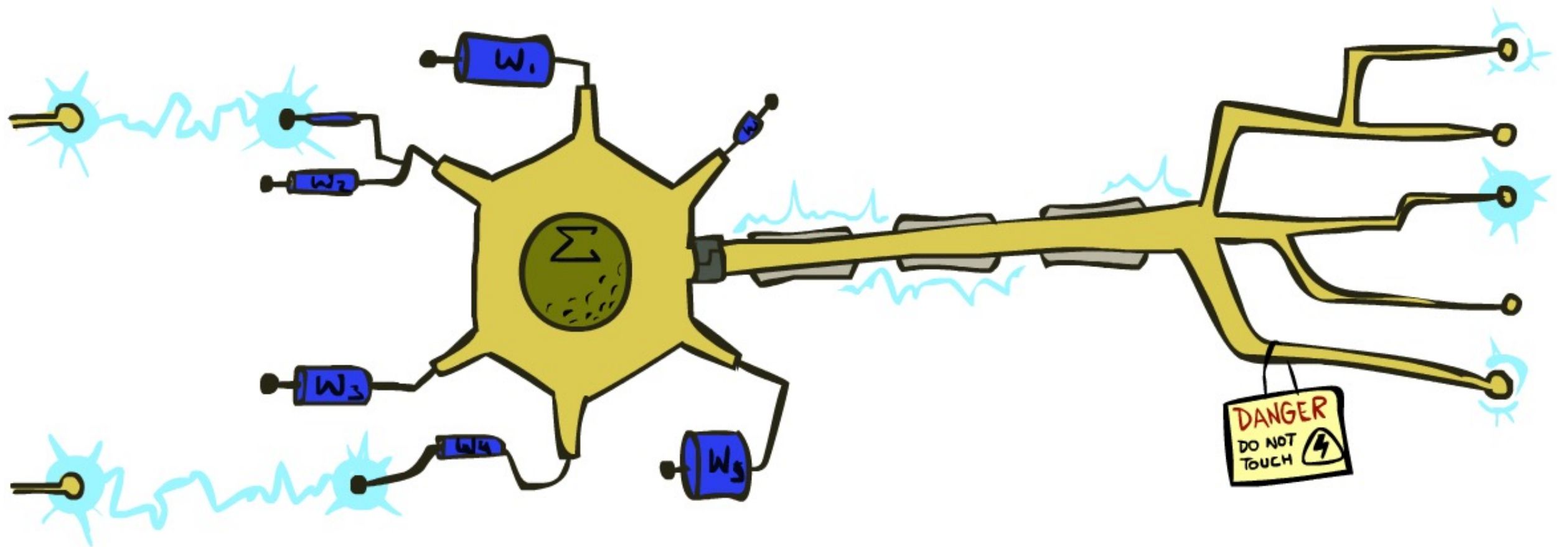


# Ve492: Introduction to Artificial Intelligence

## Discriminative Learning



Paul Weng

UM-SJTU Joint Institute

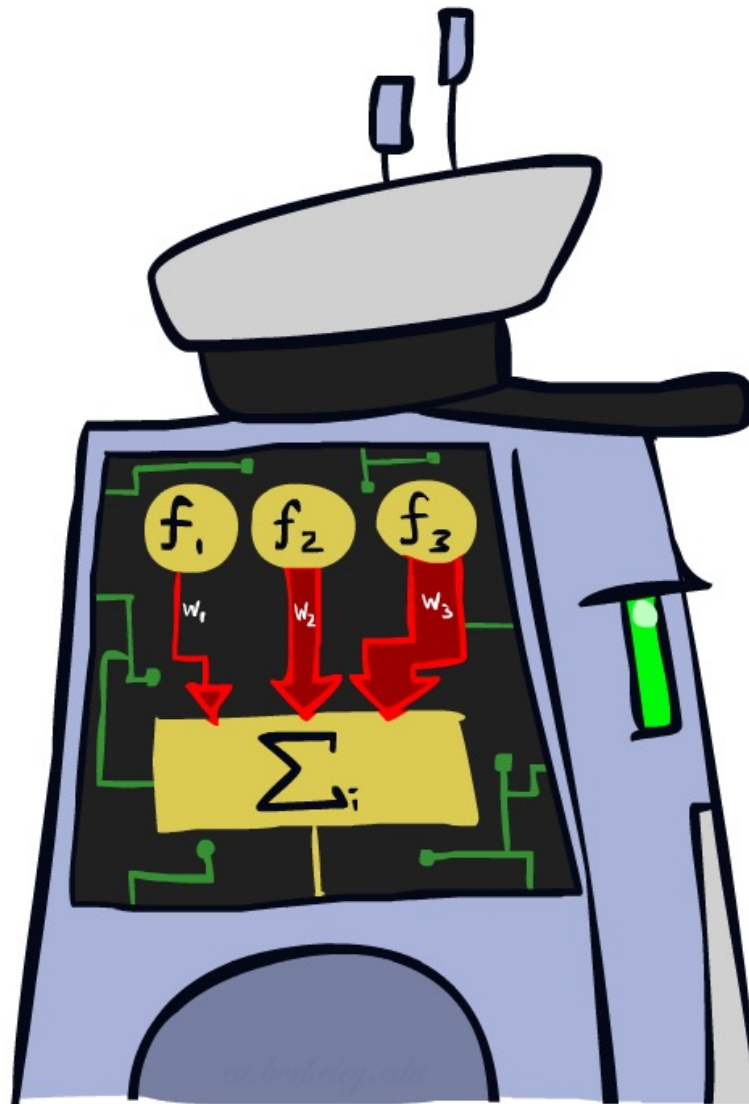
Slides adapted from <http://ai.berkeley.edu>, AIMA, UM

# Error-Driven Classification

---



# Linear Classifiers



# Feature Vectors

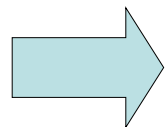
$x$

$\varphi(x)$

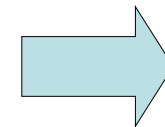
$y$

Hello,

Do you want free  
printr cartridges?  
Why pay more when  
you can get them  
ABSOLUTELY FREE!  
Just

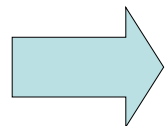


# free	:	2
YOUR_NAME	:	0
MISSPELLED	:	2
FROM_FRIEND	:	0
...		

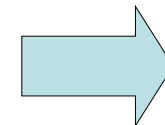


SPAM

2



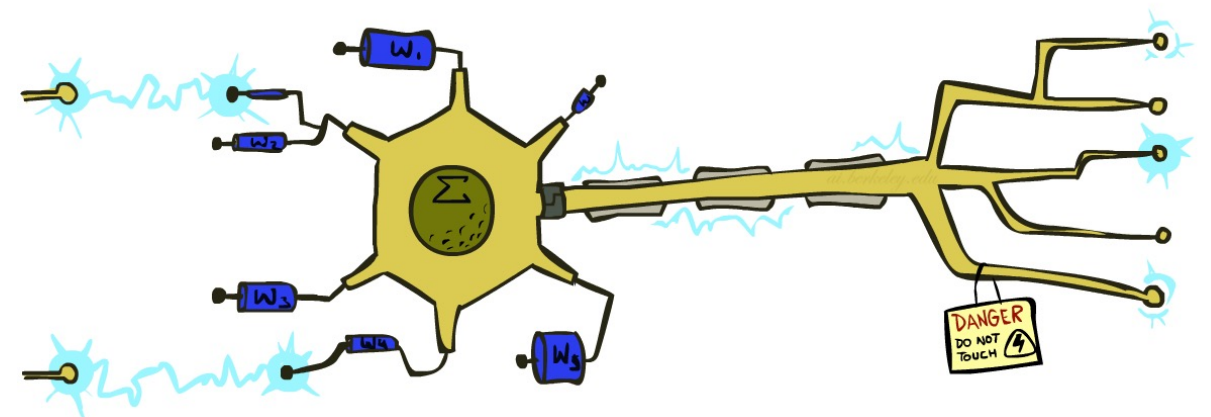
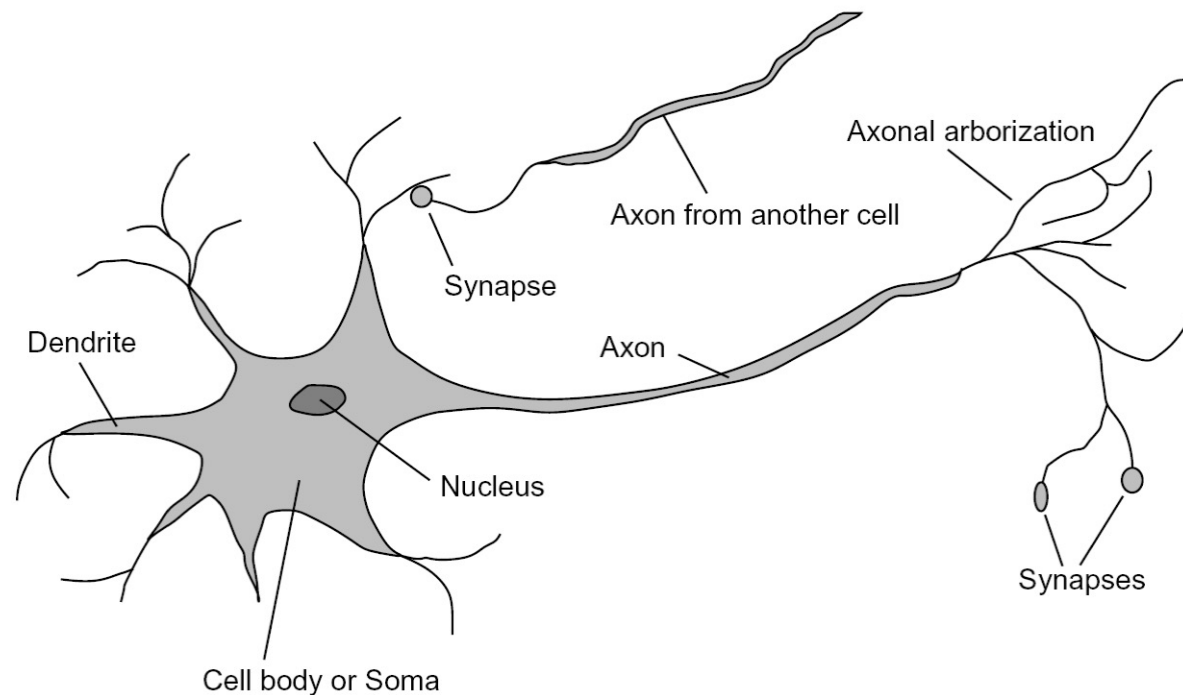
PIXEL-7,12	:	1
PIXEL-7,13	:	0
...		
NUM_LOOPS	:	1
...		



"2"

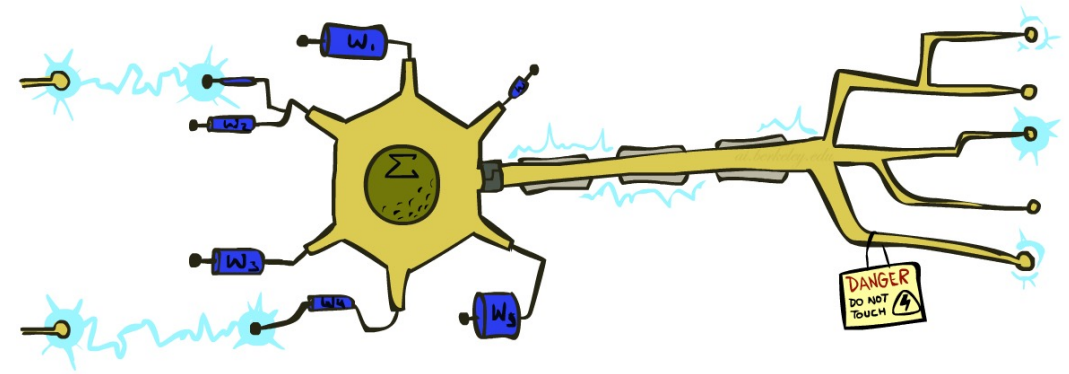
# Some (Simplified) Biology

- ❖ Very loose inspiration: human neurons



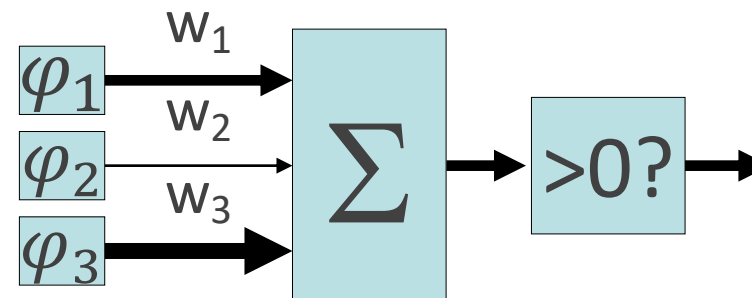
# Linear Classifiers

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



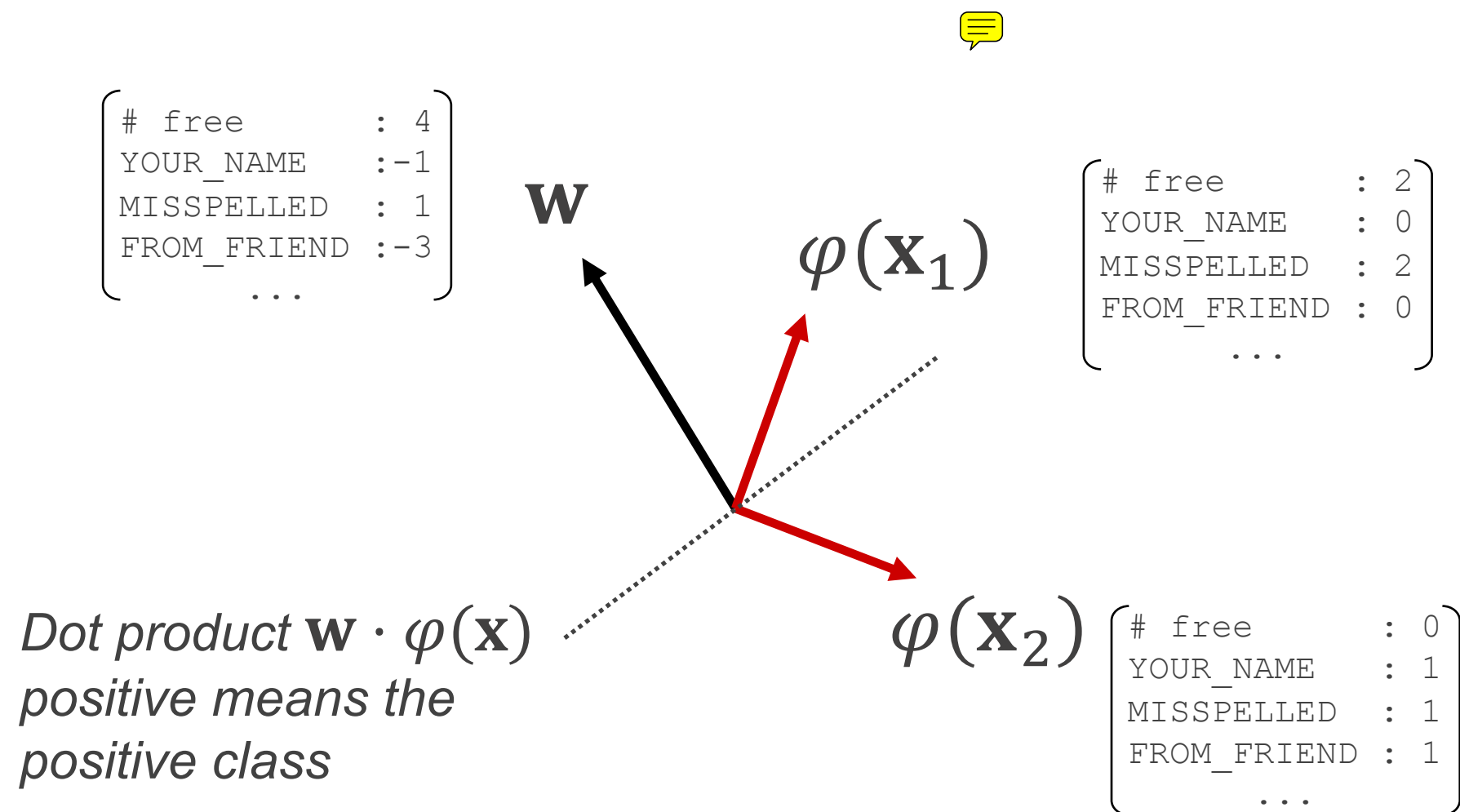
$$\text{activation}_{\mathbf{w}}(\mathbf{x}) = \sum_i w_i \varphi_i(\mathbf{x}) = \mathbf{w} \cdot \boldsymbol{\varphi}(\mathbf{x})$$

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



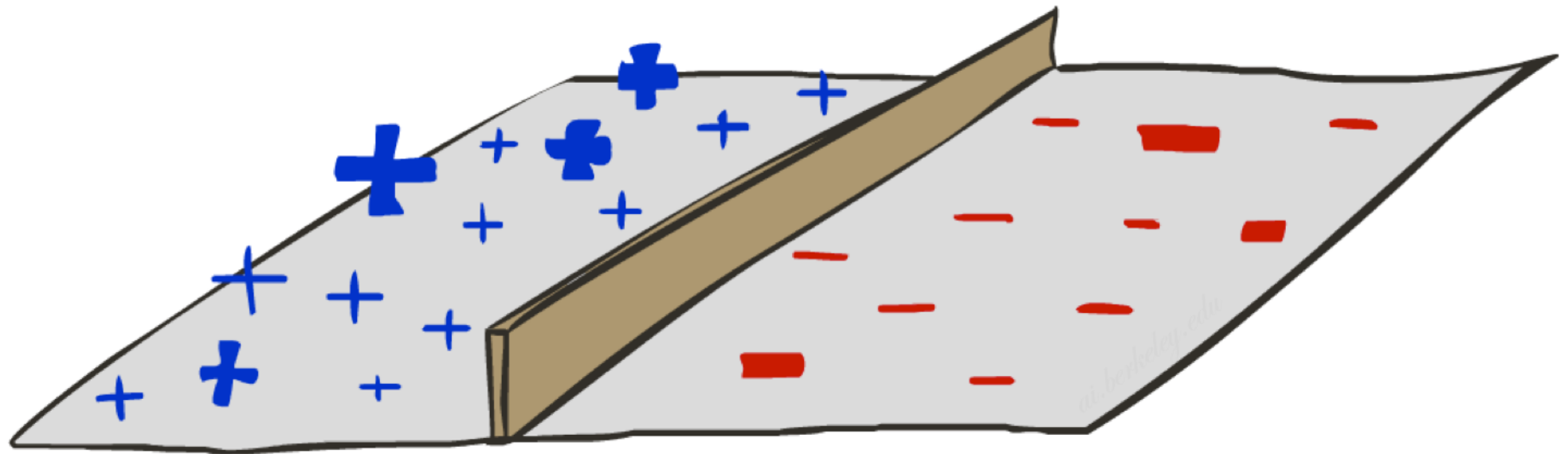
# Weights

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





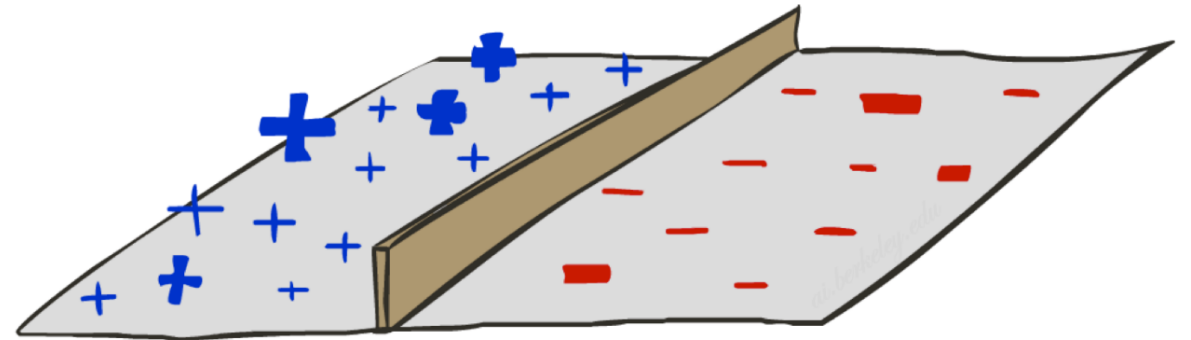
# Decision Rules





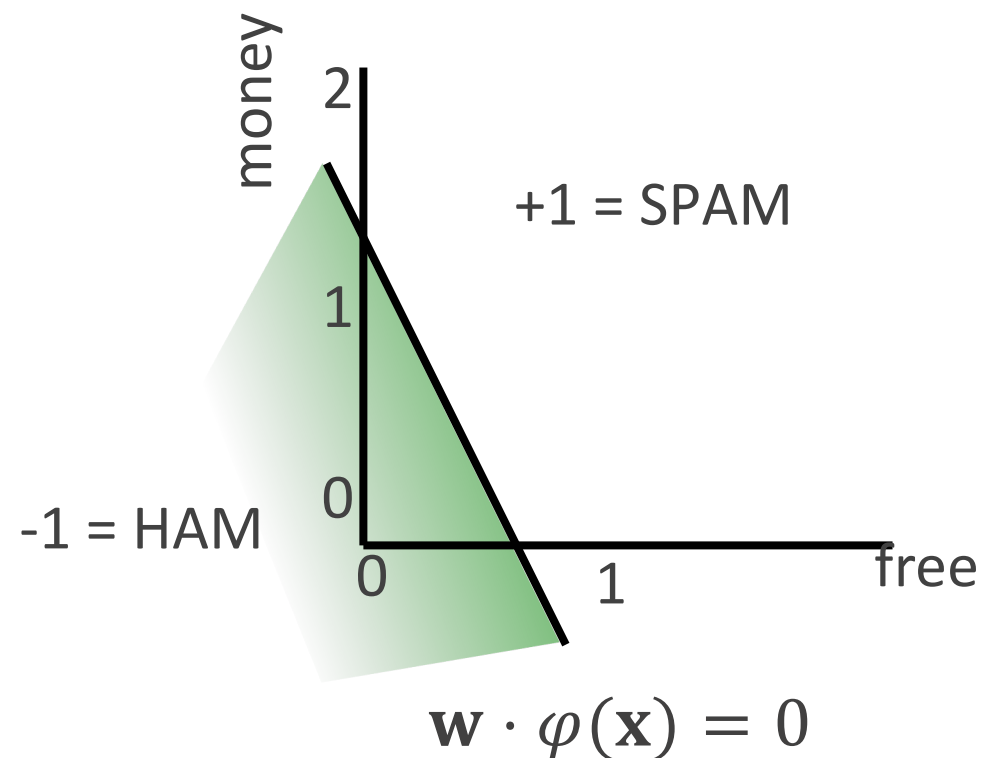
# Binary Decision Rule

- ❖ In the space of feature vectors
  - ❖ Examples are points
  - ❖ Any weight vector is a hyperplane
  - ❖ One side corresponds to  $Y=+1$
  - ❖ Other corresponds to  $Y=-1$

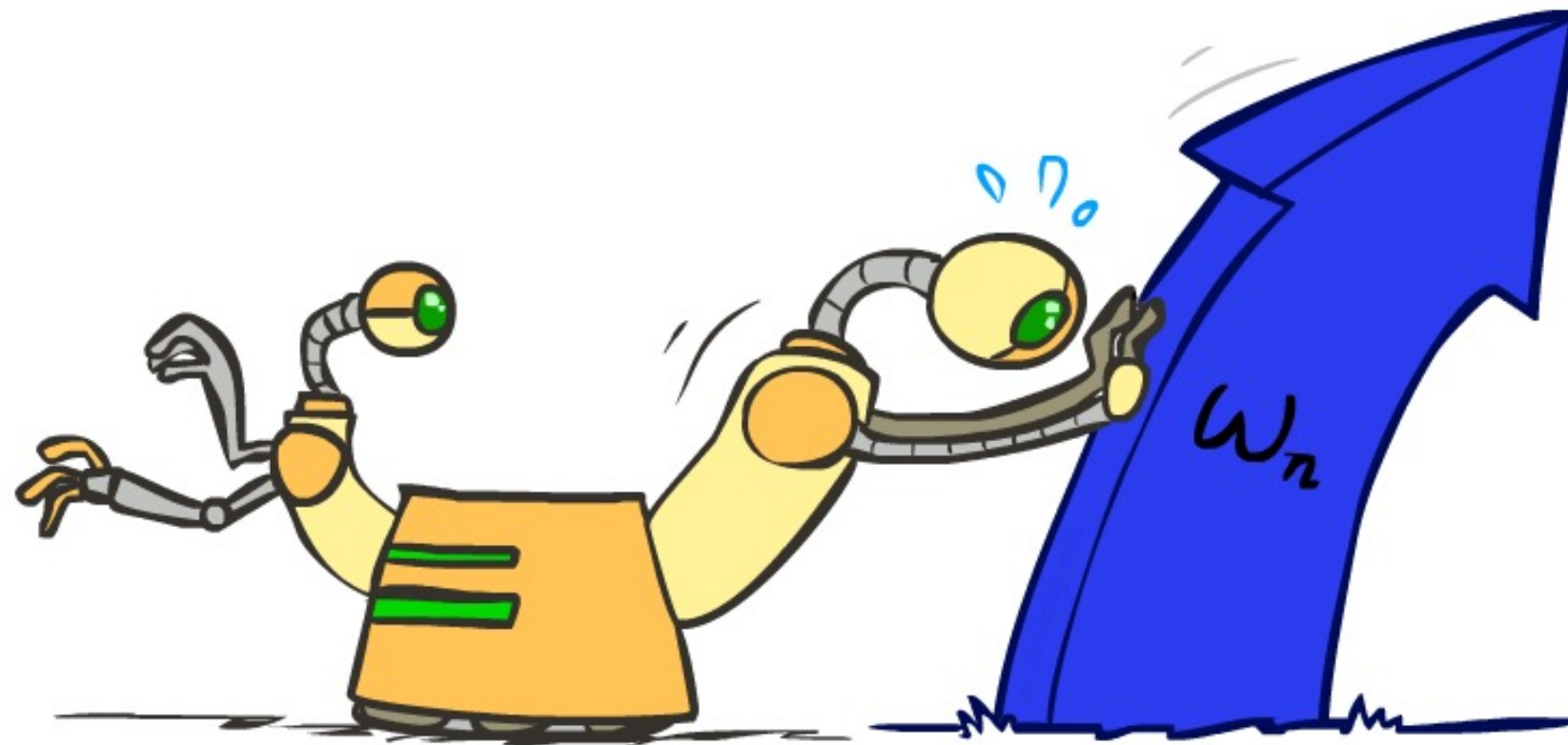


$w$

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

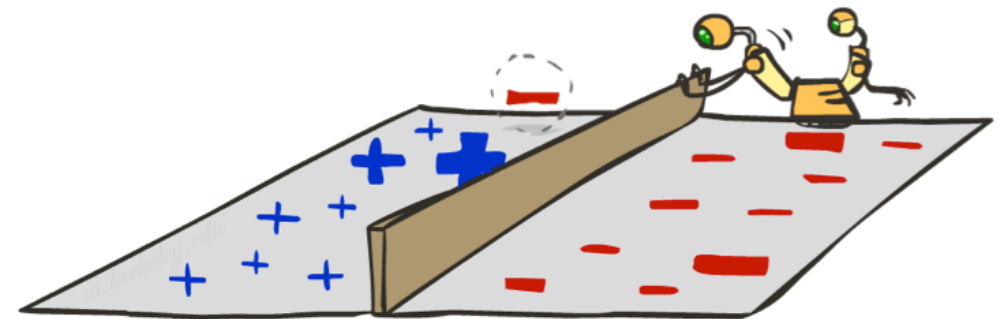
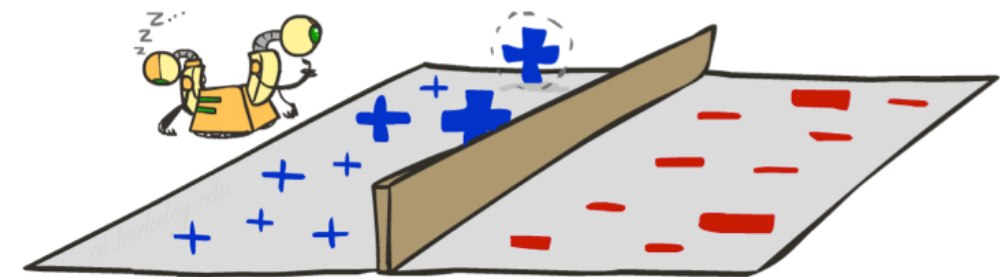
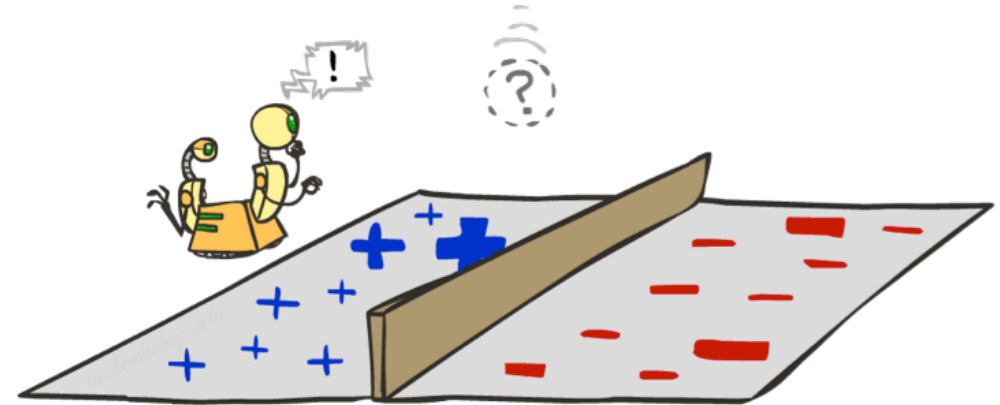


# Weight Updates



# Learning: Binary Perceptron

- ❖ Start with weights = 0
- ❖ For each training instance  $(\mathbf{x}, y^*)$ :
  - ❖ Classify with current weights
- ❖ If correct (i.e.,  $\hat{y} = y^*$ ), no change!
- ❖ If wrong: adjust the weight vector



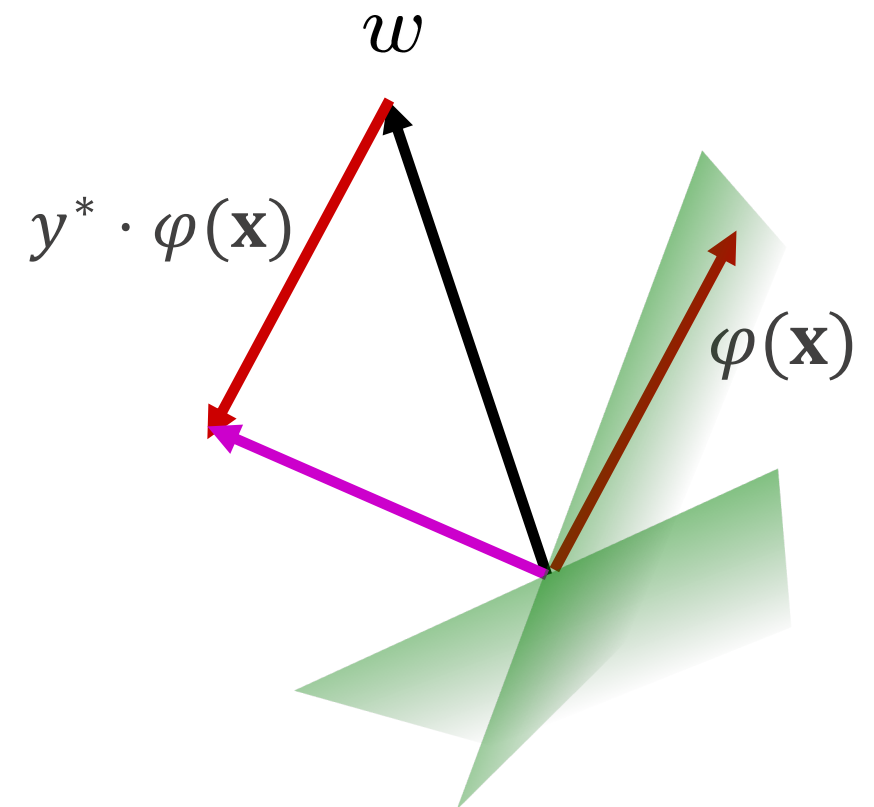
# Learning: Binary Perceptron

- ❖ Start with weights = 0
- ❖ For each training instance  $(\mathbf{x}, y^*)$ :
  - ❖ Classify with current weights

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

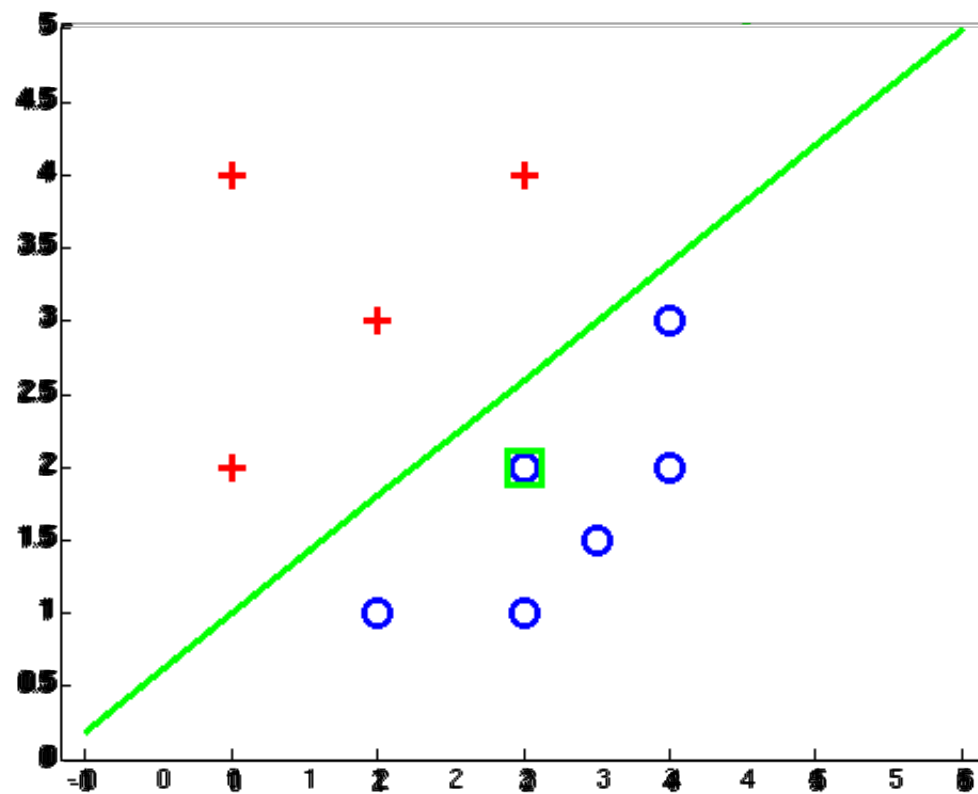
- ❖ If correct (i.e.,  $\hat{y} = y^*$ ), no change!
- ❖ If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if  $y^*$  is -1.

$$\mathbf{w} = \mathbf{w} + y^* \cdot \varphi(\mathbf{x})$$



# Examples: Perceptron

## ❖ Separable Case



# Multiclass Decision Rule

❖ If we have multiple classes:

❖ A weight vector for each class:

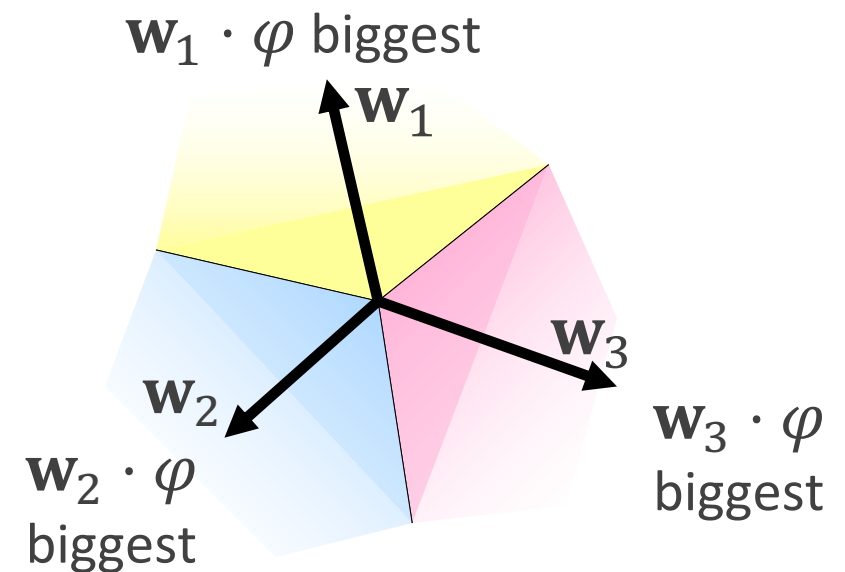
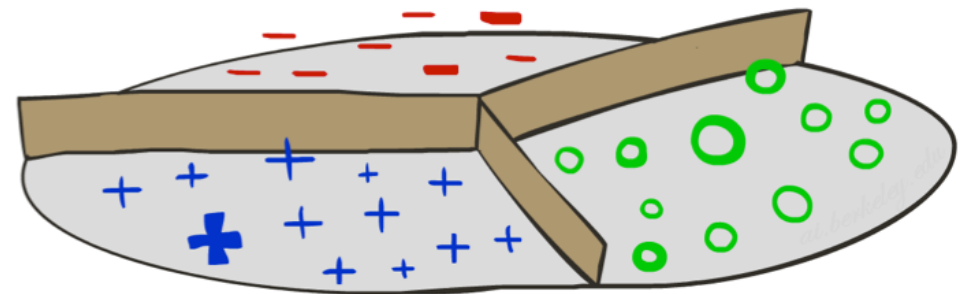
$$\mathbf{w}_y$$

❖ Score (activation) of a class  $y$ :

$$\mathbf{w}_y \cdot \varphi(\mathbf{x})$$

❖ Prediction highest score wins

$$\hat{y} = \operatorname{argmax}_y \mathbf{w}_y \cdot \varphi(\mathbf{x})$$



# Quiz: Binary Classif. As Multiclass Decision Rule

---

- ❖ Multiclass decision rule

$$\hat{y} = \operatorname{argmax}_y \mathbf{w}_y \cdot \varphi(\mathbf{x})$$

- ❖ Denote **W** the weight vector of the positive class.
- ❖ What could be the weight vector of the negative class?



# Learning: Multiclass Perceptron

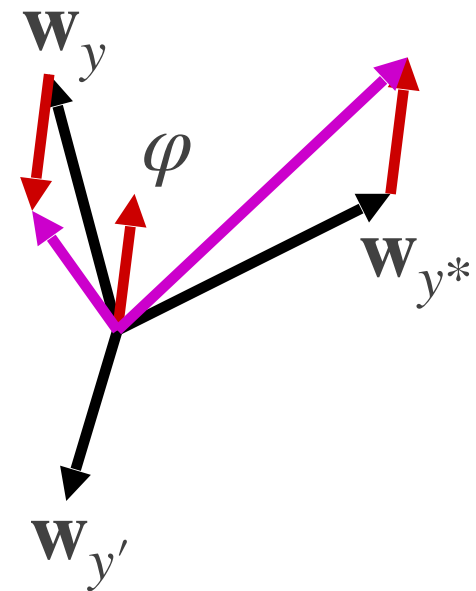
- ❖ Start with all weights = 0
- ❖ Pick up training examples one by one
- ❖ Predict with current weights

$$\hat{y} = \operatorname{argmax}_y \mathbf{w}_y \cdot \varphi(\mathbf{x})$$

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

$$\mathbf{w}_{\hat{y}} = \mathbf{w}_{\hat{y}} - \varphi(\mathbf{x})$$

$$\mathbf{w}_{y^*} = \mathbf{w}_{y^*} + \varphi(\mathbf{x})$$



# 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
...		

$w_{TECH}$

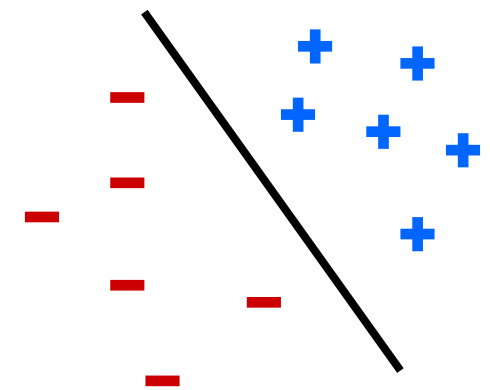
BIAS	:	0
win	:	0
game	:	0
vote	:	0
the	:	0
...		

# Properties of Perceptrons

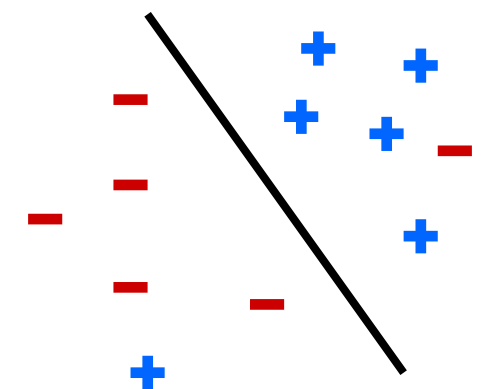
- ❖ Separability: true if some parameters get the training set perfectly correct
- ❖ Convergence: if the training set is separable, perceptron will eventually converge (binary case)
- ❖ Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

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

Separable

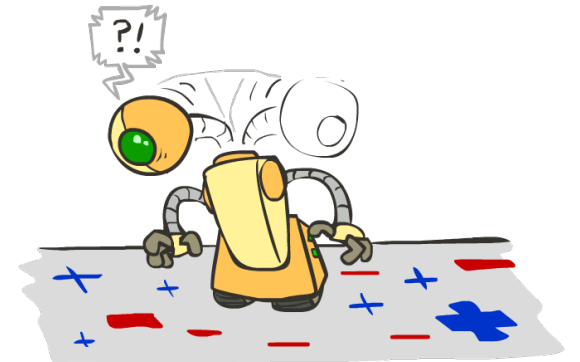
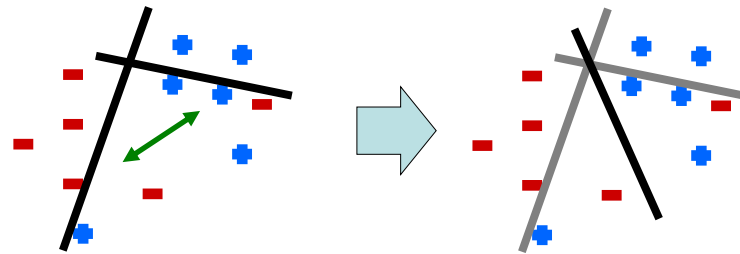


Non-Separable

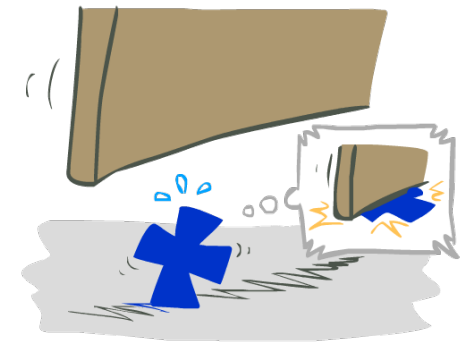
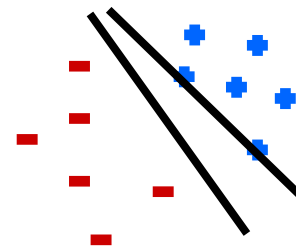


# Problems with the Perceptron

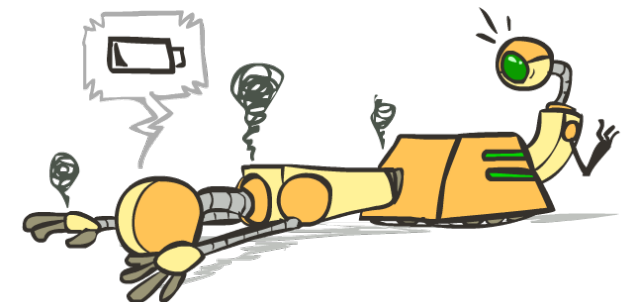
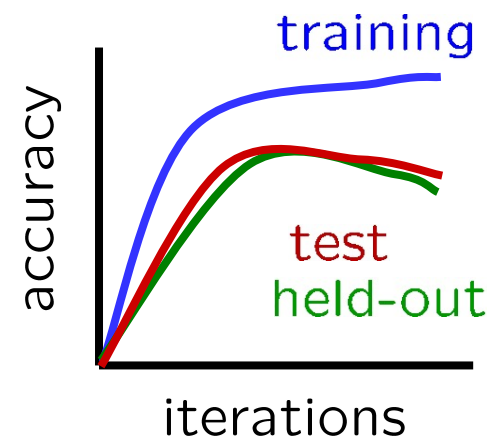
- ❖ Noise: if the data isn't separable, weights might thrash
  - ❖ Averaging weight vectors over time can help (averaged perceptron)



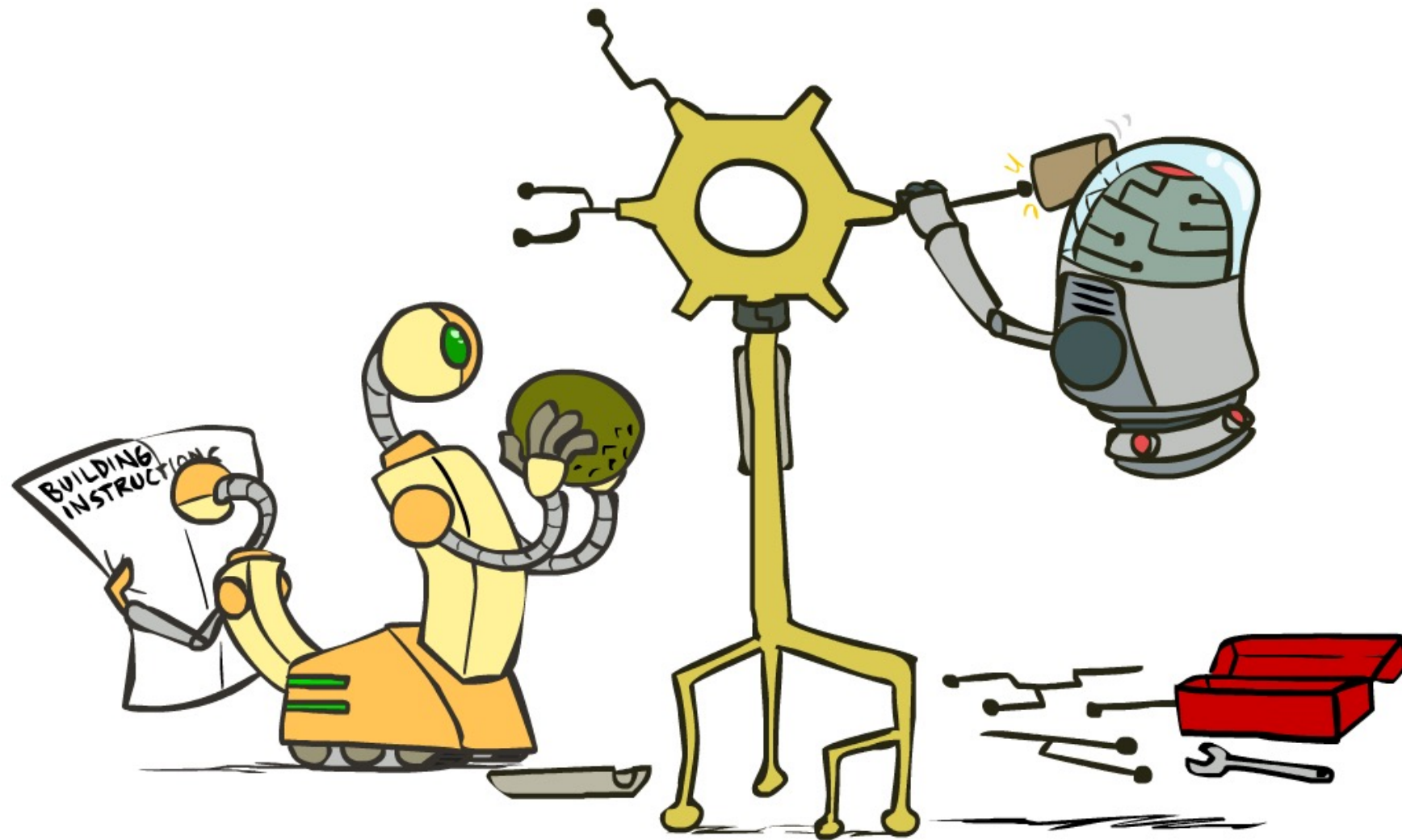
- ❖ Mediocre generalization: finds a “barely” separating solution



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



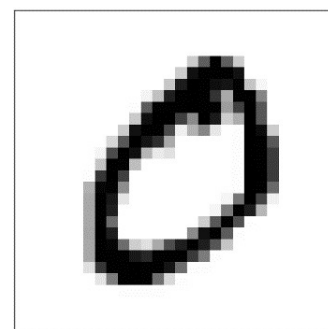
# Improving the Perceptron



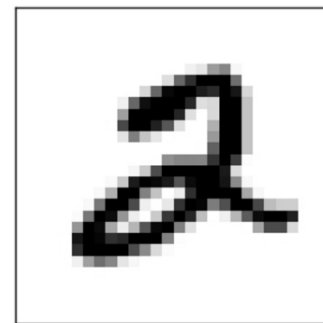
# Probabilistic Classification

- ❖ Naïve Bayes provides *probabilistic* classification

Answers the query:  $P(Y = y_i | x_1, \dots, x_n)$



1: 0.001  
2: 0.001  
...  
0: 0.991



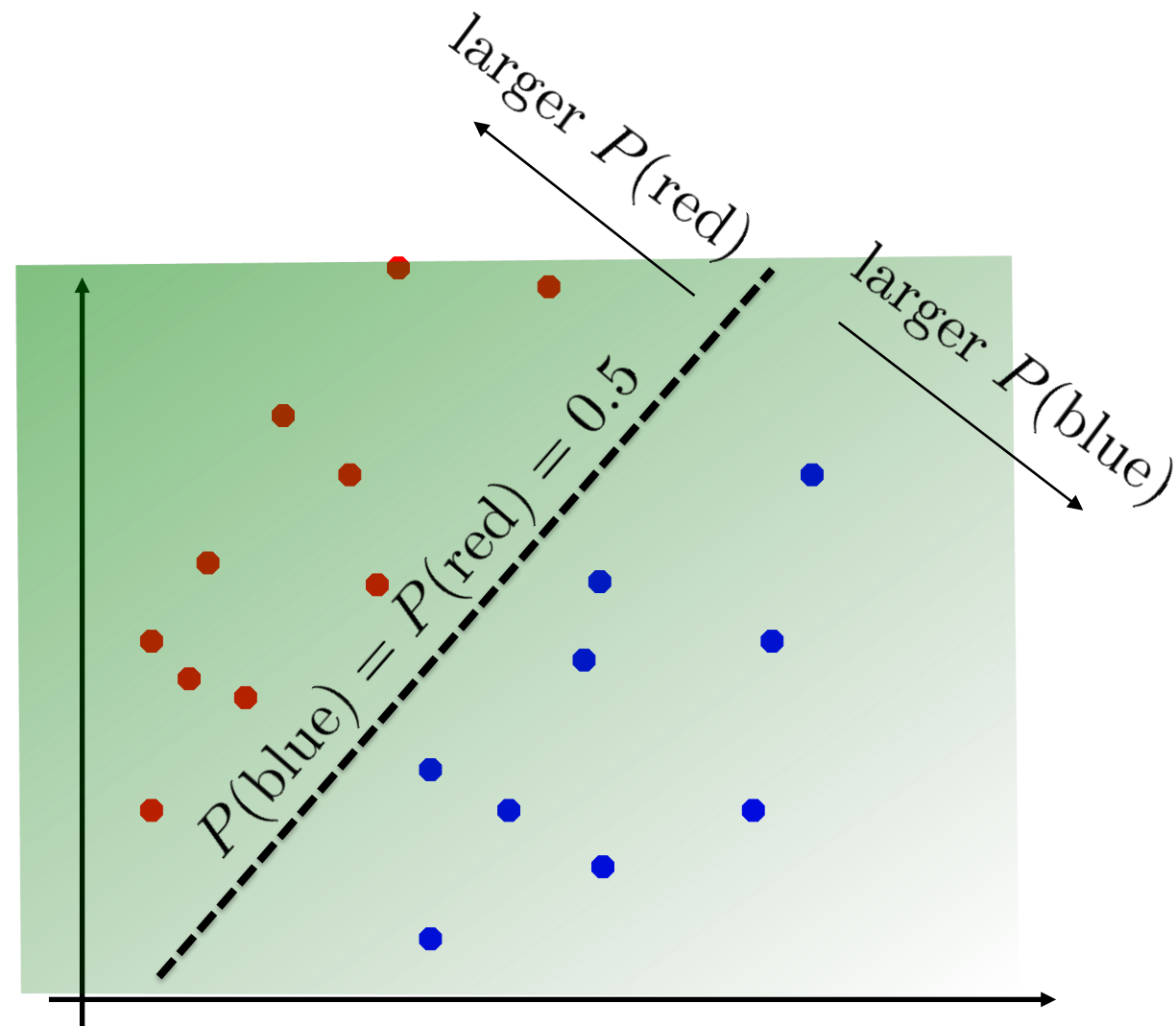
1: 0.001  
2: 0.703  
...  
6: 0.264  
...  
0: 0.001

- ❖ 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: To simplify notations, “ $x$ ” denotes “ $\varphi(\mathbf{x})$ ” from now on

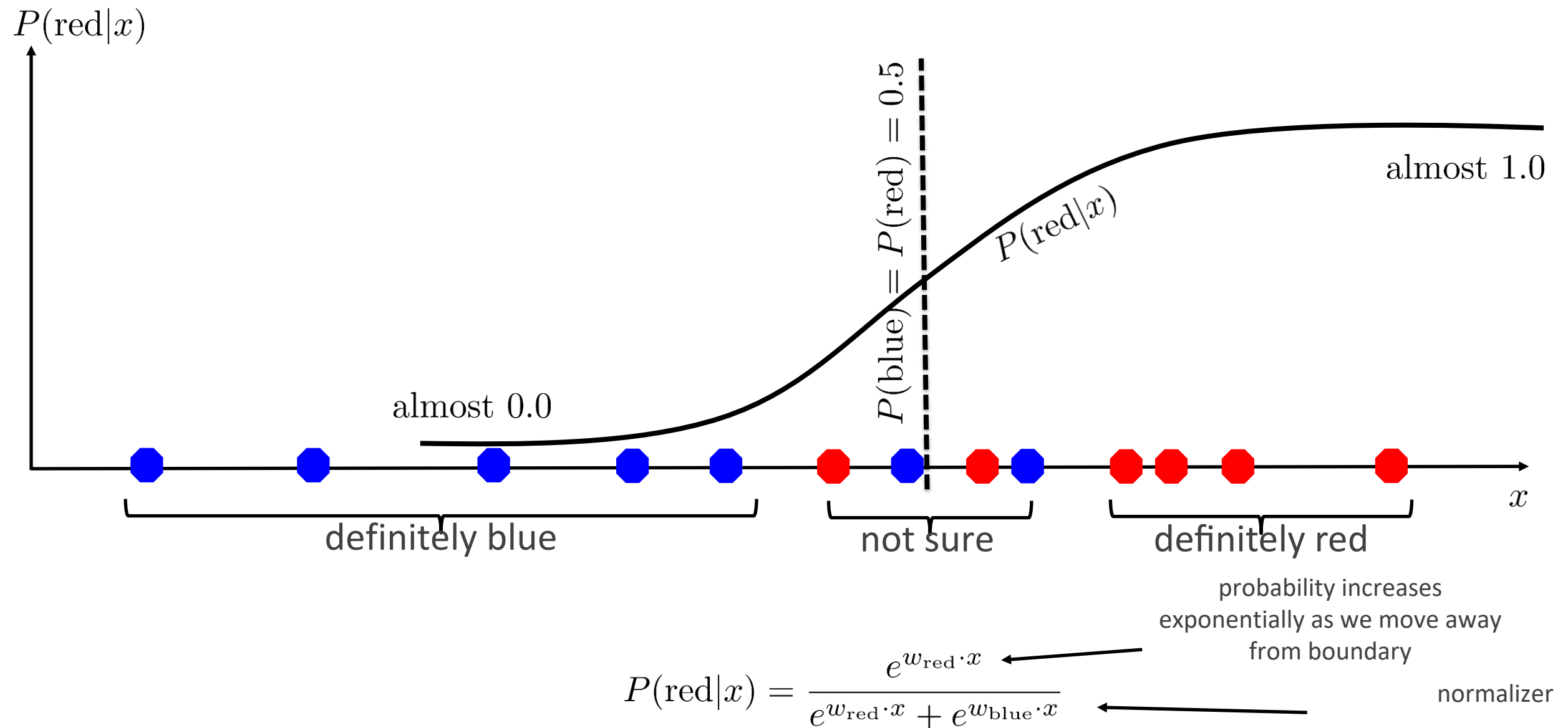
# *A Probabilistic* Perceptron



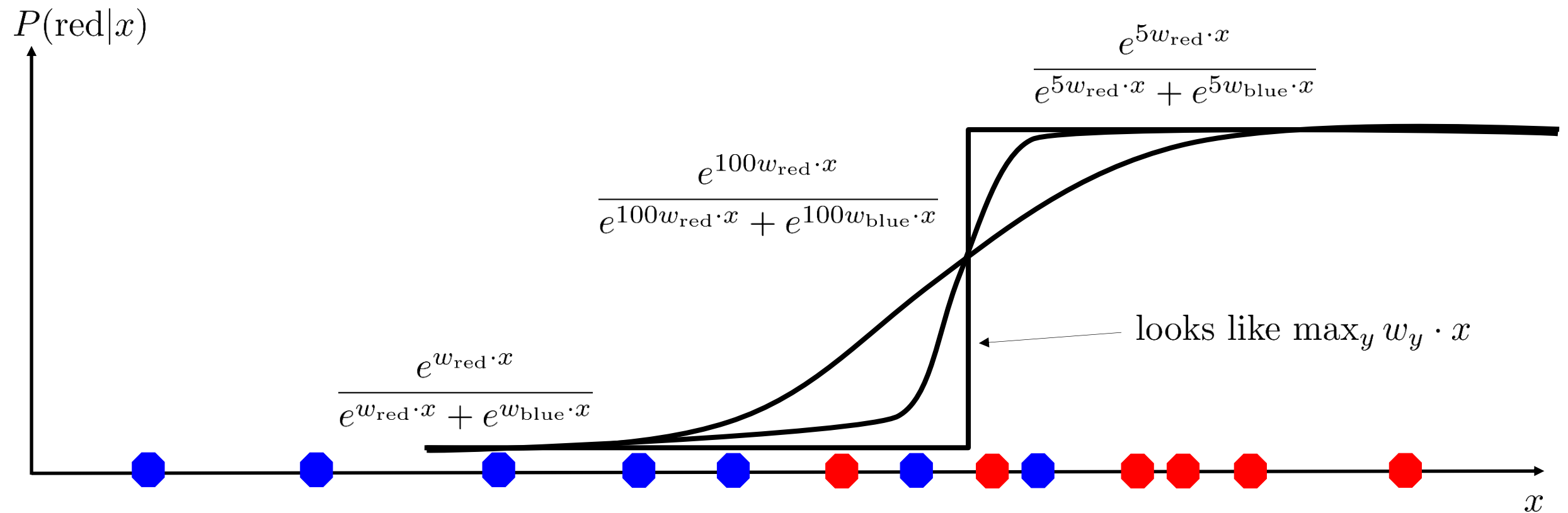
As  $w_y \cdot x$  gets bigger,  $P(y|x)$  gets bigger



# A 1D Example



# The *Soft* Max

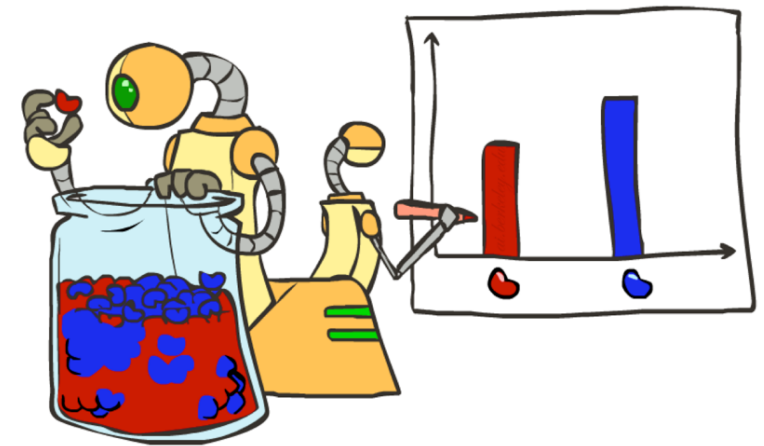


$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$

# How to Learn?

## ❖ Maximum likelihood estimation

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



## ❖ Maximum *conditional* likelihood estimation

$$\begin{aligned}\theta^* &= \arg \max_{\theta} P(\mathbf{Y}|\mathbf{X}, \theta) \\ &= \arg \max_{\theta} \prod_i \underbrace{P_{\theta}(y_i|x_i)} \\ \ell(w) &= \prod_i \frac{e^{w_{y_i} \cdot x_i}}{\sum_y e^{w_y \cdot x_i}}\end{aligned}$$

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

# 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  $ll(w)$

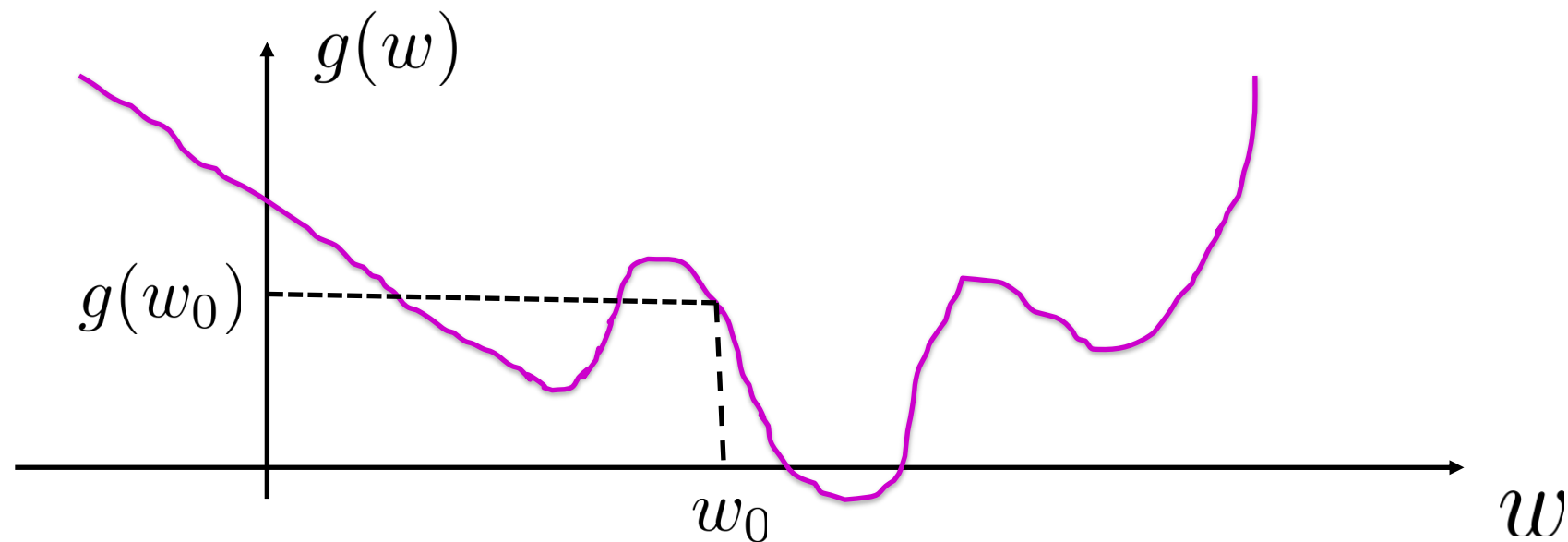
❖ Challenge: how to find a good  $w$  ?

$$\max_w ll(w)$$

❖ Equivalently:

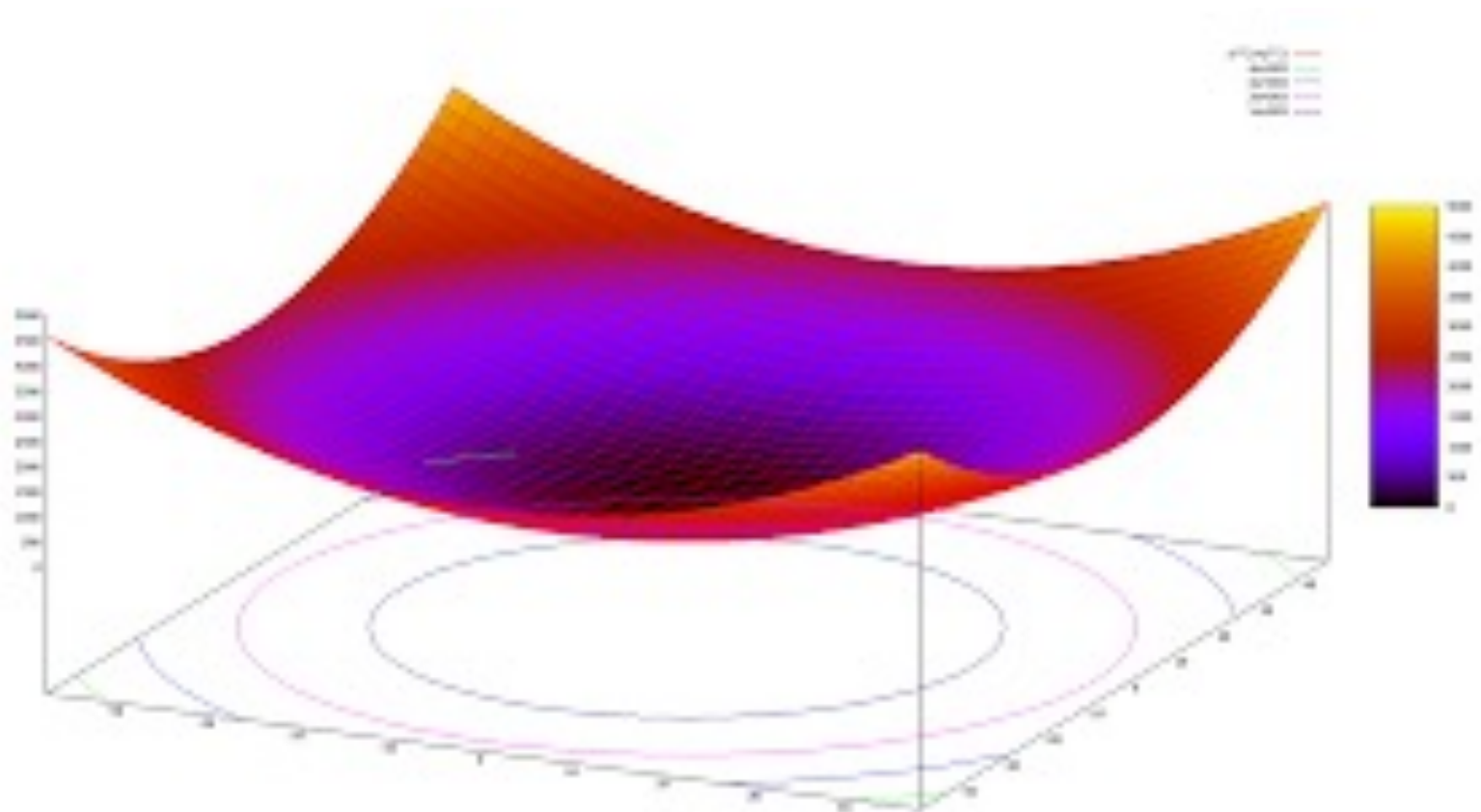
$$\min_w -ll(w)$$

# 1D optimization



- ❖ Could evaluate  $g(w_0 + h)$  and  $g(w_0 - h)$
- ❖ Then step in best direction
- ❖ Or, evaluate derivative:  $\frac{\partial g(w_0)}{\partial w} = \lim_{h \rightarrow 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$
- ❖ 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

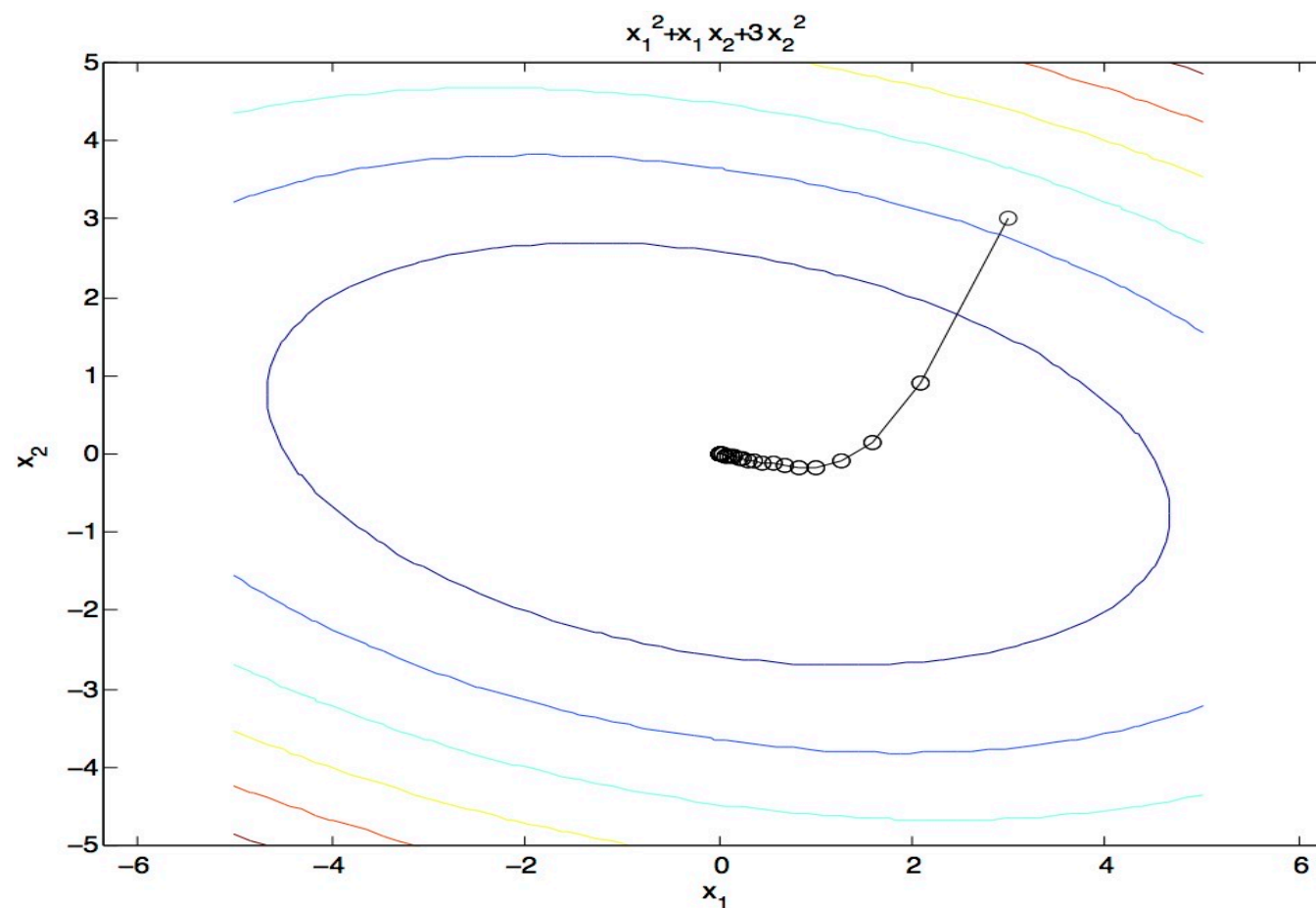


Figure source: Mathworks

# Steepest Direction

- ❖ Steepest Direction = direction of the gradient

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

# How to Learn?

---

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

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

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

# 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  $\nabla_w \mathcal{L} = \sum_i \Delta_i$

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

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

- ❖  $\alpha$ : learning rate — hyperparameter 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  $1 - P(y_i|x_i)$

$\underbrace{\hspace{10em}}$   
probability of *incorrect* answer

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

with weight  $P(y|x_i)$

$\underbrace{\hspace{10em}}$   
probability of *incorrect* answer

compare this to the  
multiclass perceptron:  
probabilistic  
weighting!

# Logistic Regression Demo!

<https://playground.tensorflow.org/>