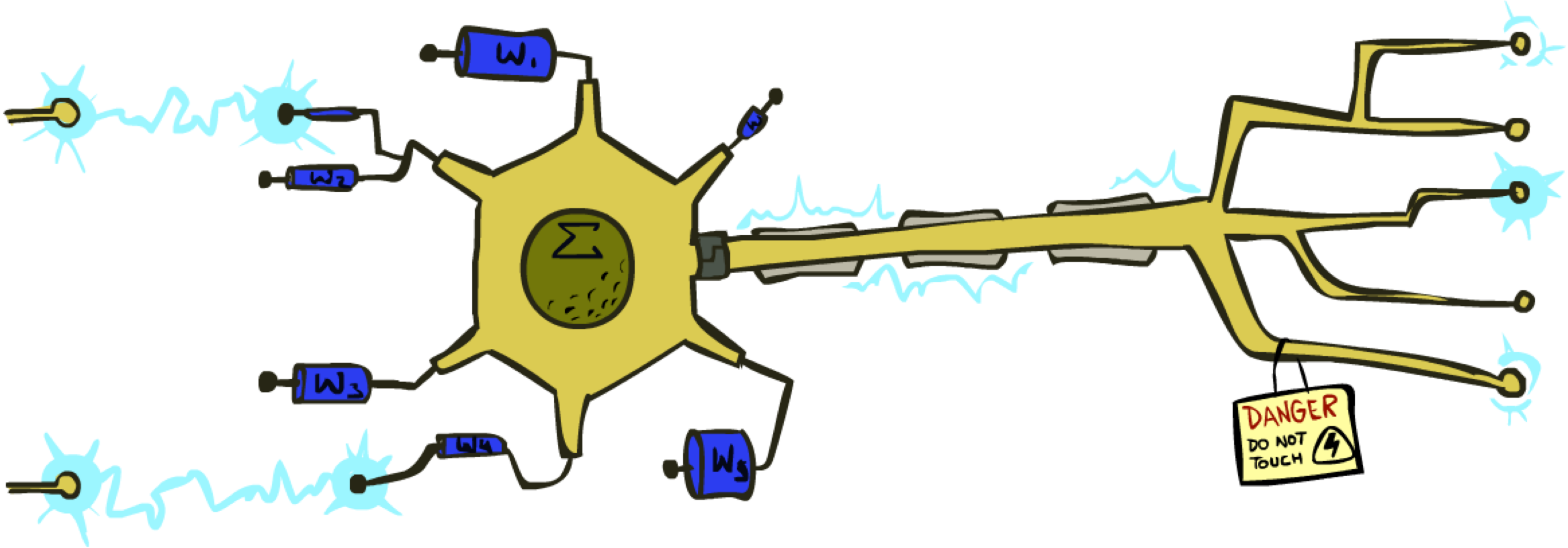
CS 188: Artificial Intelligence

Perceptrons, Linear / Logistic Regression

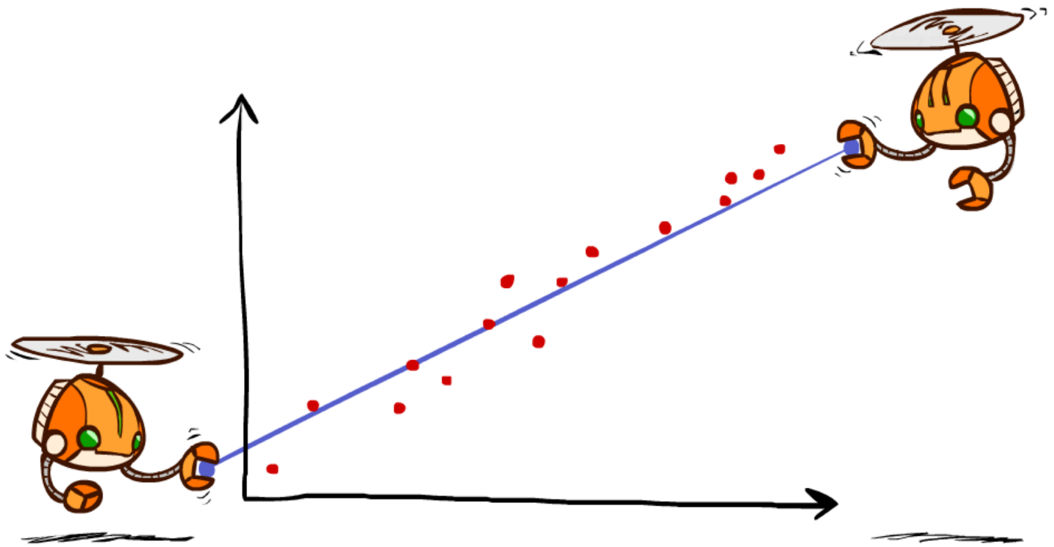


Instructors: Angela Liu and Yanlai Yang

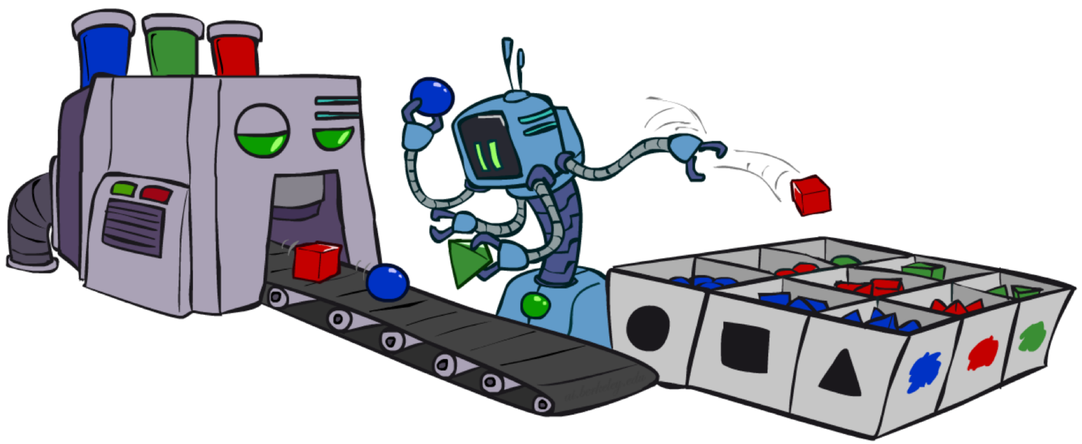
University of California, Berkeley

(Slides adapted from Pieter Abbeel, Dan Klein, Anca Dragan, Stuart Russell and Dawn Song)

Supervised Learning

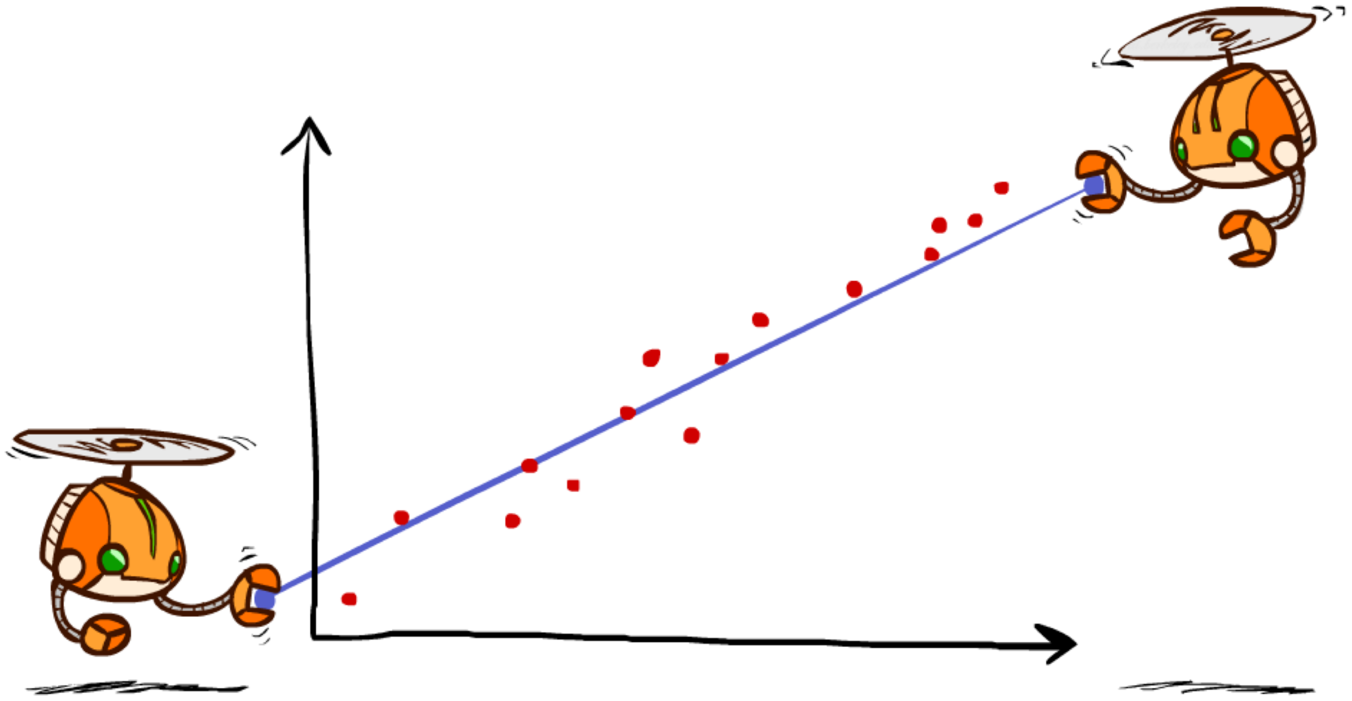


Regression
learning a function with
real-valued output value



Classification
learning a function with
discrete output value

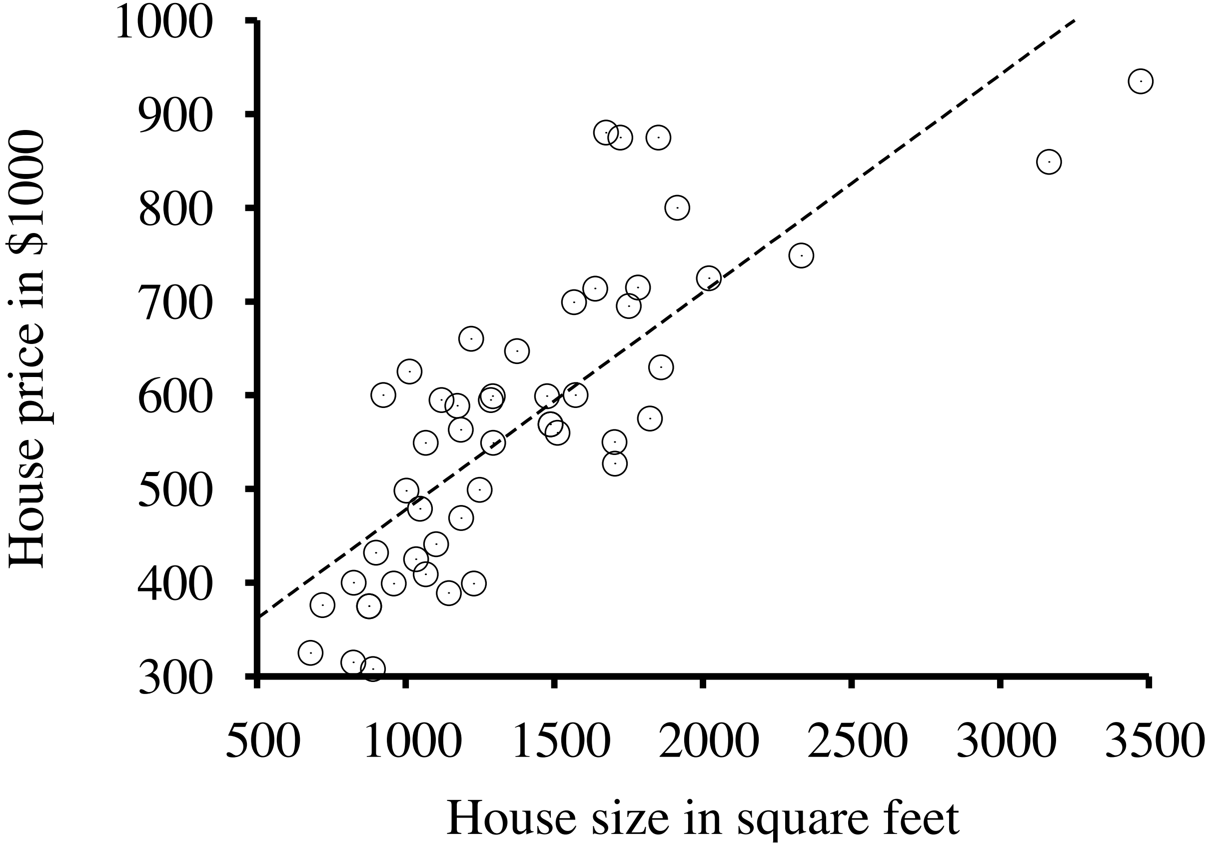
Linear Regression



Model: Linear functions

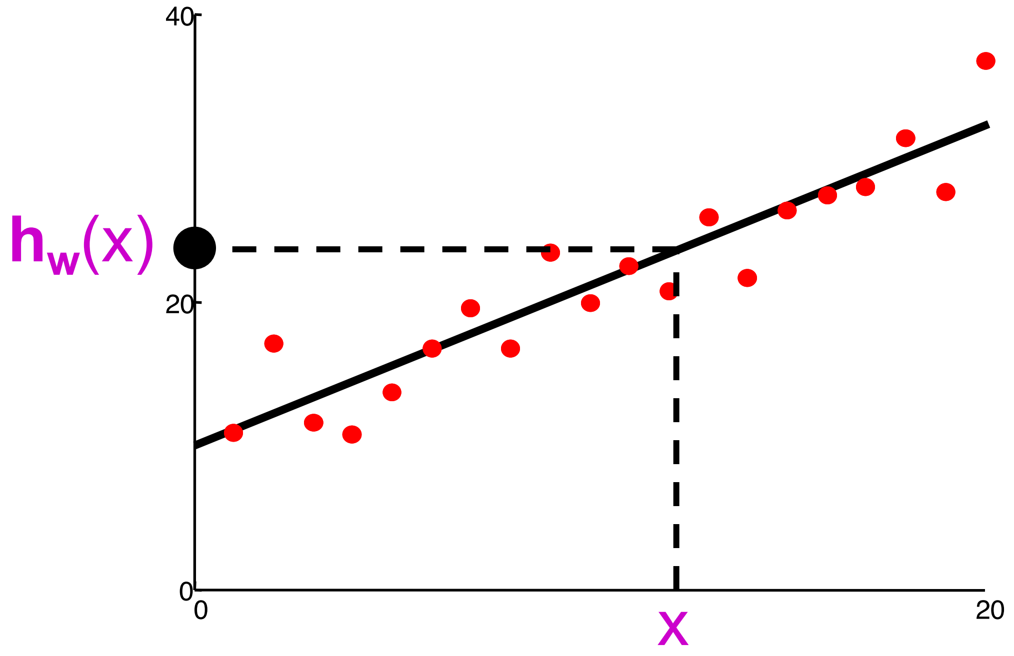
Linear Regression

(x, y), x: house size, y: house price

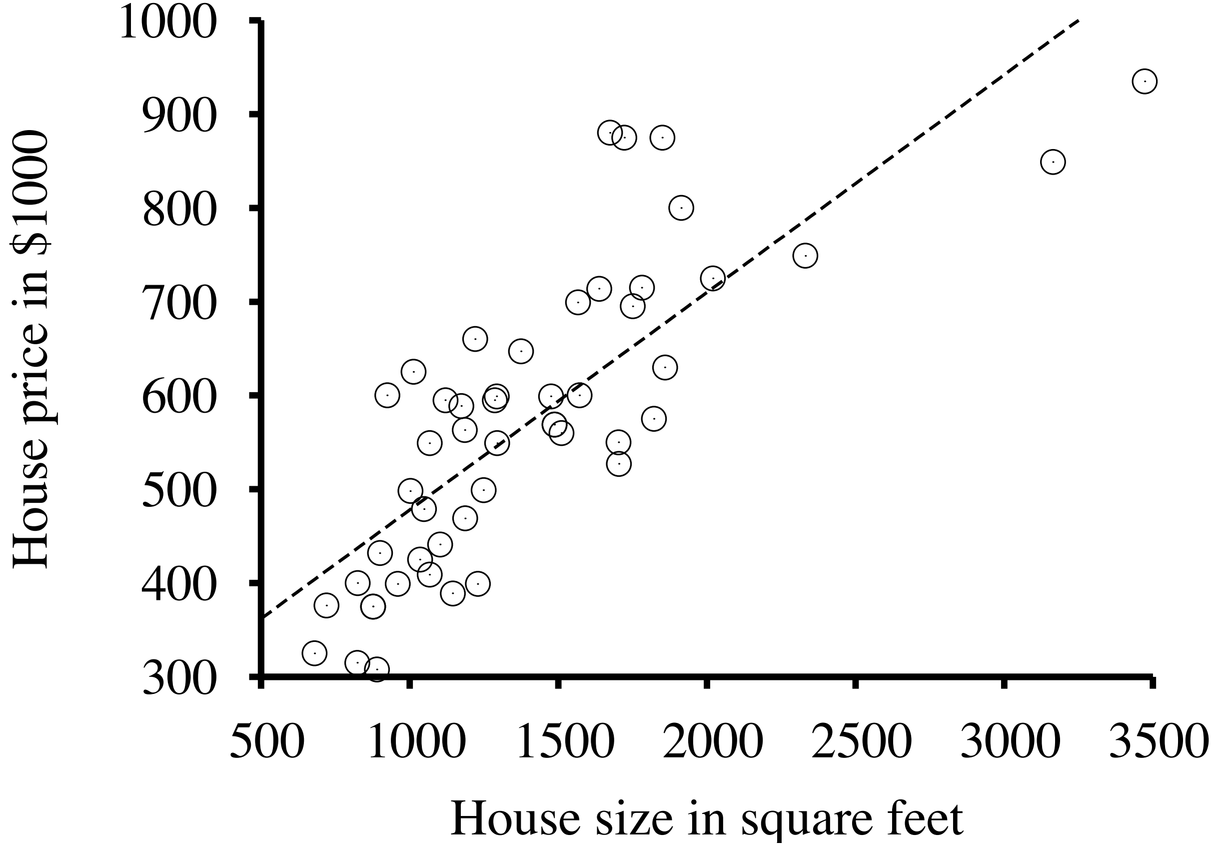


Berkeley house prices, 2009

Linear regression = fitting a straight line/hyperplane



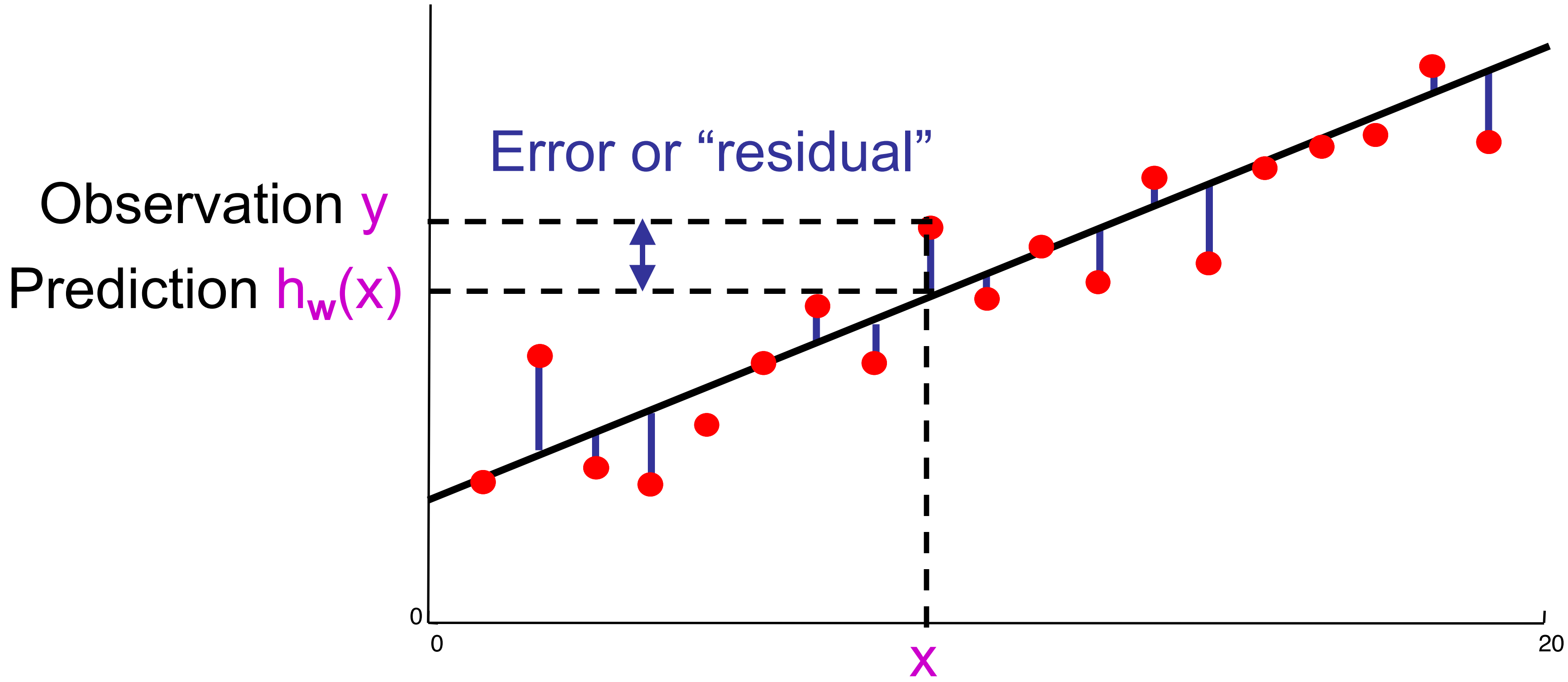
Prediction: $h_w(x) = w_0 + w_1 x$



Berkeley house prices, 2009

Prediction error

Error on one instance: $y - h_w(x)$



Find w

- Define loss function

- Find w* to minimize loss function

Least squares: Minimizing squared error

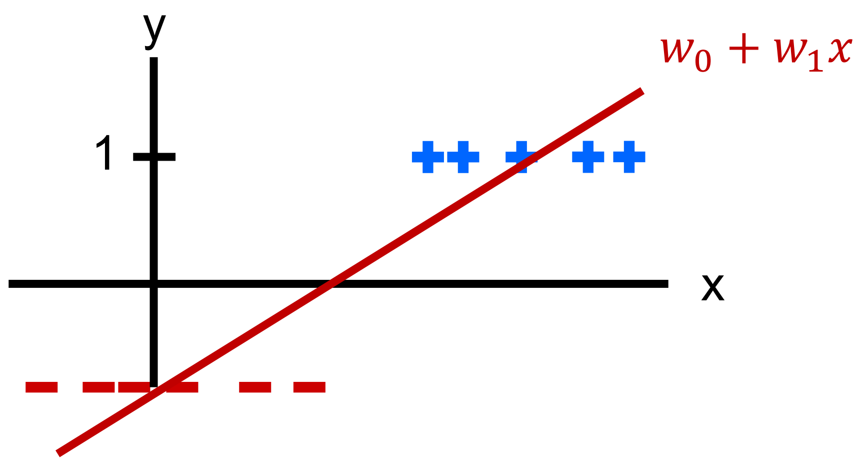
- L2 loss function: sum of squared errors over all examples
  - Loss = ________________________________
- We want the weights w* that minimize loss
- At w* the derivatives of loss w.r.t. each weight are zero:
  - $\partial \text{Loss} / \partial w_0$ = ______________________________
  - $\partial \text{Loss} / \partial w_1$ = ______________________________
- Exact solutions for N examples:
  - $w_1 = \left[N \sum_j x_j y_j - \left(\sum_j x_j\right)\left(\sum_j y_j\right)\right] / \left[N \sum_j x_j^2 - \left(\sum_j x_j\right)^2\right]$ and $w_0 = \frac{1}{N}\left[\sum_j y_j - w_1 \sum_j x_j\right]$
- For the general case where x is an n-dimensional vector
  - **X** is the data matrix (all the data, one example per row); **y** is the column of labels
  - $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

Least squares: Minimizing squared error

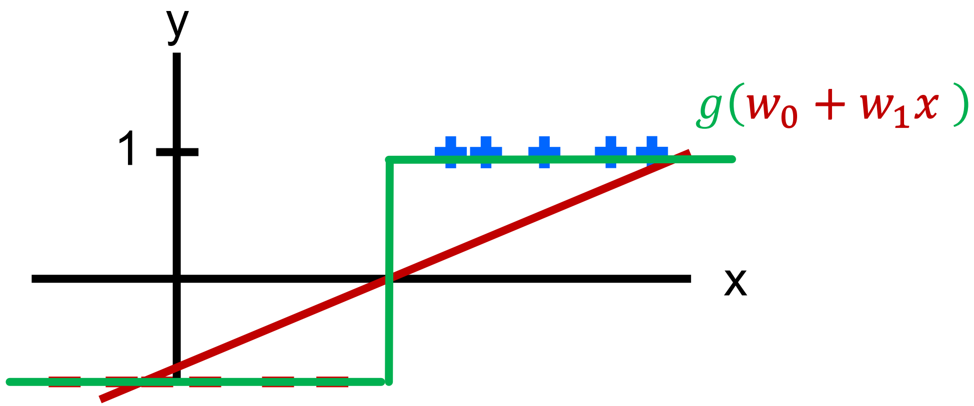
- L2 loss function: sum of squared errors over all examples
  - Loss $= \Sigma_j (y_j - h_{\mathbf{w}}(x_j))^2 = \Sigma_j (y_j - (w_0 + w_1 x_j))^2$
- We want the weights w^* that minimize loss
- At w^* the derivatives of loss w.r.t. each weight are zero:
  - $\partial \text{Loss} / \partial w_0 = -2 \ \Sigma_j (y_j - (w_0 + w_1 x_j)) = 0$
  - $\partial \text{Loss} / \partial w_1 = -2 \ \Sigma_j (y_j - (w_0 + w_1 x_j)) \, x_j = 0$
- Exact solutions for N examples:
  - $w_1 = \left[N \sum_j x_j y_j - (\sum_j x_j)(\sum_j y_j)\right] / \left[N \sum_j x_j^2 - (\sum_j x_j)^2\right]$ and $w_0 = \frac{1}{N}\left[\sum_j y_j - w_1 \sum_j x_j\right]$
- For the general case where x is an n-dimensional vector
  - **X** is the data matrix (all the data, one example per row); **y** is the column of labels
  - $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

Regression vs Classification

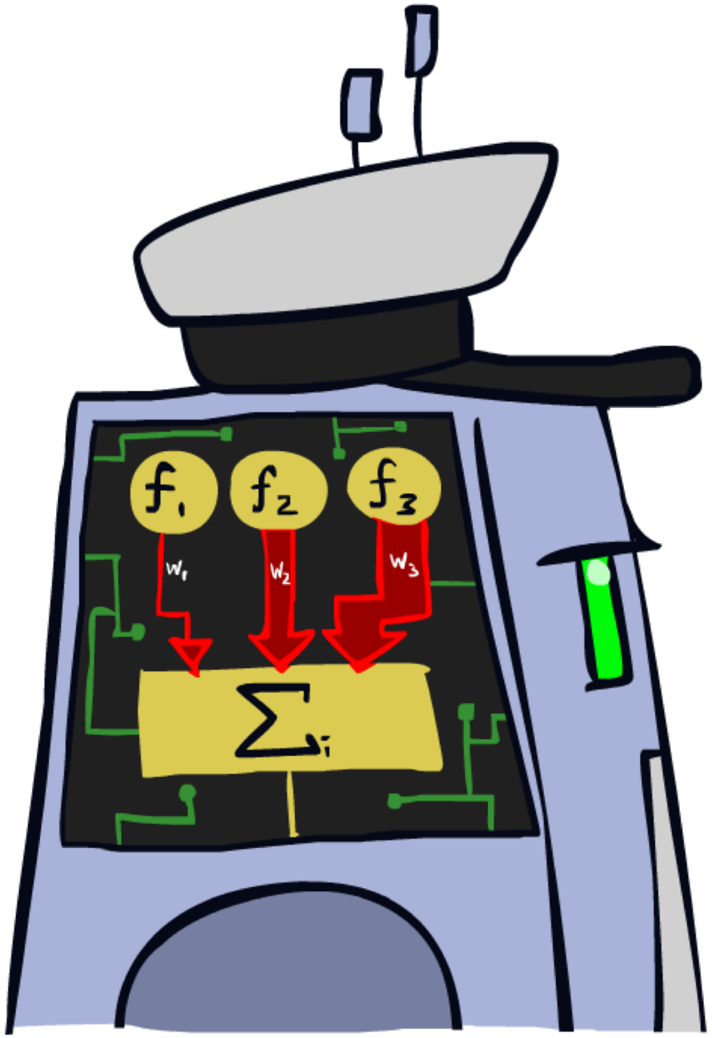
- Linear regression when output is binary, $y \in \{-1, 1\}$
  - $h_w(x) = w_0 + w_1 x$



- Linear classification
  - Used with discrete output values
  - Threshold a linear function
  - $h_w(x) = 1,$ if $w_0 + w_1 x \geq 0$
  - $h_w(x) = -1,$ if $w_0 + w_1 x < 0$



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
...
```

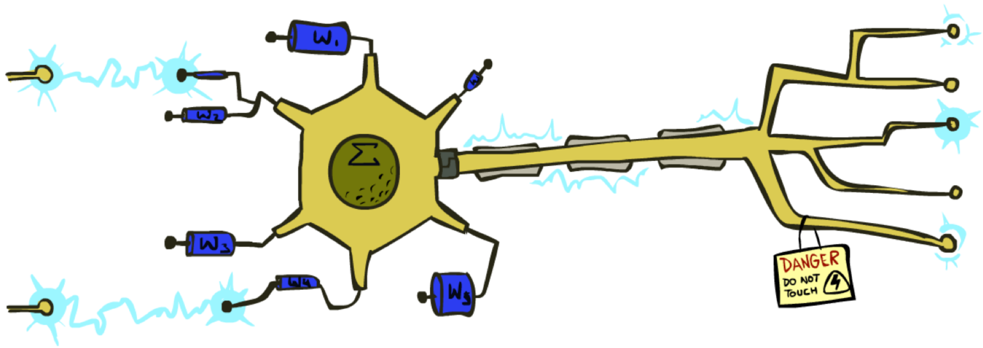
SPAM
or
+

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

“2”

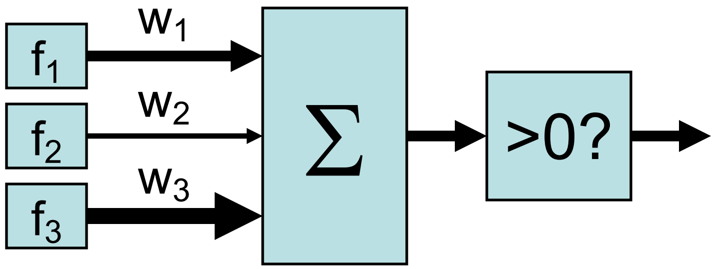
Linear Classifiers

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



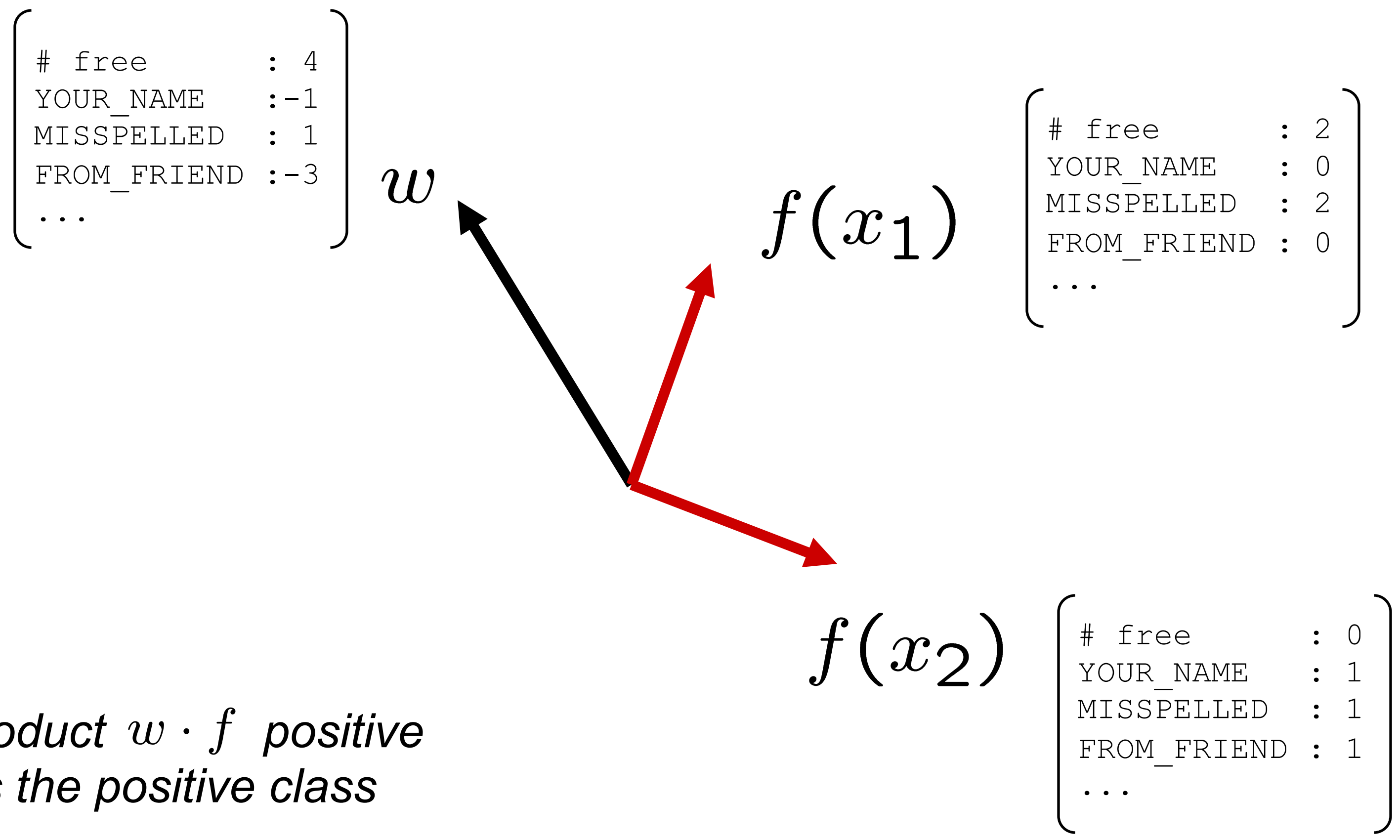
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

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



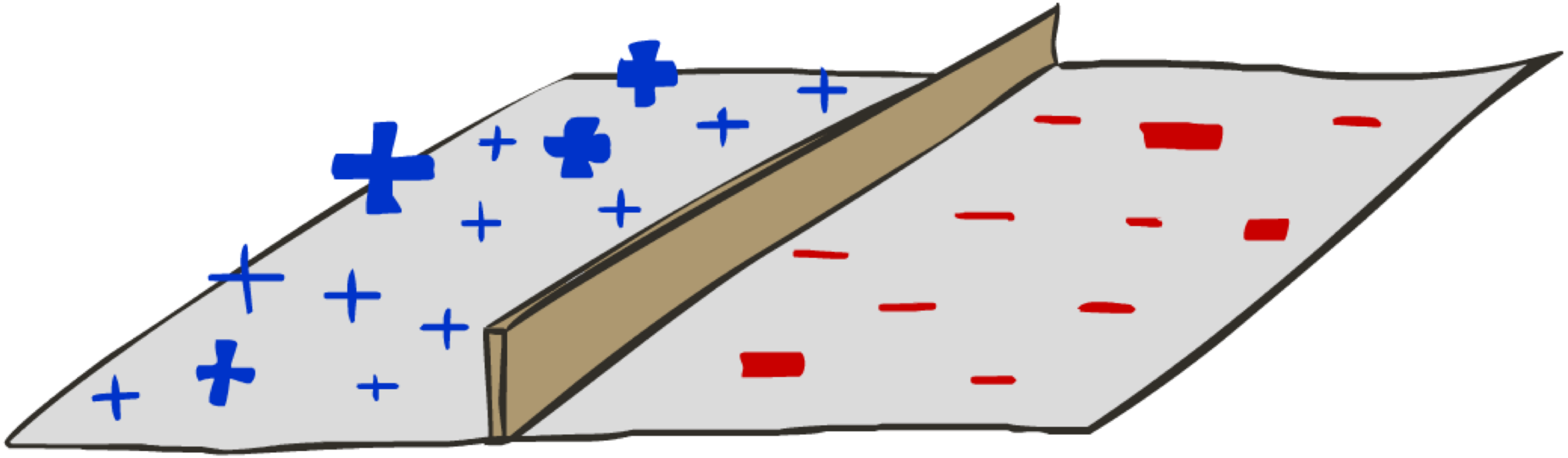
Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples



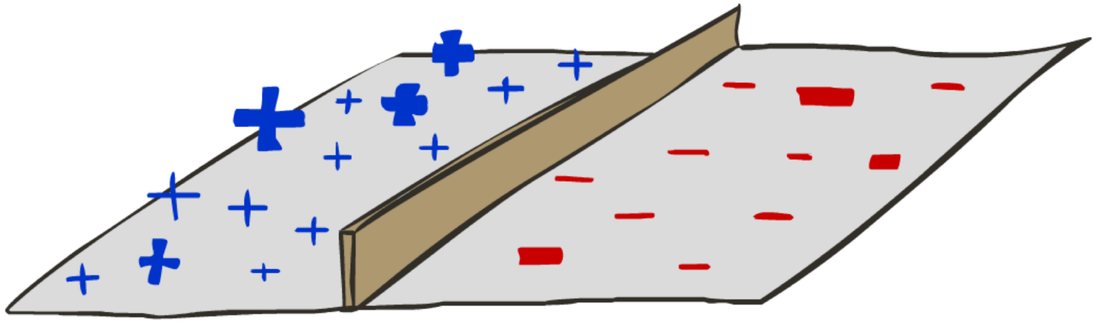
Dot product $w \cdot f$ *positive means the positive class*

Decision Rules



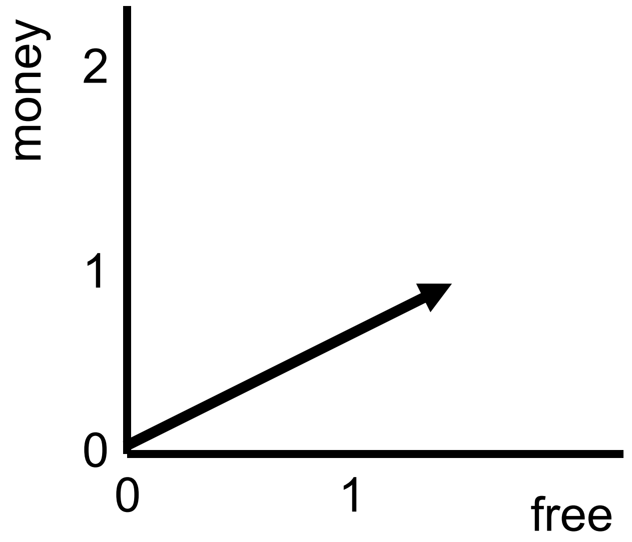
Binary Decision Rule

- 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



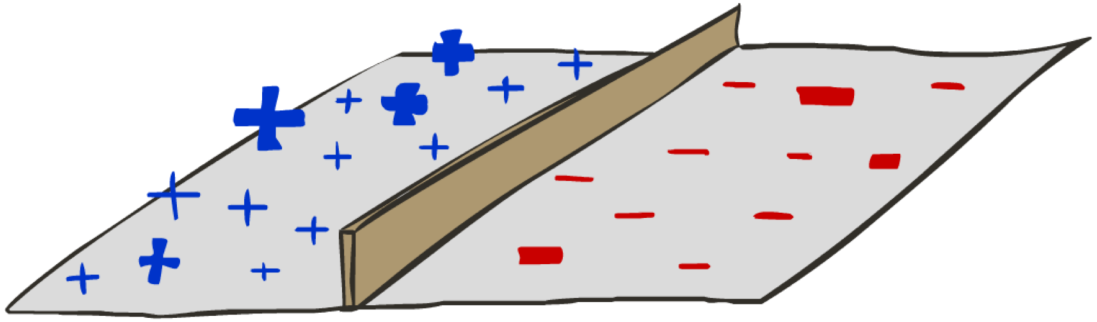
w

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



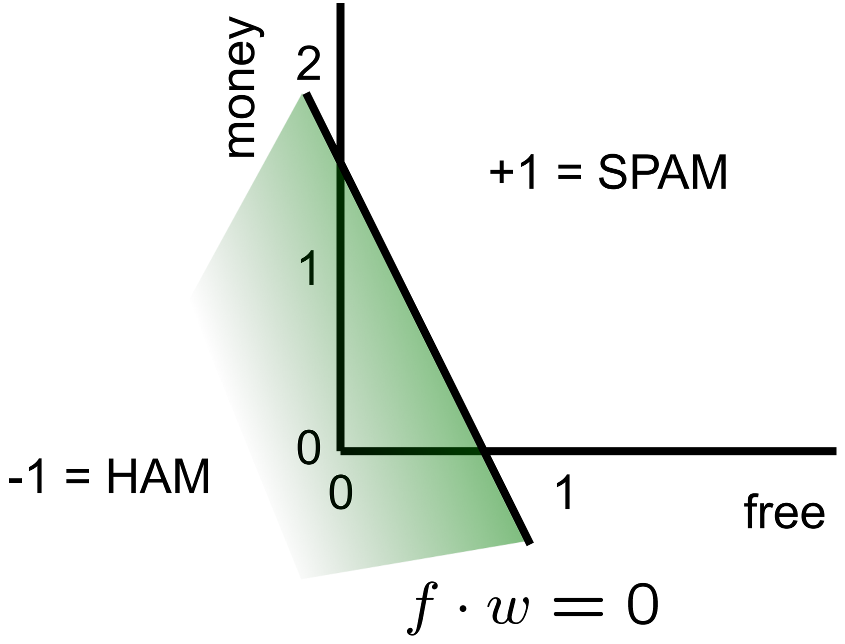
Binary Decision Rule

- 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

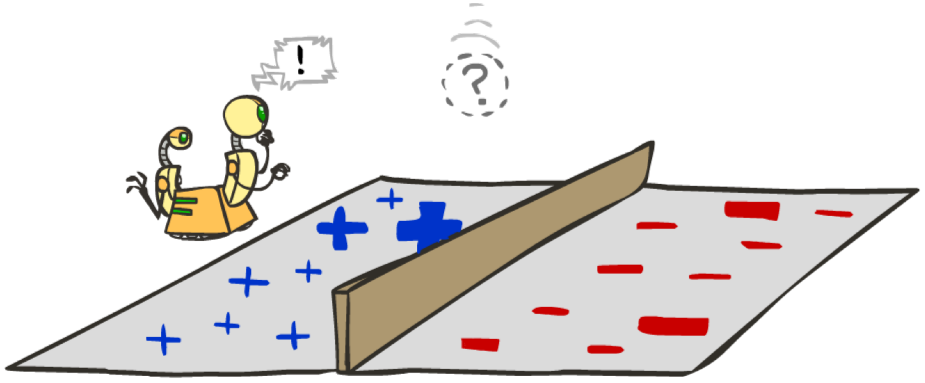
```
BIAS  : -3
free  :  4
money :  2
...
```



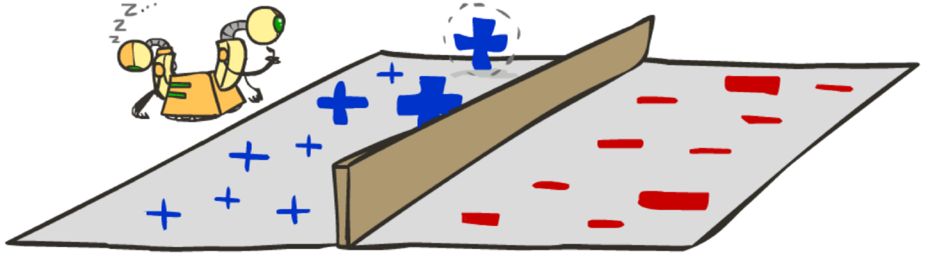
Weight Updates

Learning: Binary Perceptron

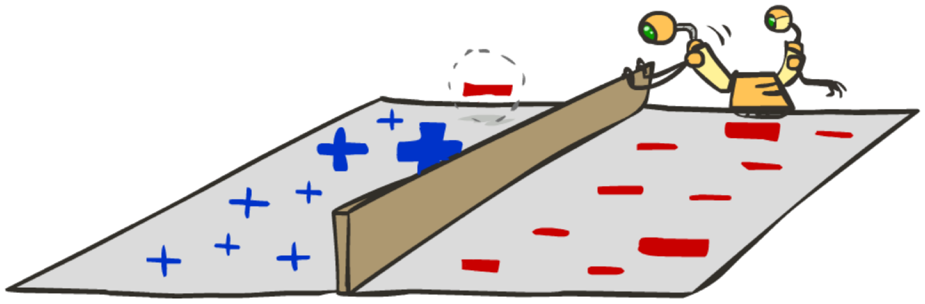
- Start with weights = 0
- For each training instance:
  - Classify with current weights



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



  - If wrong: adjust the weight vector



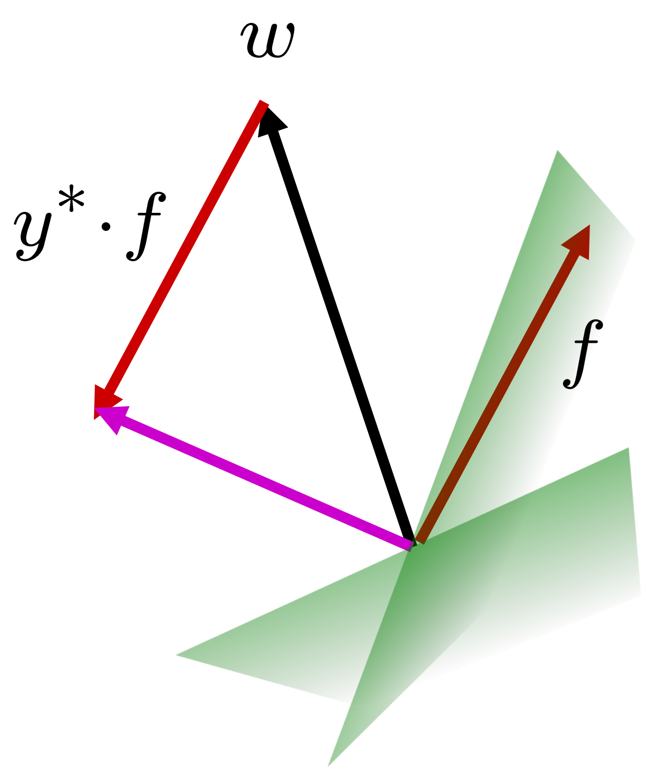
Learning: Binary Perceptron

- 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.

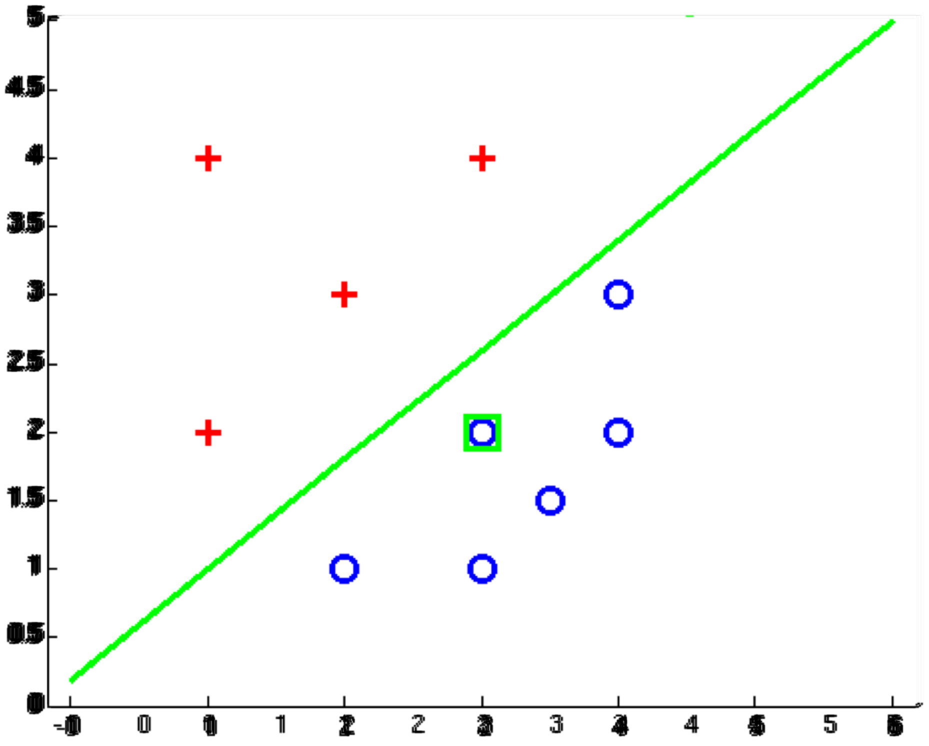
$$w = w + y^* \cdot f$$



Before: $w\,f$
After: $wf + y^* f f$
$f f$ >=0

Examples: Perceptron

- Separable Case



Multiclass Decision Rule

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

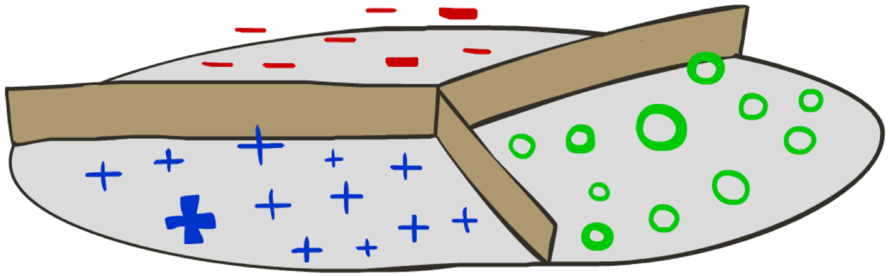
$$w_y$$

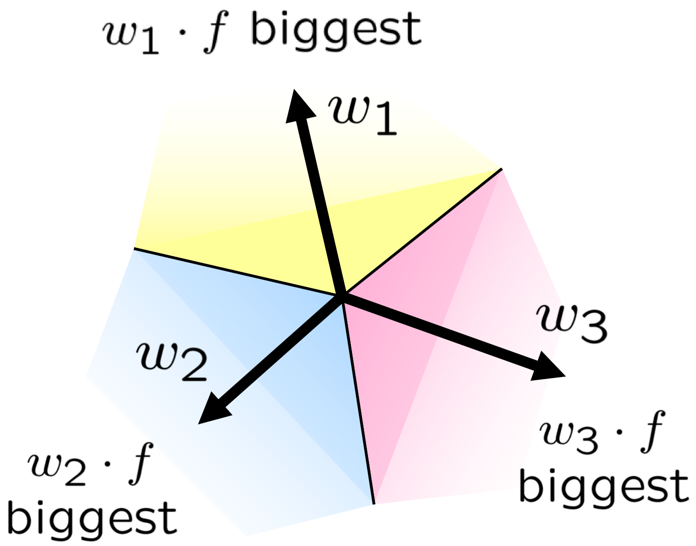
  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - Prediction highest score wins

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





Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

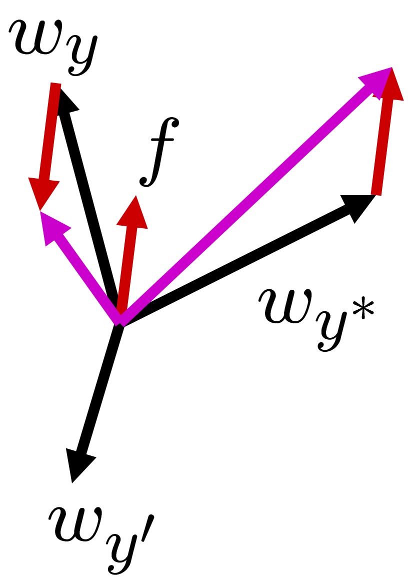
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

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

- 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

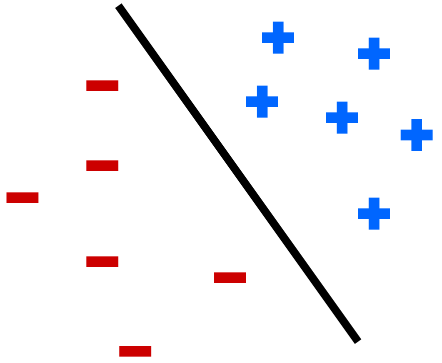
BIAS	: 0
win	: 0
game	: 0
vote	: 0
the	: 0
...	

Properties of Perceptrons

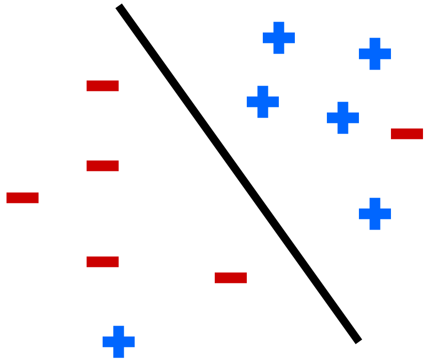
- Separability: true if some parameters get the training set perfectly correct

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

Separable

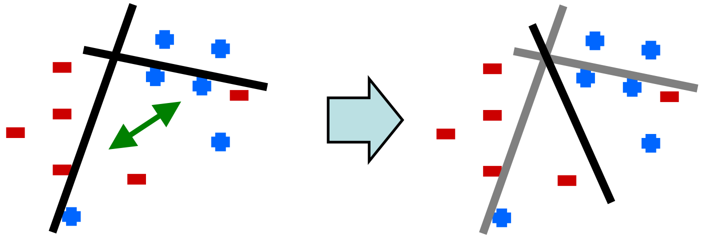


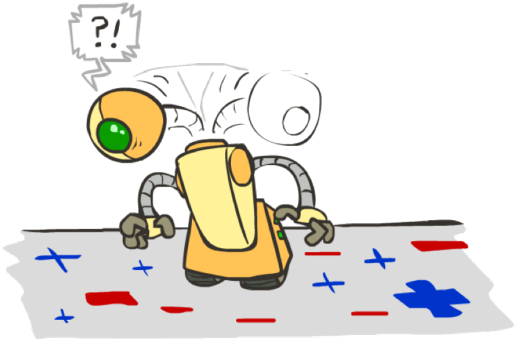
Non-Separable



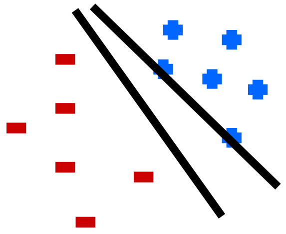
Problems with the Perceptron

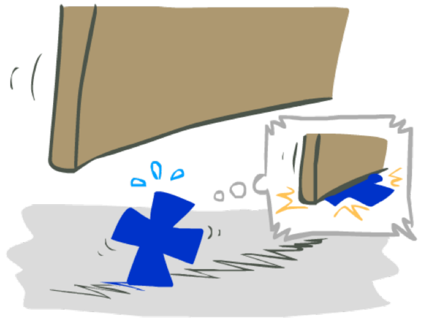
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



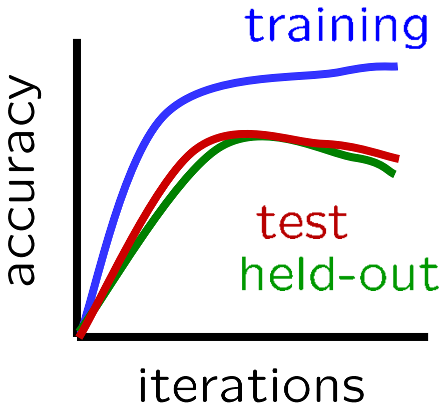


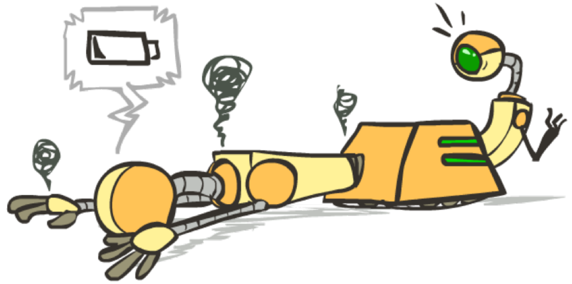
- Mediocre generalization: finds a "barely" separating solution



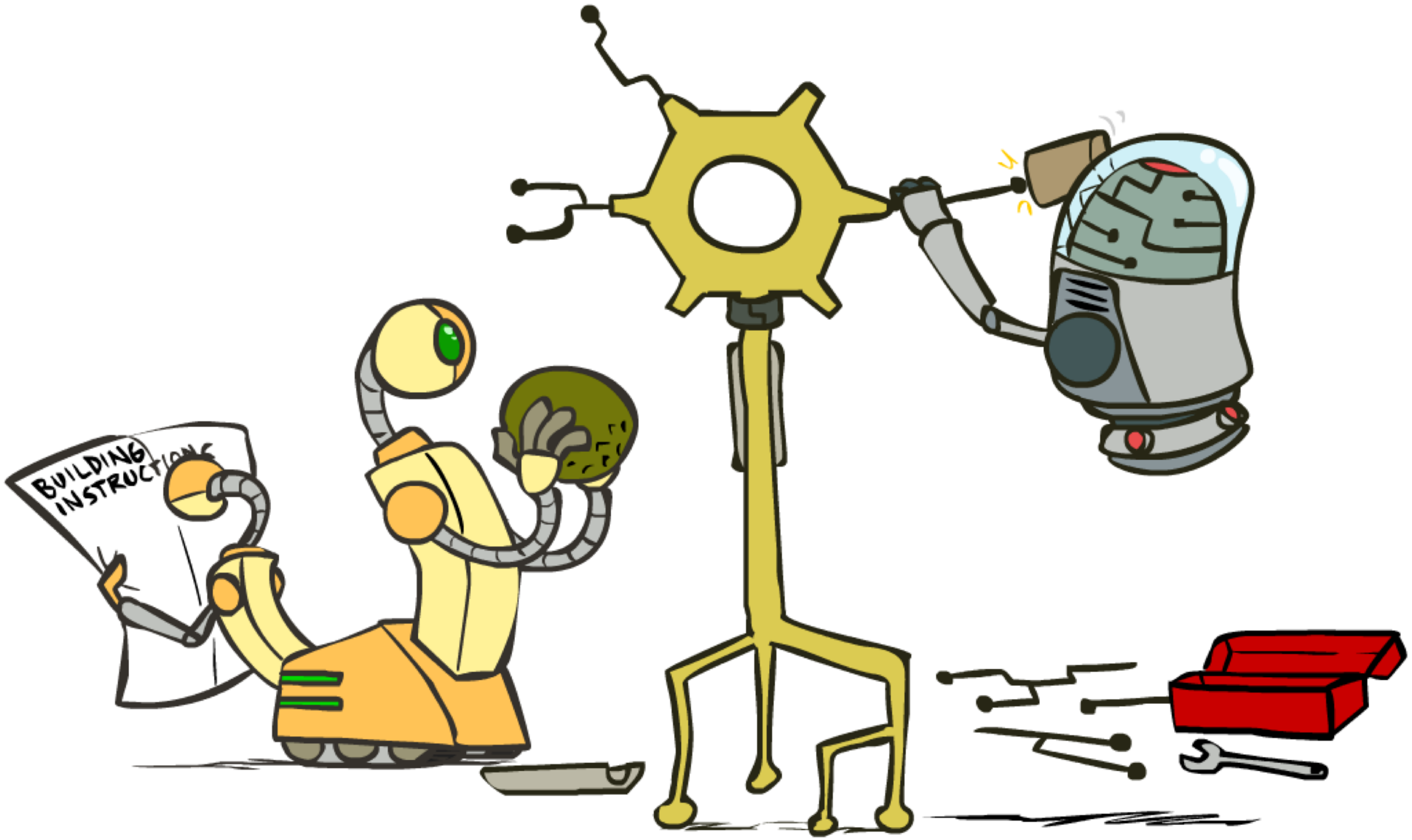


- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



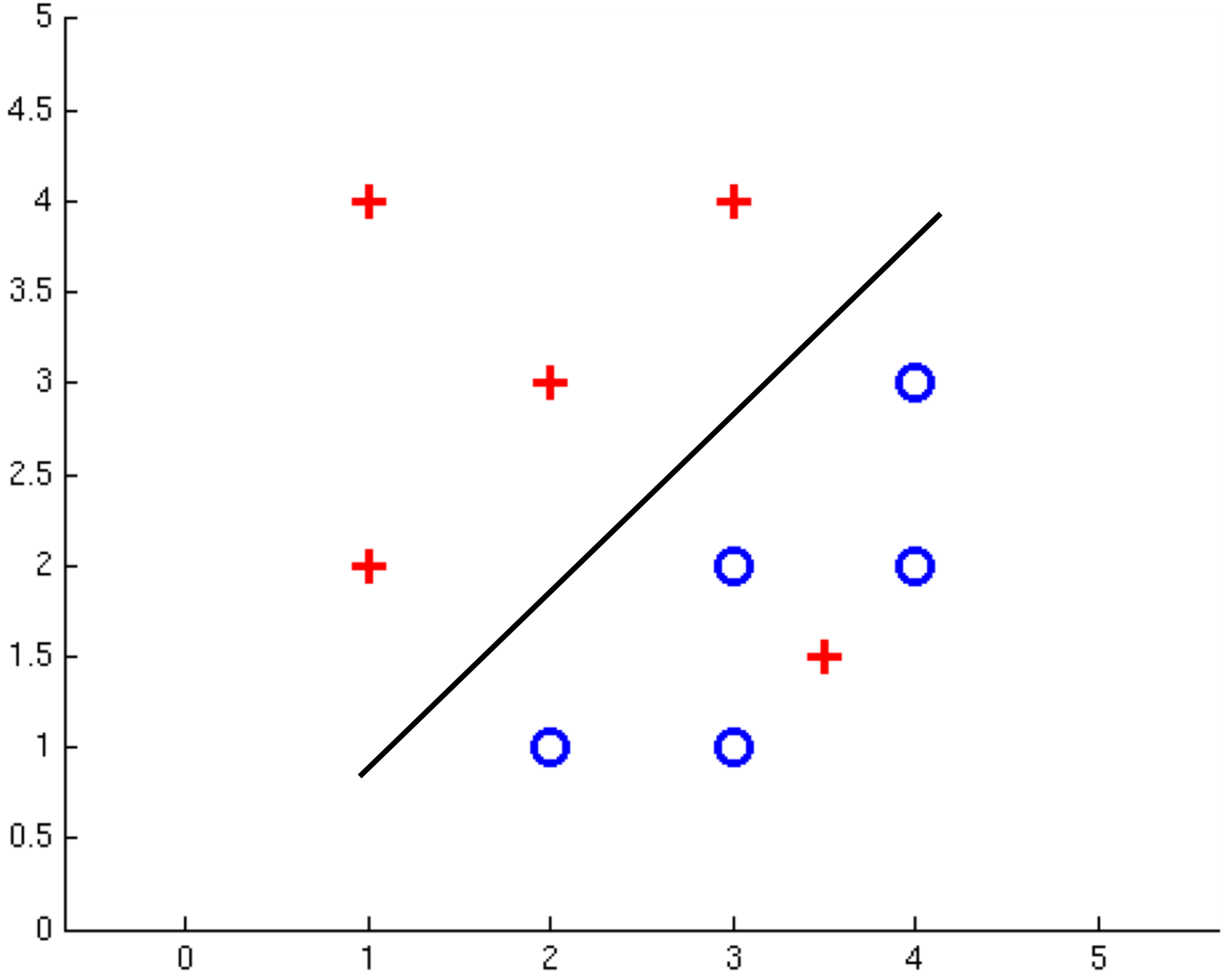


Improving the Perceptron

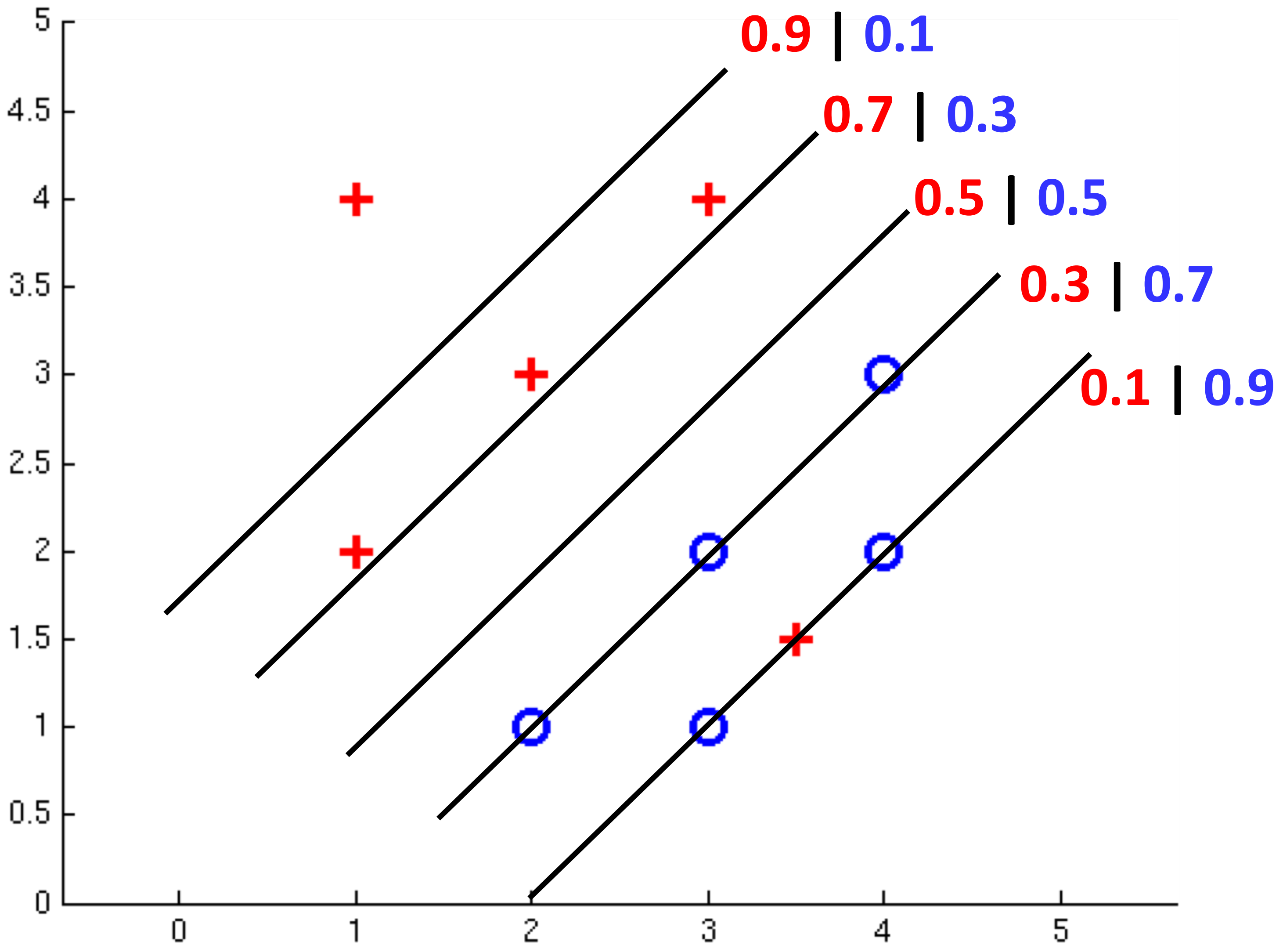


Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake



Non-Separable Case: Probabilistic Decision

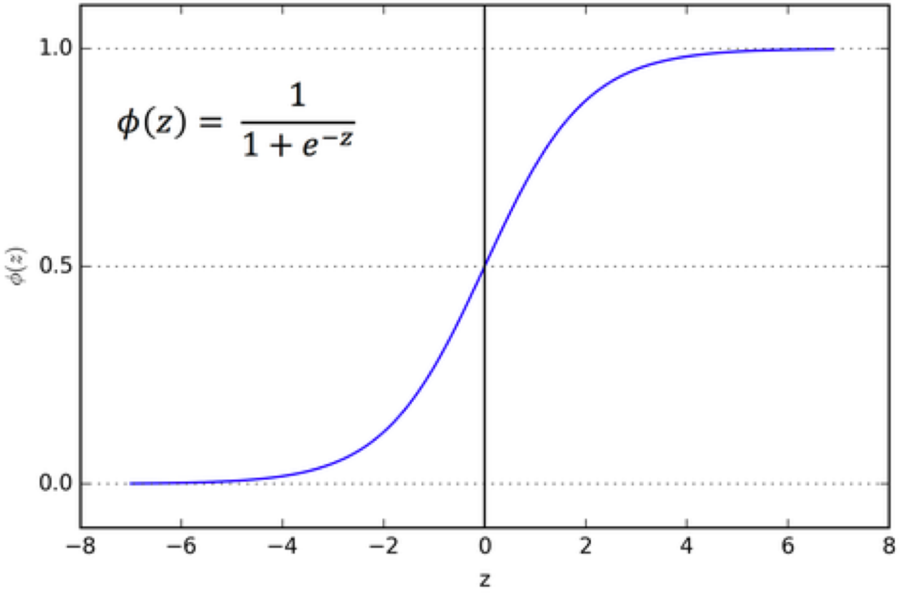


How to get probabilistic decisions?

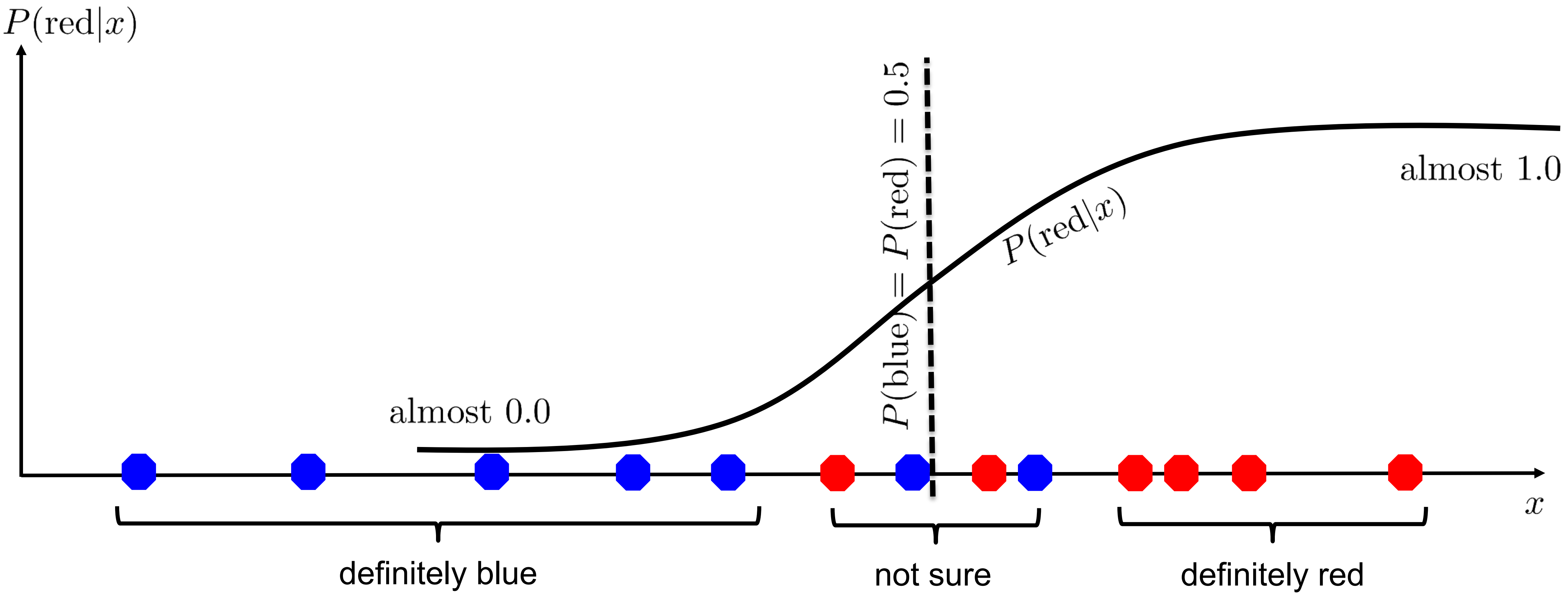
- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive → want probability going to 1
- If $z = w \cdot f(x)$ very negative → want probability going to 0

- Sigmoid function

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



A 1D Example

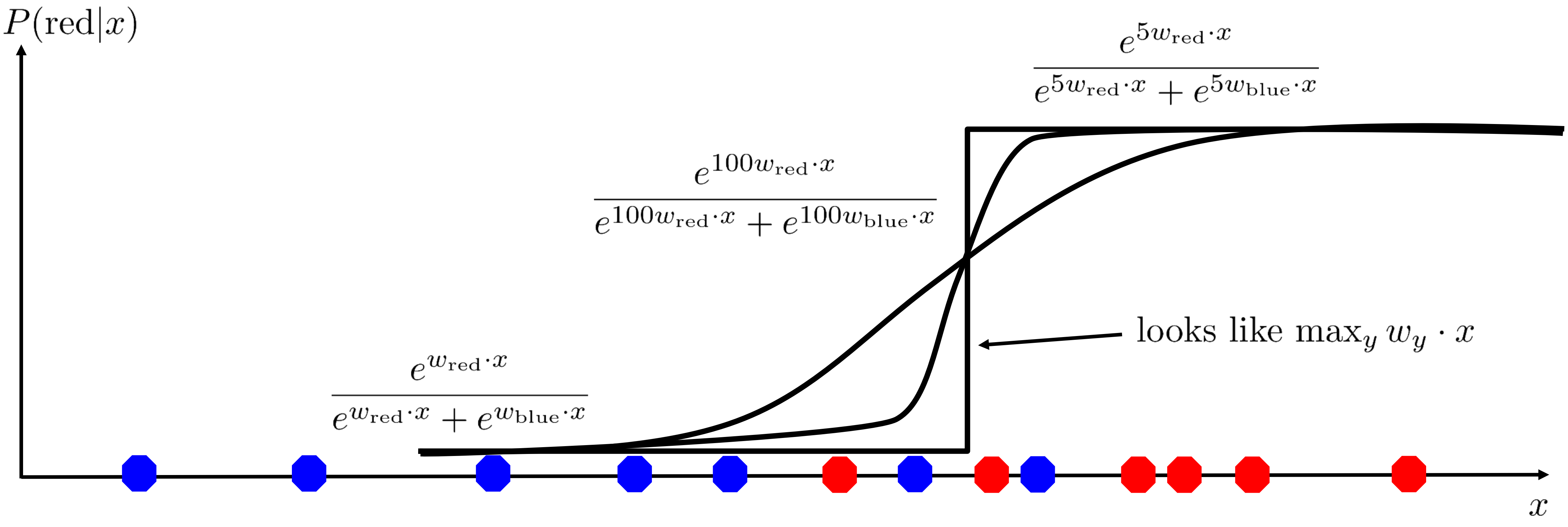


$$P(\mathrm{red}|x) = \frac{e^{w_{\mathrm{red}} \cdot x}}{e^{w_{\mathrm{red}} \cdot x} + e^{w_{\mathrm{blue}} \cdot x}}$$

probability increases exponentially as we move away from boundary

normalizer

The *Soft* Max



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

Best w?

- Maximum likelihood estimation:

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

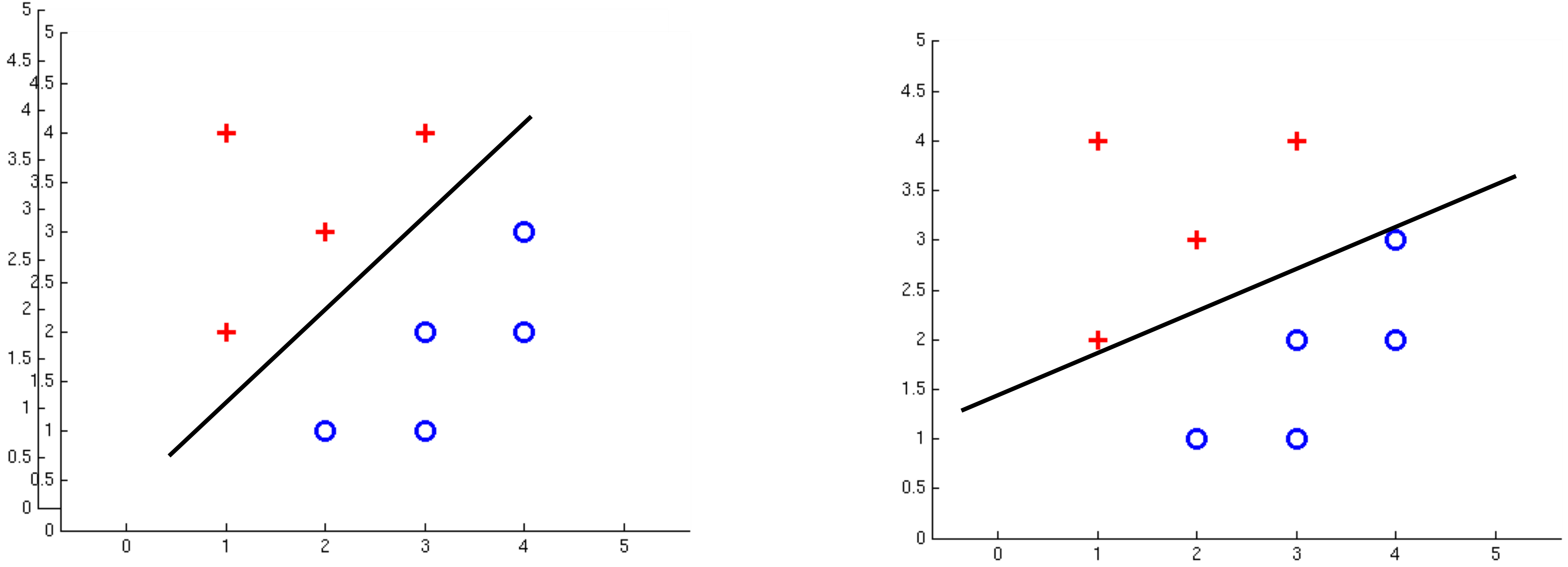
with:

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

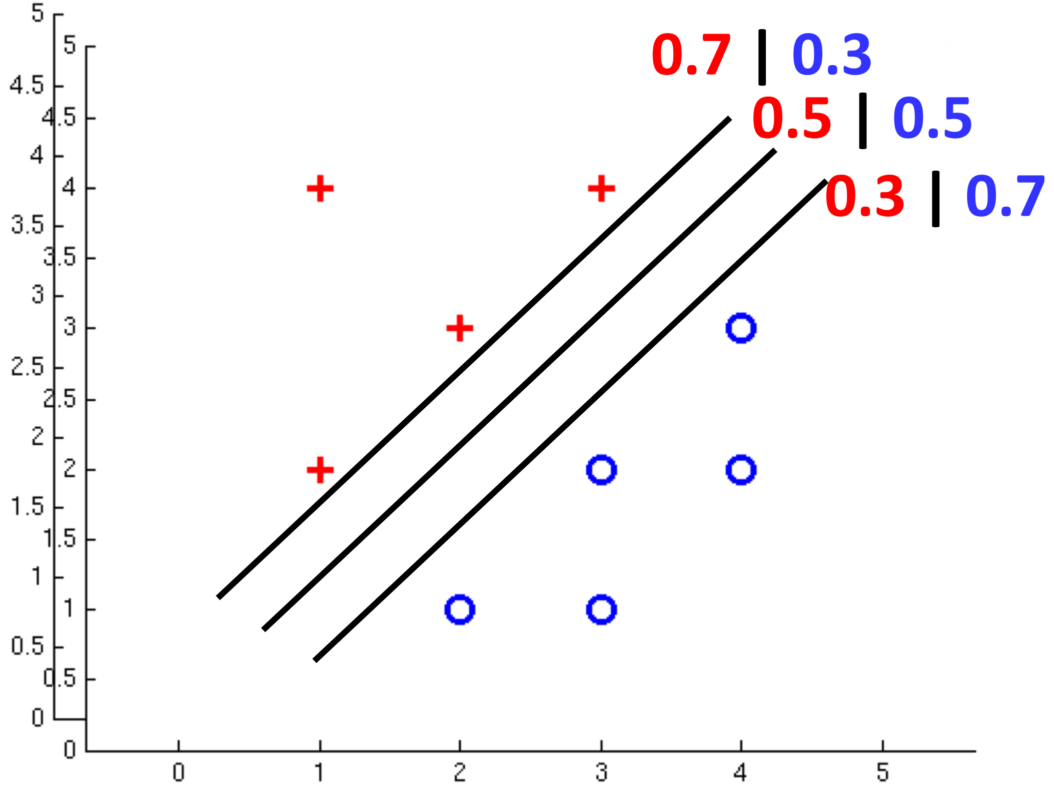
$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

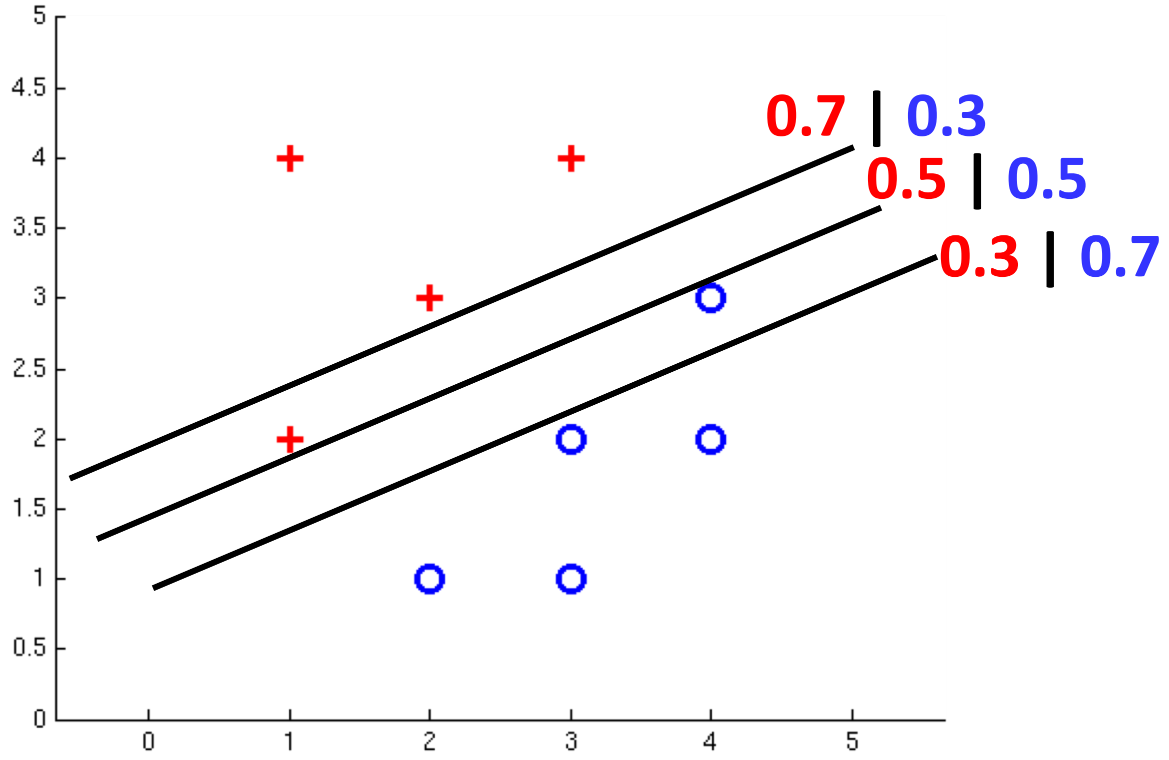
= Logistic Regression

Separable Case: Deterministic Decision – Many Options



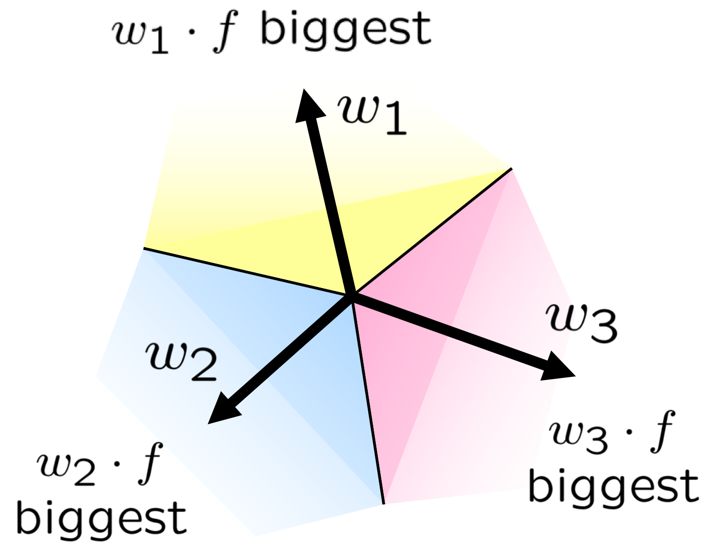
Separable Case: Probabilistic Decision – Clear Preference





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

Optimization

i.e., how do we solve:

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

Hill Climbing

Recall from CSPs lecture: simple, general idea

Start wherever

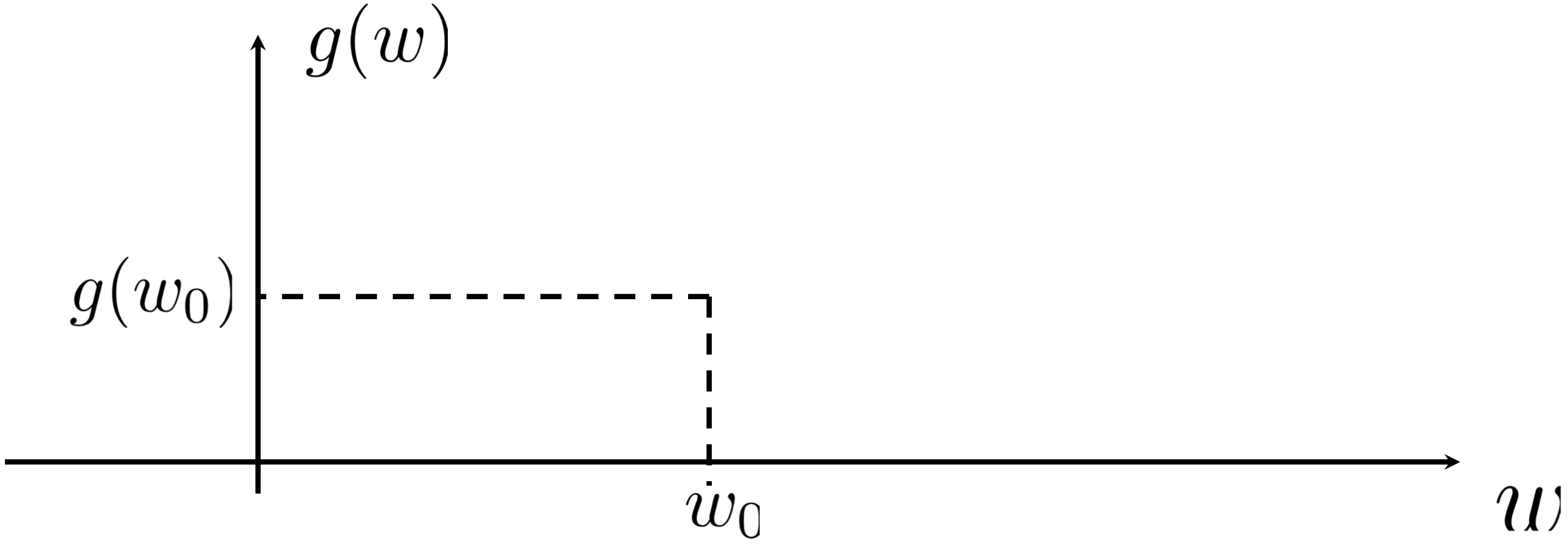
Repeat: move to the best neighboring state

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



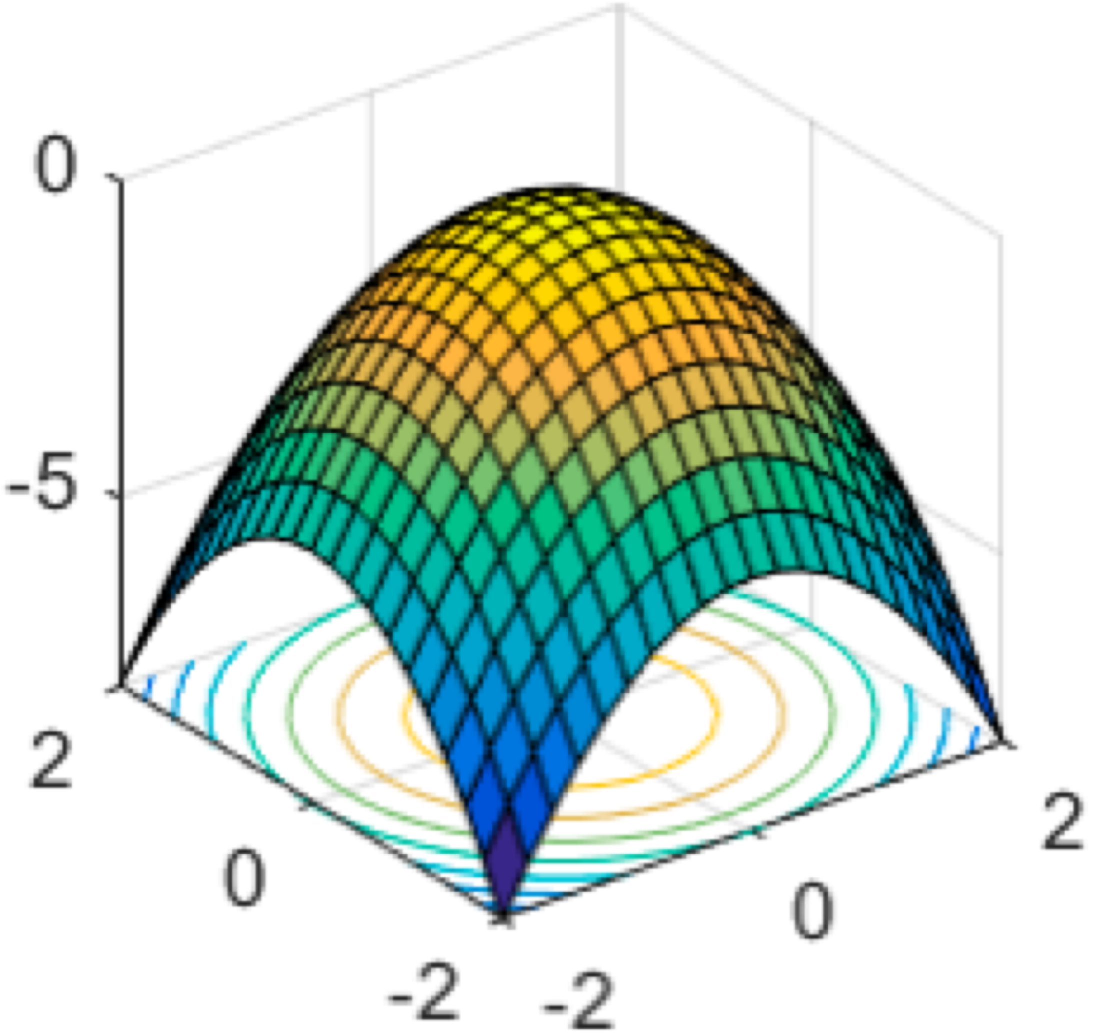
Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$

Then step in best direction

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

Tells which direction to step into

2-D Optimization



Source: offconvex.org

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

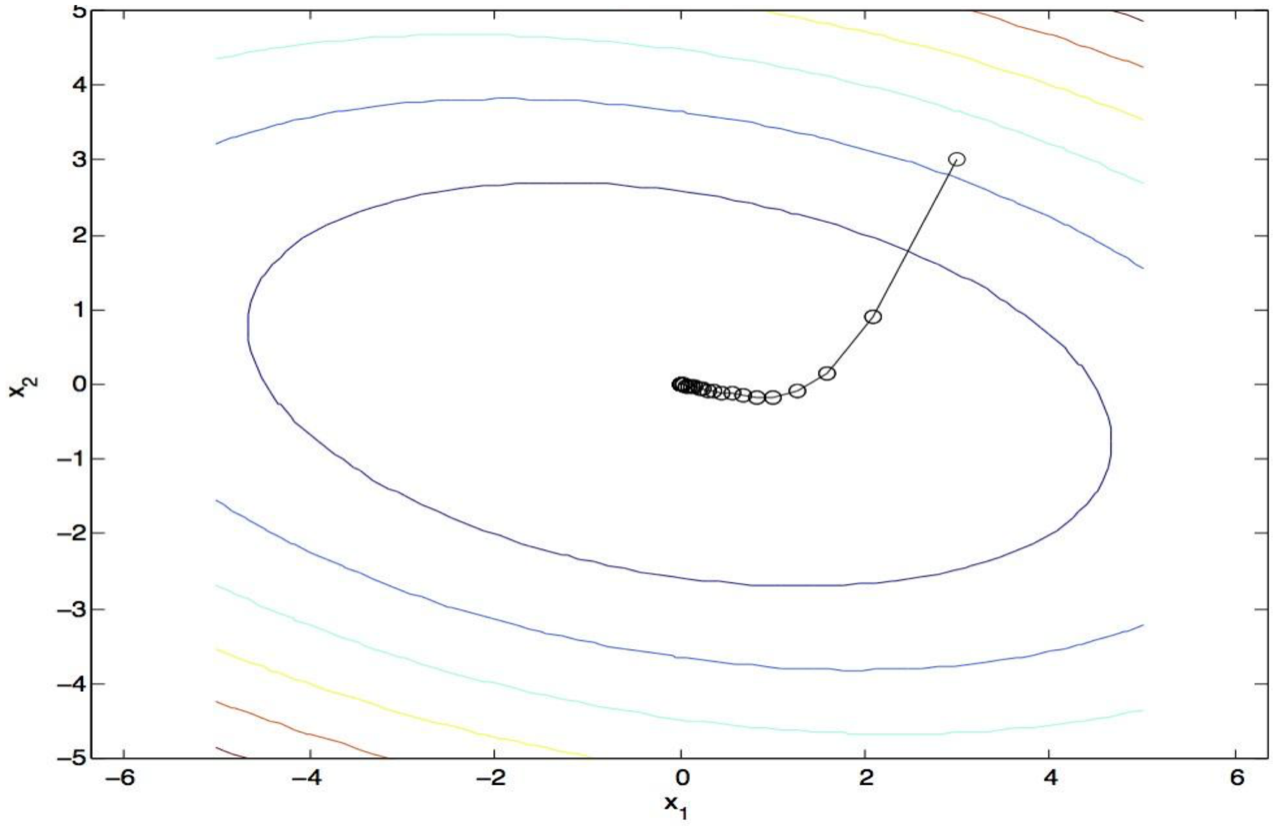


Figure source: Mathworks

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

w

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, …
```

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

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'