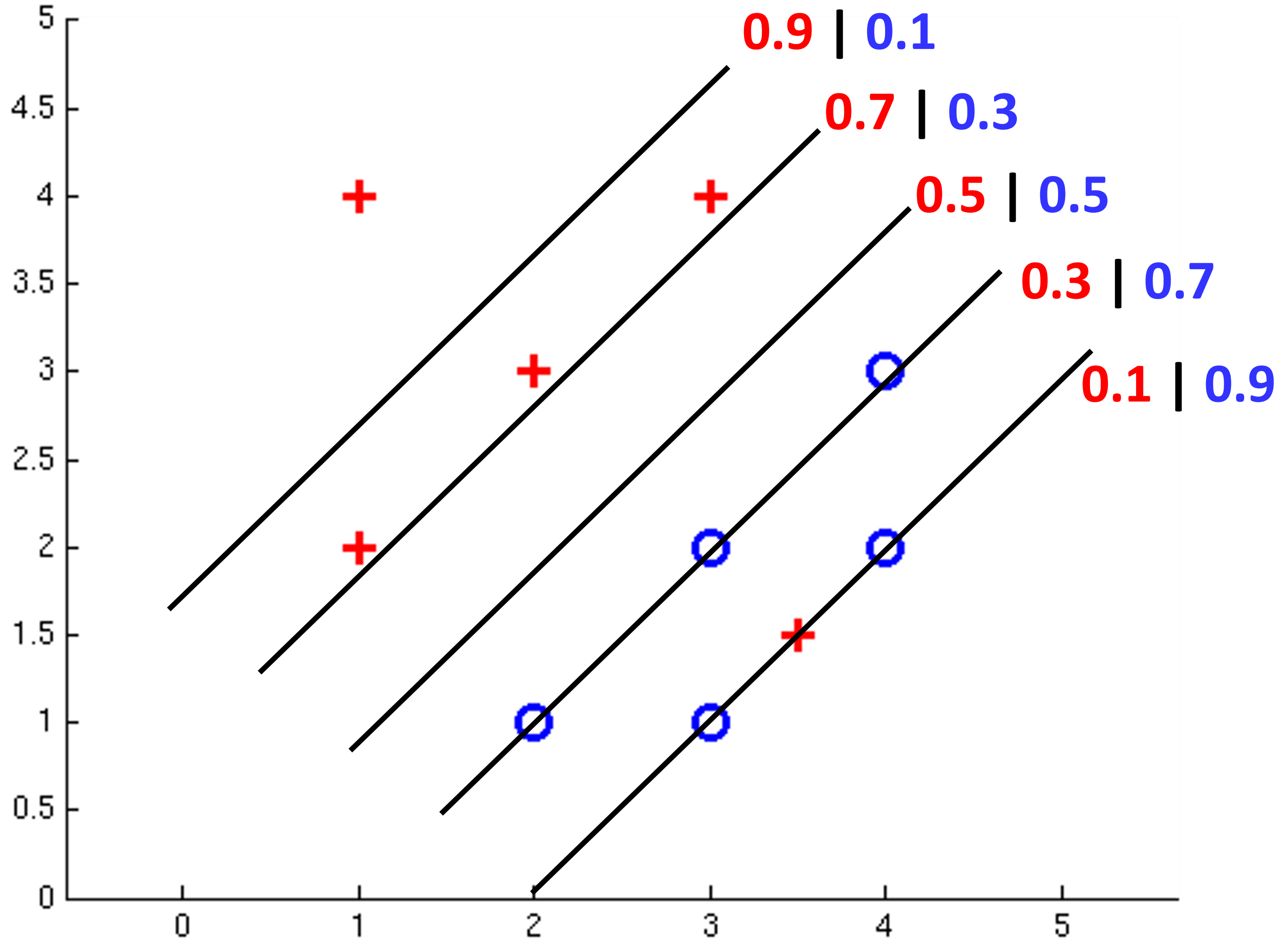


Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake



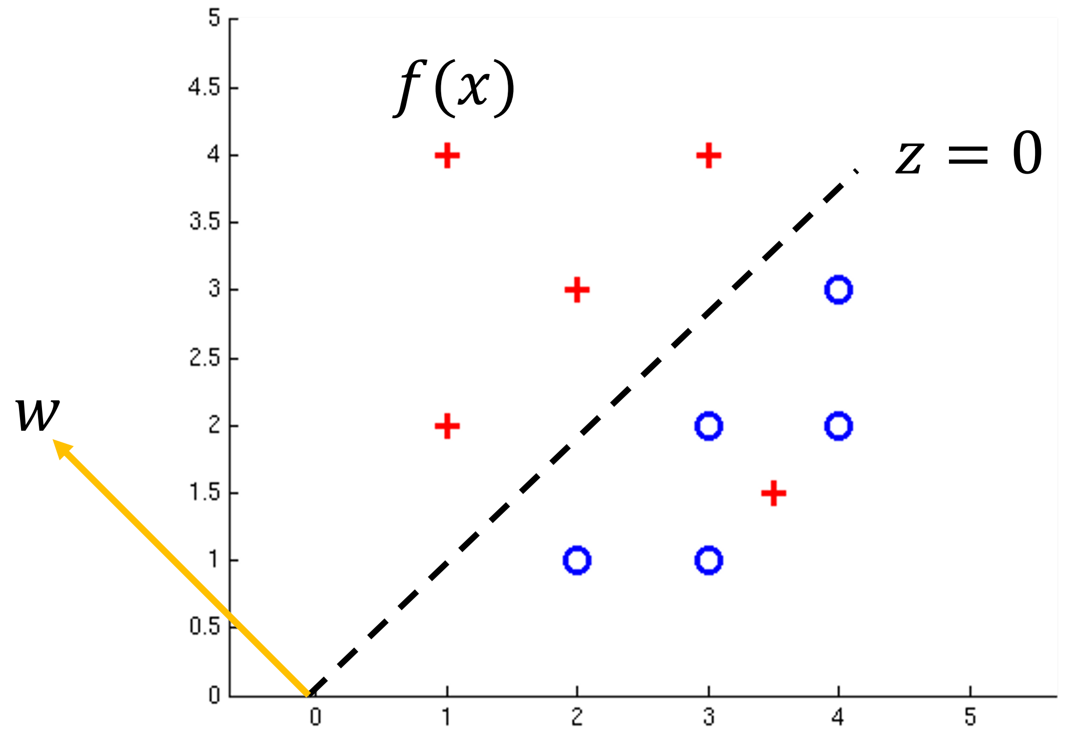
Non-Separable Case: Probabilistic Decision



How to get probabilistic decisions?

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

  - Example:



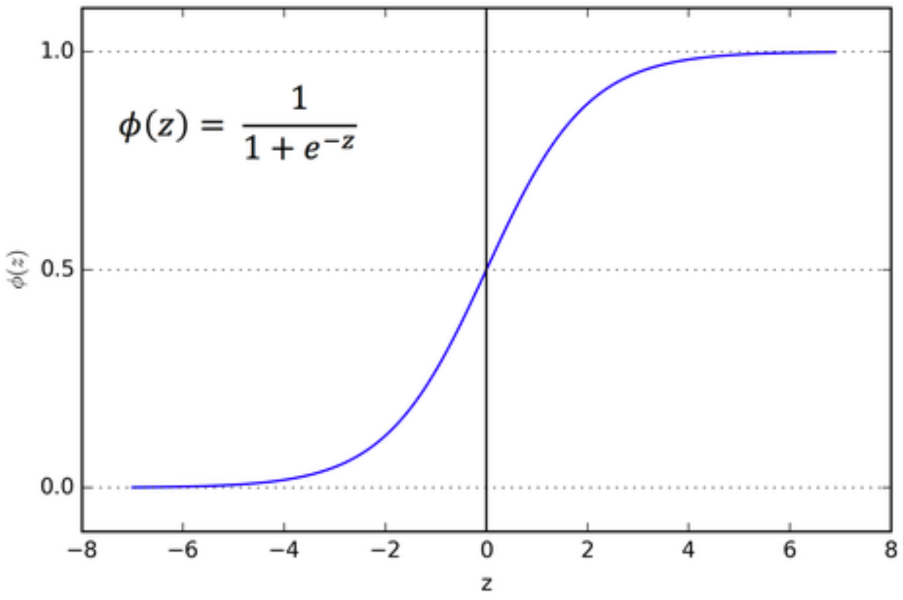
How to get probabilistic decisions?

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

- Sigmoid function

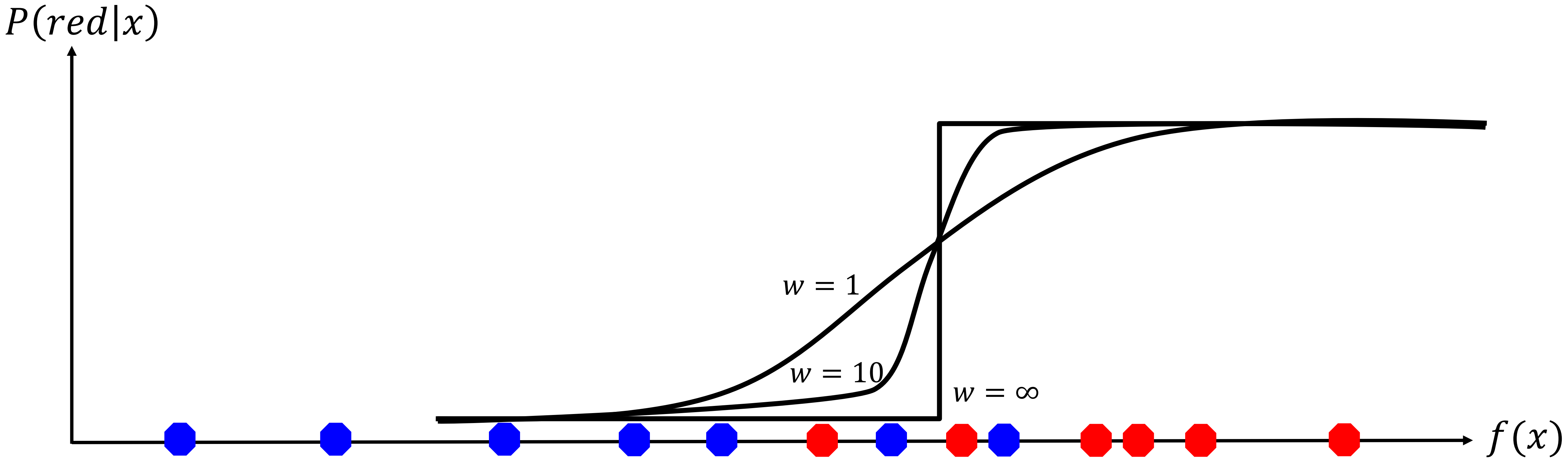
$$\phi(z) = \frac{1}{1 + e^{-z}}$$

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

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

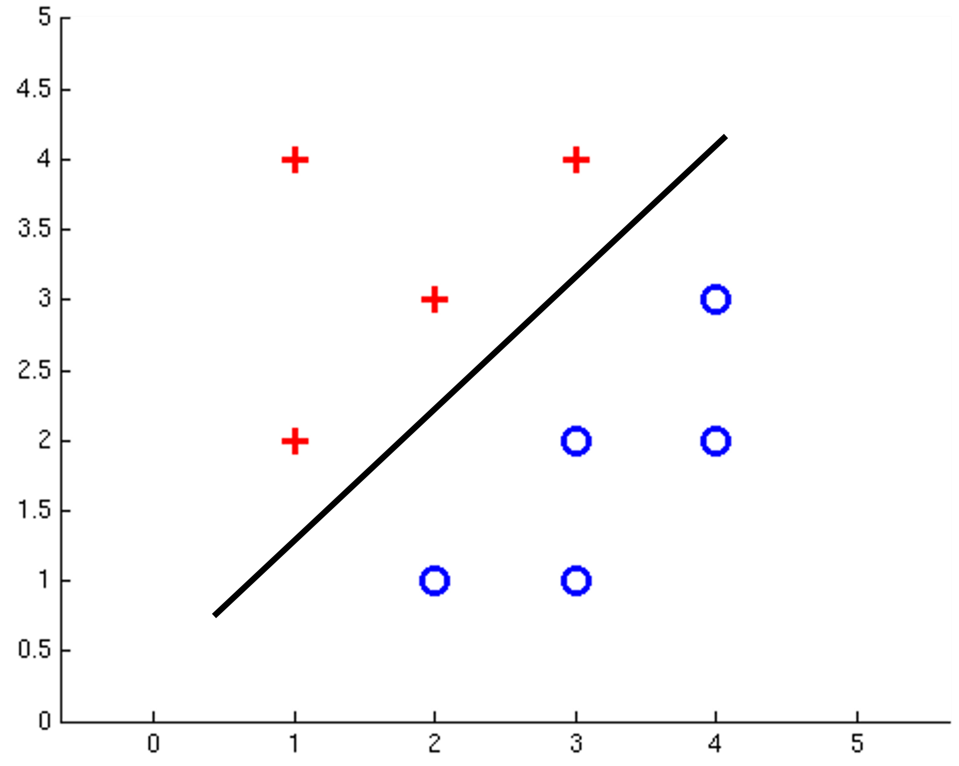
= Logistic Regression

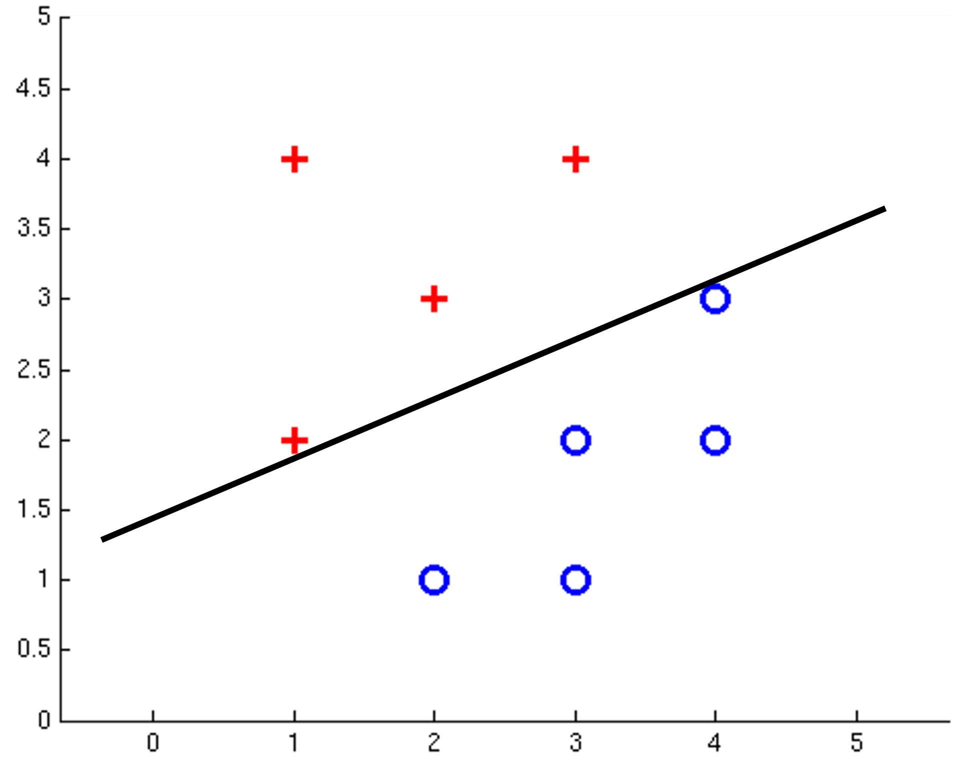
A 1D Example: varying *w*



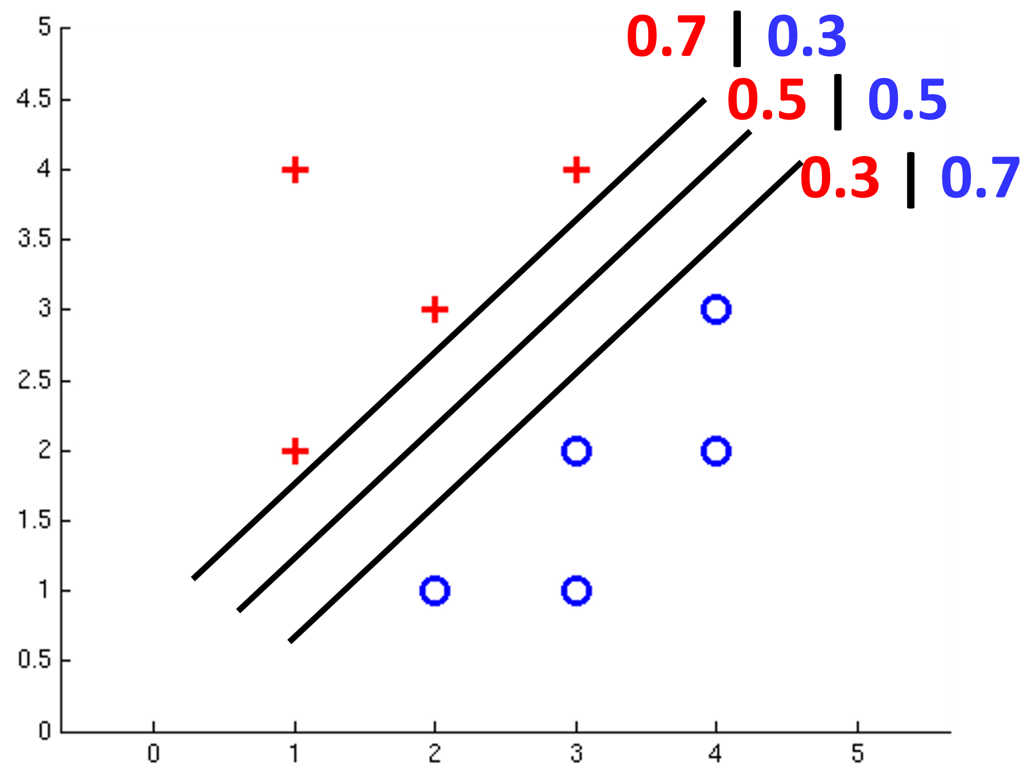
$$P(red|x\,;w) = \phi(w \cdot f(x)) = \frac{1}{1 + e^{-w \cdot f(x)}}$$

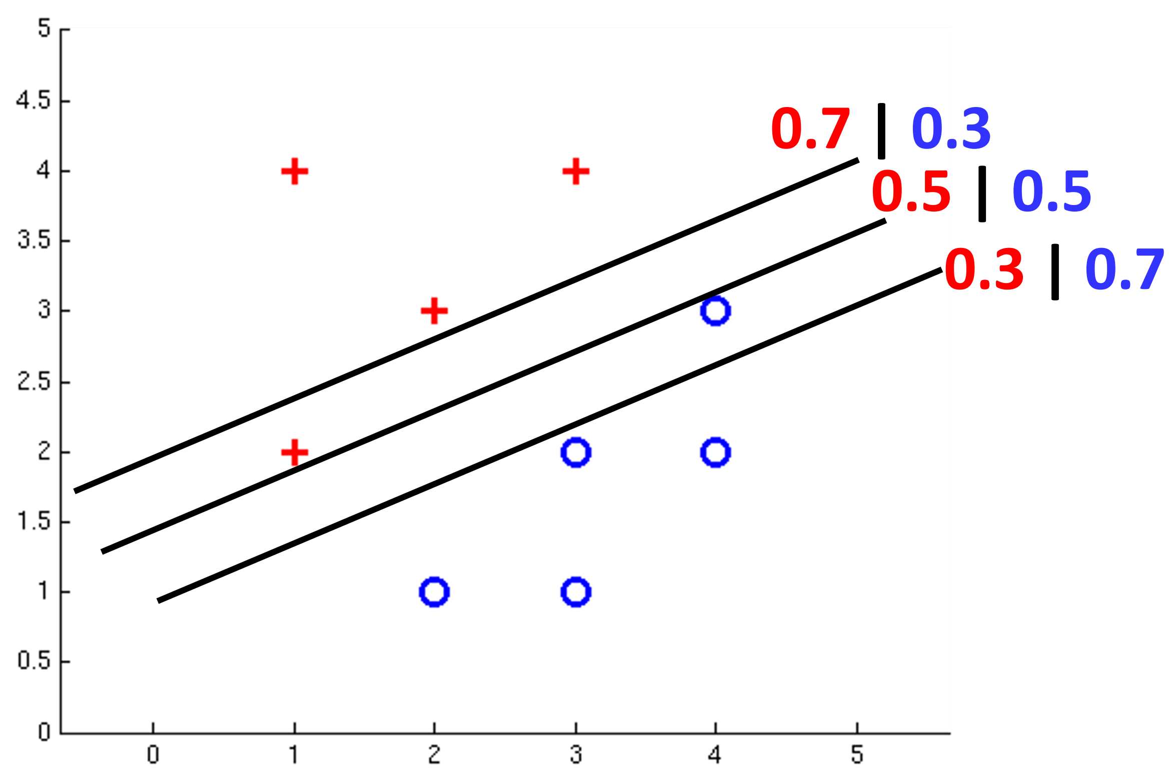
Separable Case: Deterministic Decision – Many Options





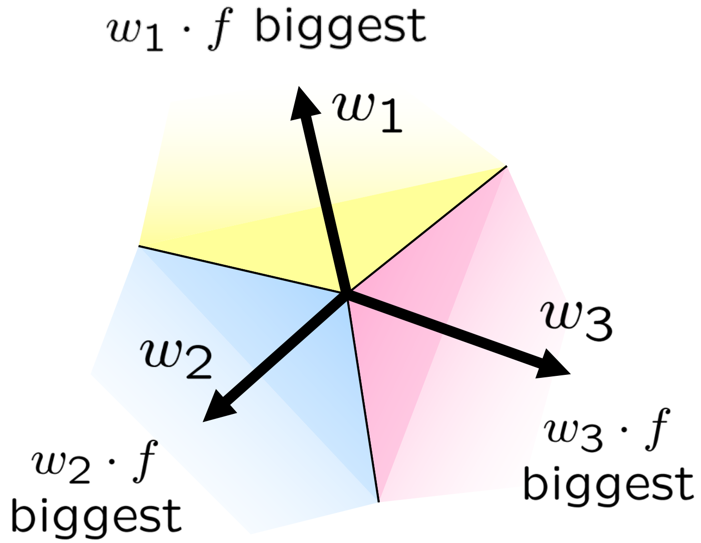
Separable Case: Probabilistic Decision – Clear Preference





Multiclass Logistic Regression

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



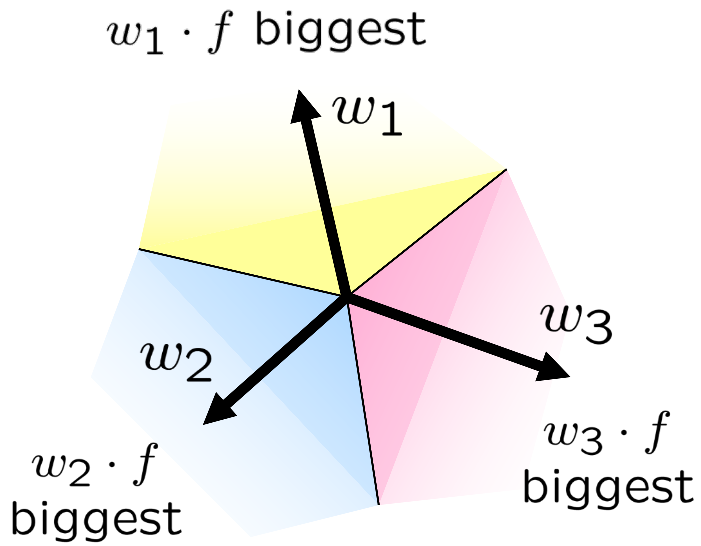
- How to make the scores into probabilities?

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

  - In general: $\text{softmax}(z_1, \ldots, z_n)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$

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?

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

= Multi-Class Logistic Regression

Logistic Regression: Learning

- Have probabilistic model $P(y|x;w)$
- How to find best w?
- **Maximum likelihood estimation:** find w that maximizes $\mathrm{P}(\mathrm{D}|w)$
  - **Dataset:** input-output pairs $\mathrm{x}^{(i)}, y^{(i)}$ that are indep. and identically distributed (i.i.d)

$$P(D|w) = \prod_i P(x^{(i)}, y^{(i)}|w) = \prod_i P(y^{(i)}|x^{(i)}; w)$$

$$P\left(x^{(i)}, y^{(i)}\middle|w\right) = P(y^{(i)}|x^{(i)}; w) \cdot P(x^{(i)}|w)$$

Assume $P(x^{(i)}|w)$ is uniform

  - Optimization problem:

$$\hat{w} = \underset{w}{\operatorname{argmax}}\, P(D|w) = \underset{w}{\operatorname{argmax}} \log P(D|w) = \underset{w}{\operatorname{argmax}} \textstyle\sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Best *w* for **Logistic Regression**

- Given data pairs $x^{(i)}, y^{(i)}$ maximize log-likelihood:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

with:

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

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

Best *w* for **Multi-Class Logistic Regression**

- Given data pairs $\mathrm{x}^{(i)}, y^{(i)}$ maximize log-likelihood:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

with:

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

How do we maximize functions?

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

In general, cannot always take derivative and set to 0

Use numerical optimization!

Hill Climbing

Recall from CSPs lecture: simple, general idea

Start wherever

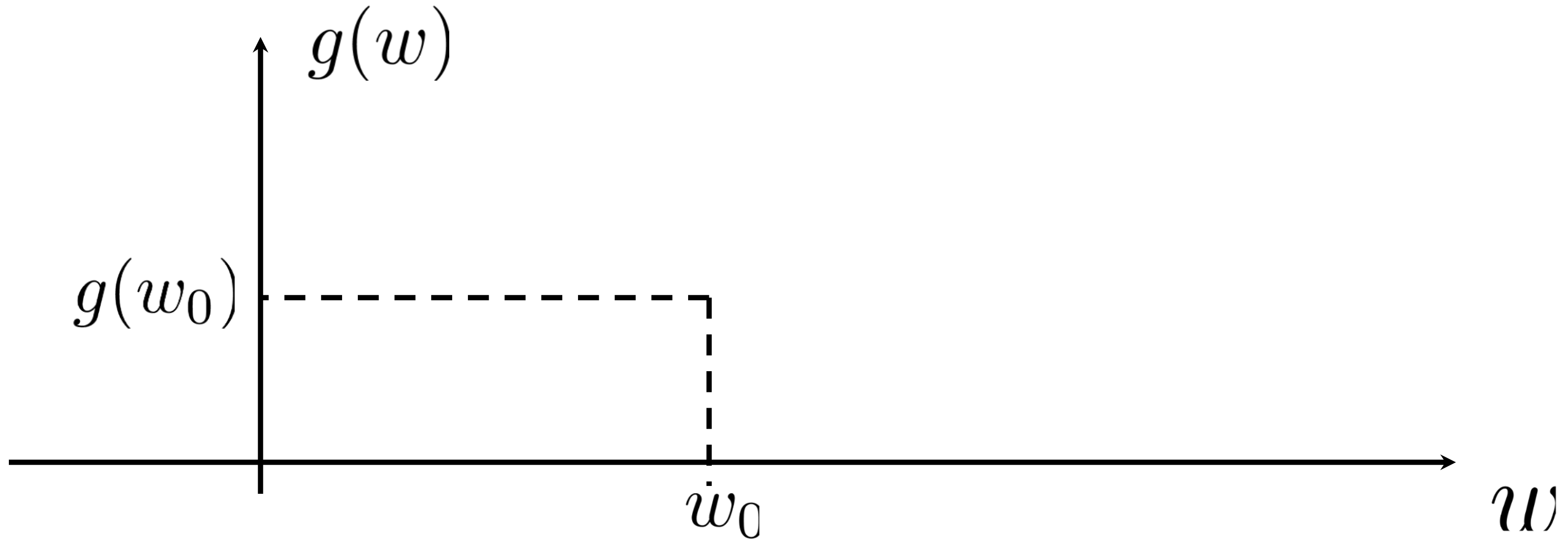
Repeat: move to the best neighboring state

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?

1-D Optimization



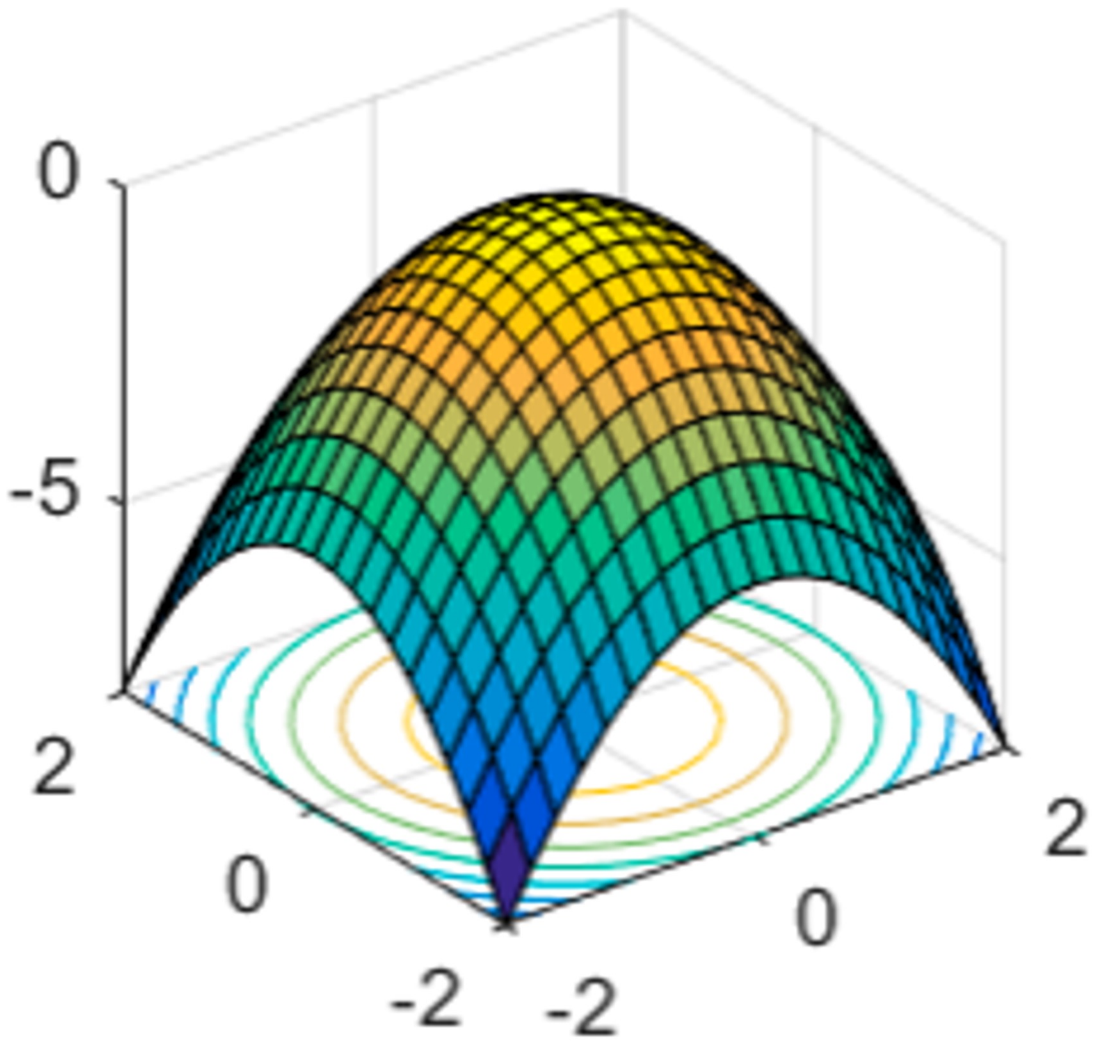
Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$

Then step in best direction

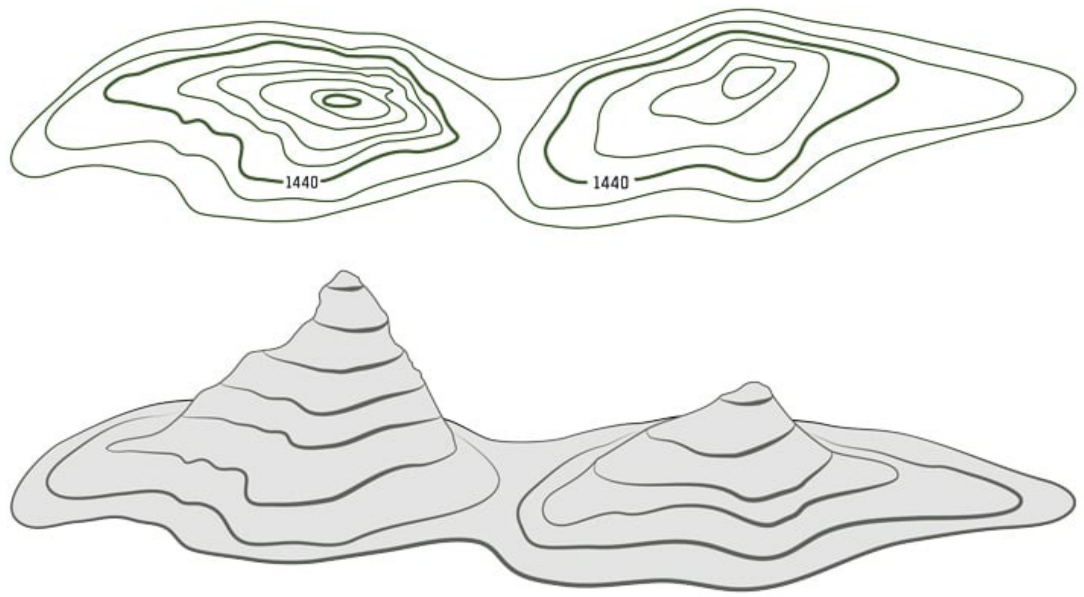
Or, evaluate derivative: $\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$

Tells which direction to step into

2-D Optimization



Source: offconvex.org



Source: REI

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

Idea:

Start somewhere

Repeat: Take a step in the gradient direction

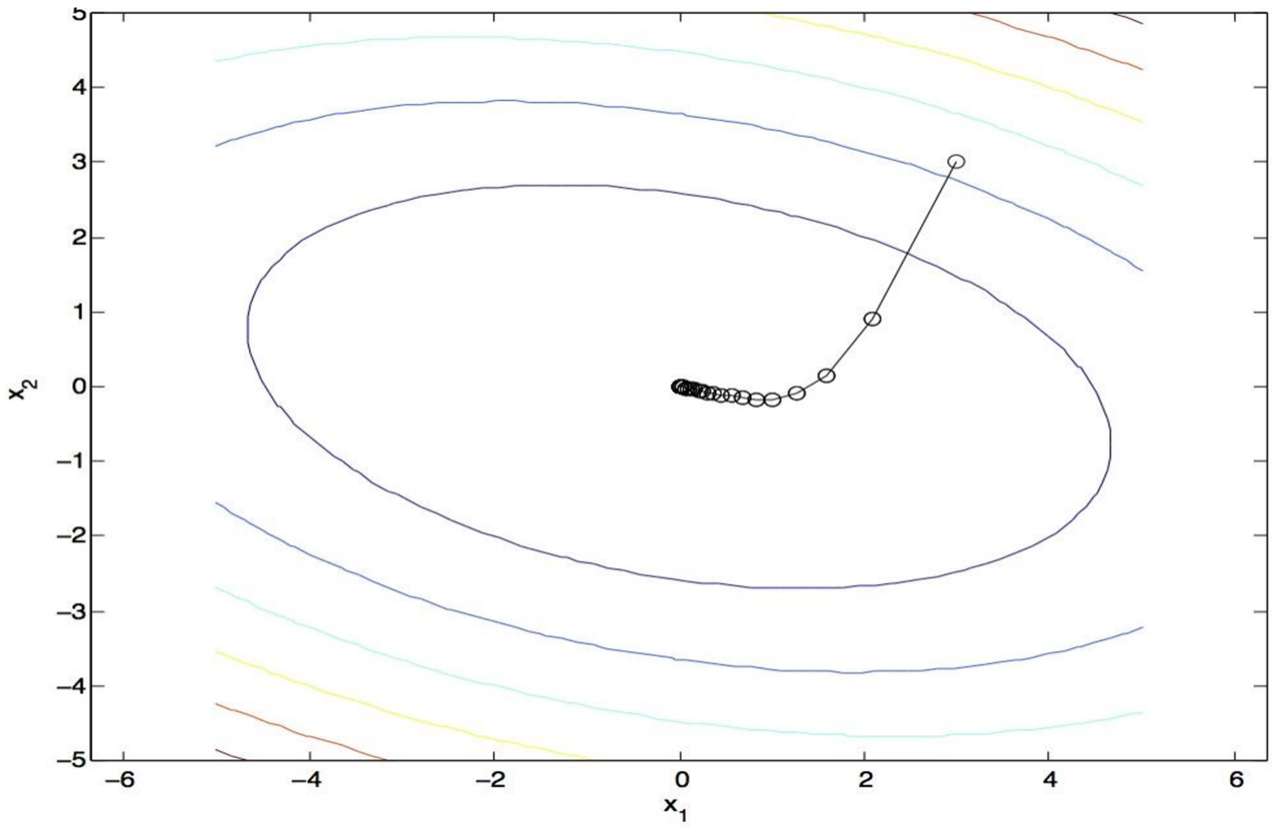


Figure source: Mathworks

Gradient in n dimensions

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

Optimization Procedure: Gradient Ascent

```
init $w$
for iter = 1, 2, …
        $w \leftarrow w + \alpha * \nabla g(w)$
```

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

Batch Gradient Ascent on the Log Likelihood Objective

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

```
init $w$
for iter = 1, 2, …
    $w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)}; w)$
```

Stochastic Gradient Ascent on the Log Likelihood Objective

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

Observation: once gradient on one training example has been computed, might as well incorporate 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

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

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

What will gradient ascent do in multi-class logistic regression?

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

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

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

adds f to the correct class weights

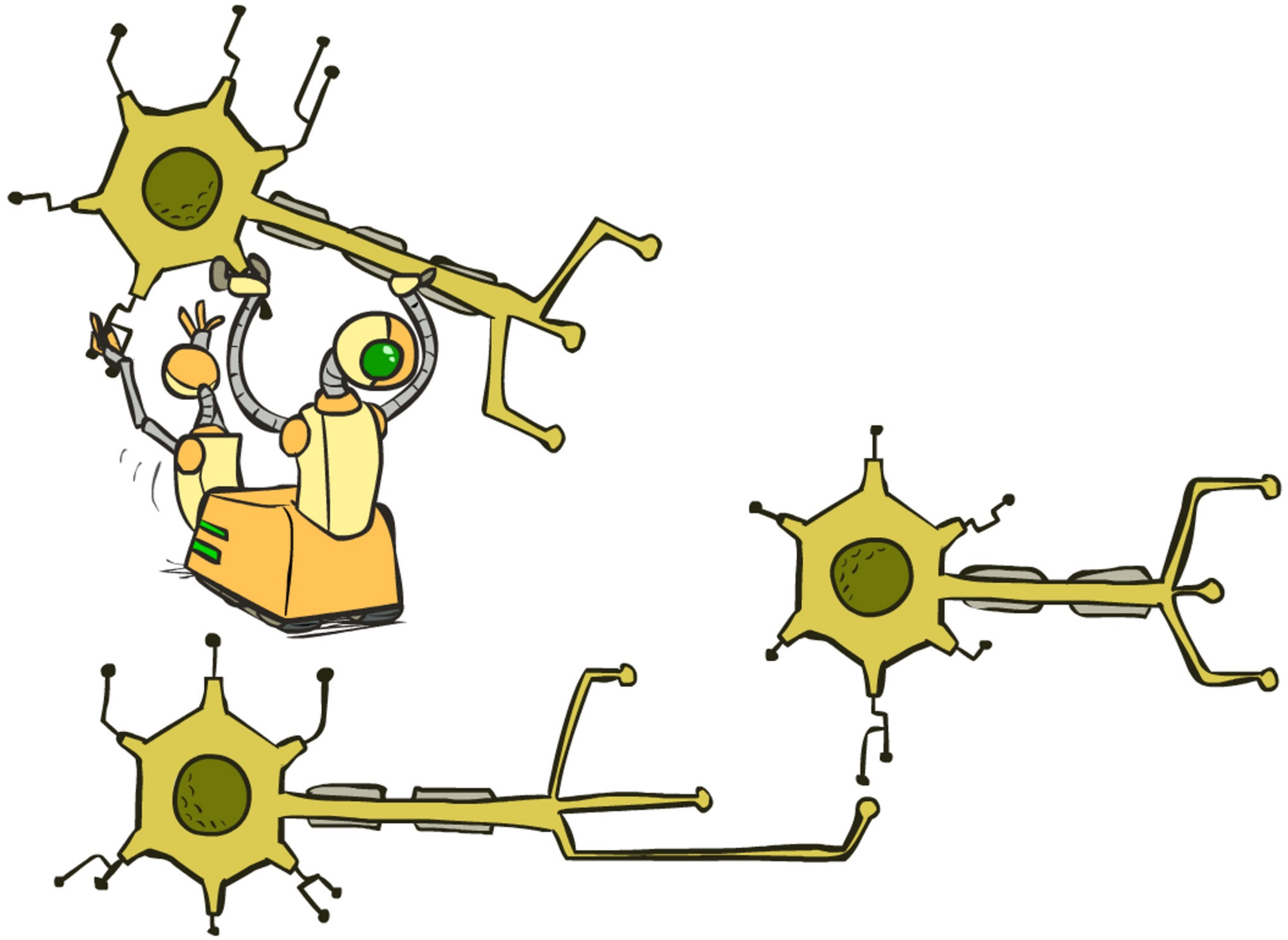
$$\frac{1}{\sum_y e^{w_y f(x^{(i)})}} \sum_y \left(e^{w_y f(x^{(i)})} [0^T f(x^{(i)})^T 0^T]^T \right)$$

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

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

subtracts f from y' weights in proportion to the probability current weights give to y'

Next Week: Neural Networks



What is the Steepest Direction?*

$$\max_{\Delta:\Delta_1^2+\Delta_2^2\leq\varepsilon} \quad g(w+\Delta)$$

First-Order Taylor Expansion: $$g(w+\Delta) \approx g(w) + \frac{\partial g}{\partial w_1}\Delta_1 + \frac{\partial g}{\partial w_2}\Delta_2$$

Steepest Ascent Direction: $$\max_{\Delta:\Delta_1^2+\Delta_2^2\leq\varepsilon} \quad g(w) + \frac{\partial g}{\partial w_1}\Delta_1 + \frac{\partial g}{\partial w_2}\Delta_2$$

Recall: $\max_{\Delta:\|\Delta\|\leq\varepsilon} \Delta^\top a$ ⍰ $\Delta = \varepsilon\frac{a}{\|a\|}$

Hence, solution: $\Delta = \varepsilon\frac{\nabla g}{\|\nabla g\|}$ **Gradient direction = steepest direction!** $\nabla g = \begin{bmatrix}\frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2}\end{bmatrix}$