CS 471/571(Fall 2023): Introduction to Artificial Intelligence

Lecture 24: Perceptrons, Logistic Regression, and Neural Nets

Thanh H. Nguyen

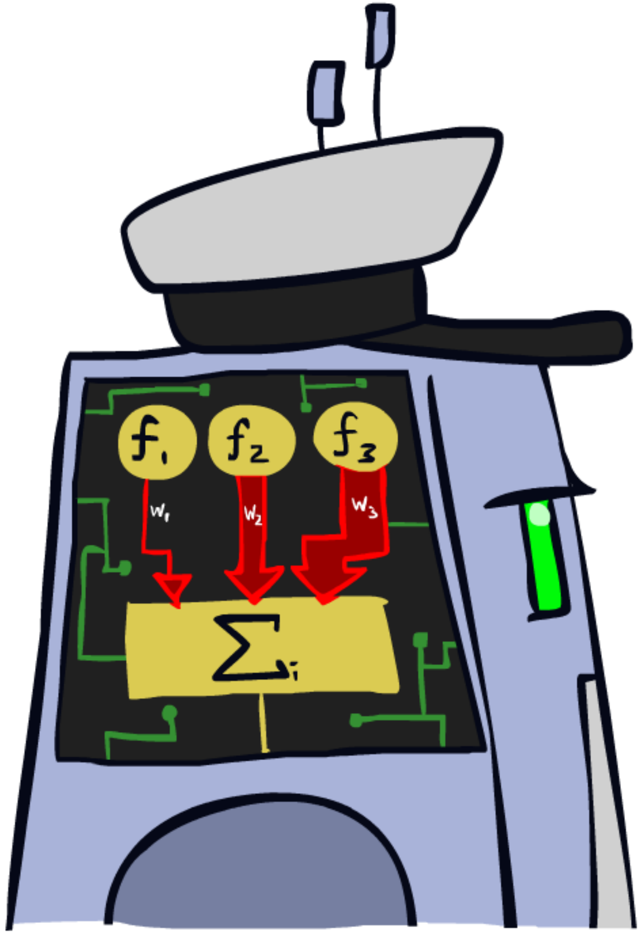
Source: http://ai.berkeley.edu/home.html

Announcement and Reminder

- Written assignment 4
  - Deadline: Wednesday, November 29th, 2023.

- Student experience survey
  - Deadline: 06:00 AM on Monday, Dec 4th, 2023
  - If >= 80% of students complete the survey, everyone will get an extra <u>2% credit</u> for your final grade

Linear Classifiers



Feature Vectors

Input | Features x | Label 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

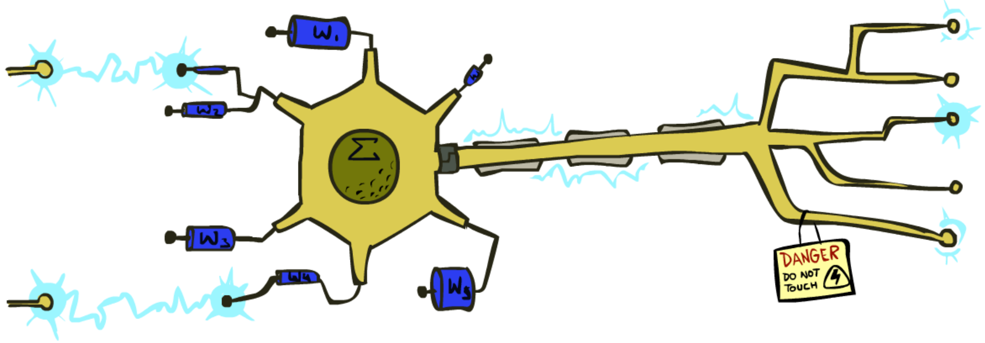
+

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

“2”

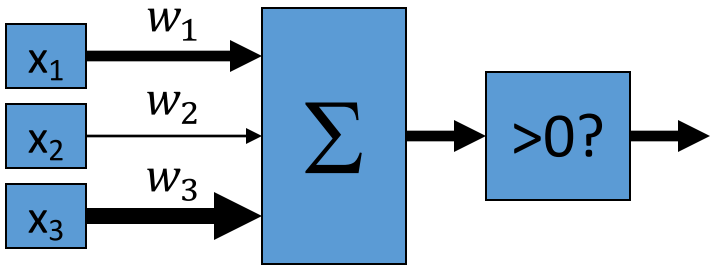
Linear Classifiers

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



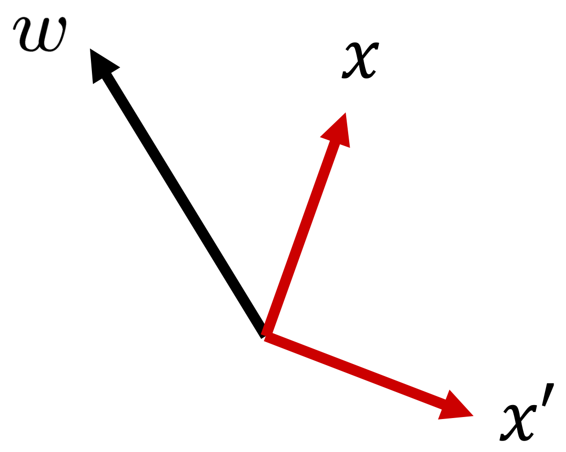
$$activation_w(x) = \sum_j w_j x_j = w \cdot x$$

- 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

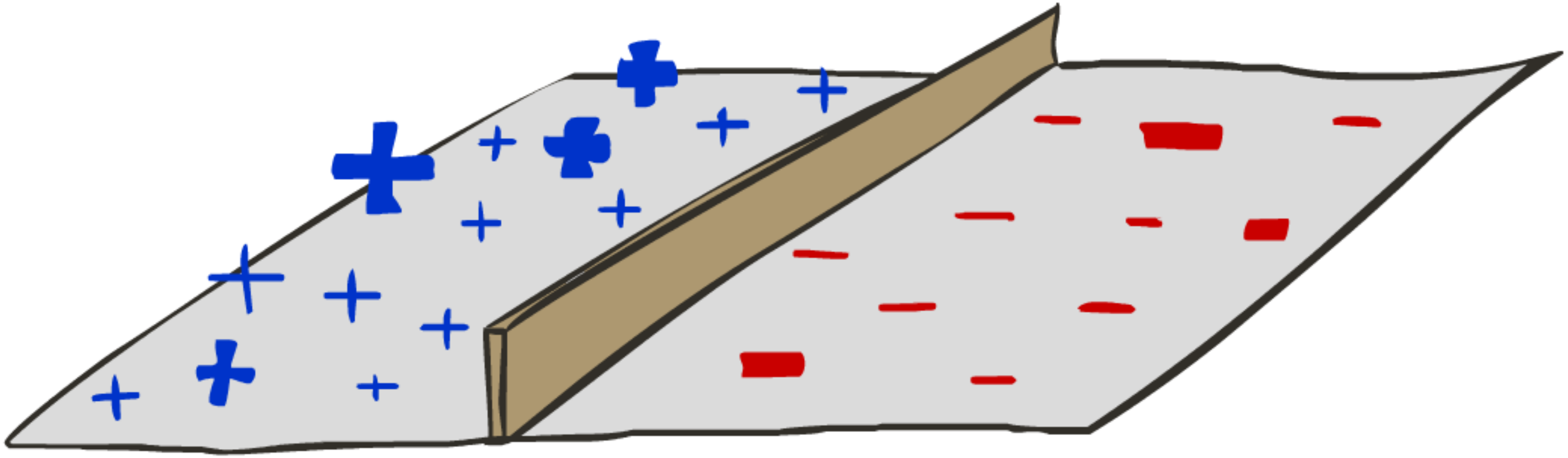



```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

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

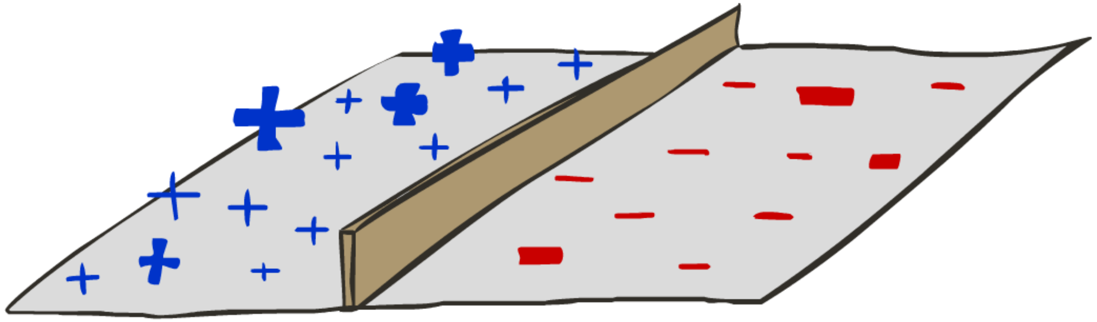
Dot product $w \cdot x$ *positive means the positive class*

Decision Rules



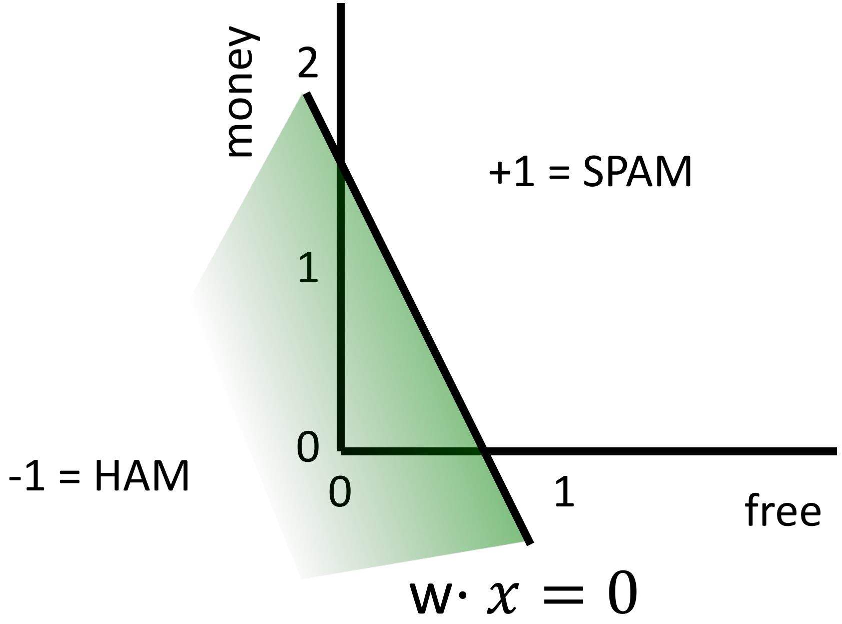
Binary Decision Rule

- 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

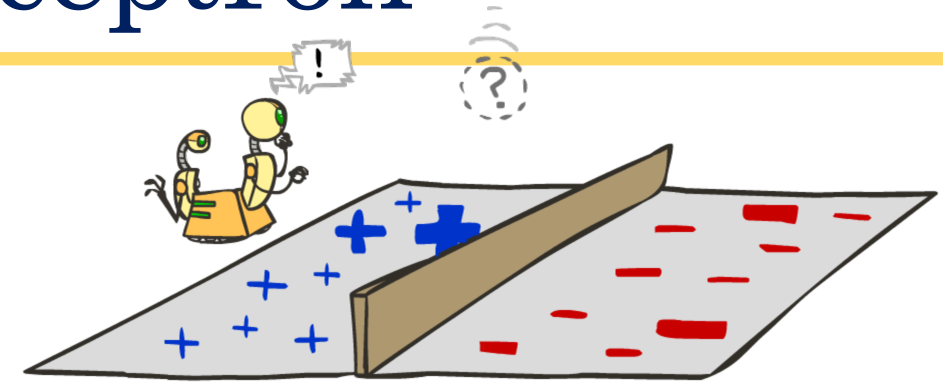
```
BIAS  : -3
free  :  4
money :  2
...
```



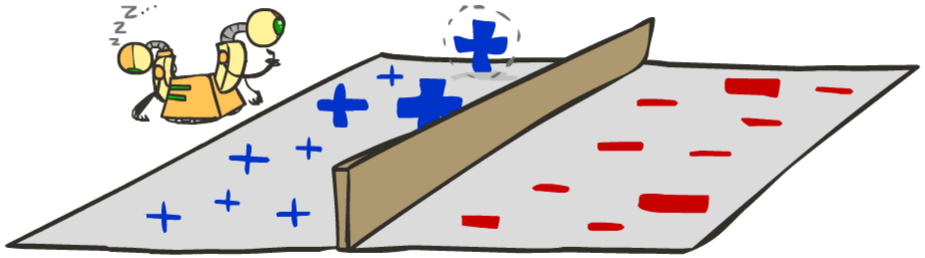
Weight Updates

Learning: Binary Perceptron

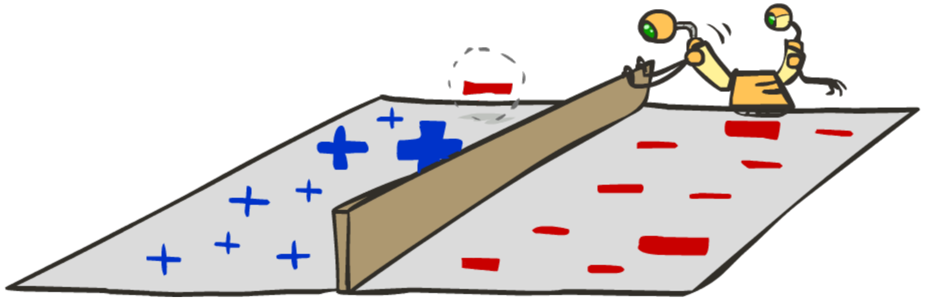
- Start with weights = 0
- For each training instance:
  - Classify with current weights



  - If correct (i.e., $\hat{y} = y$), no change!



  - If wrong: adjust the weight vector



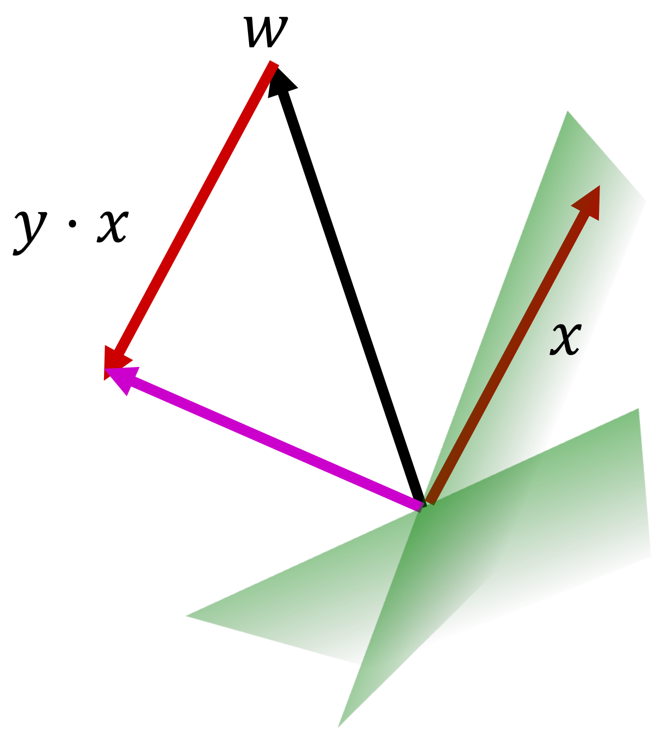
Learning: Binary Perceptron

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

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

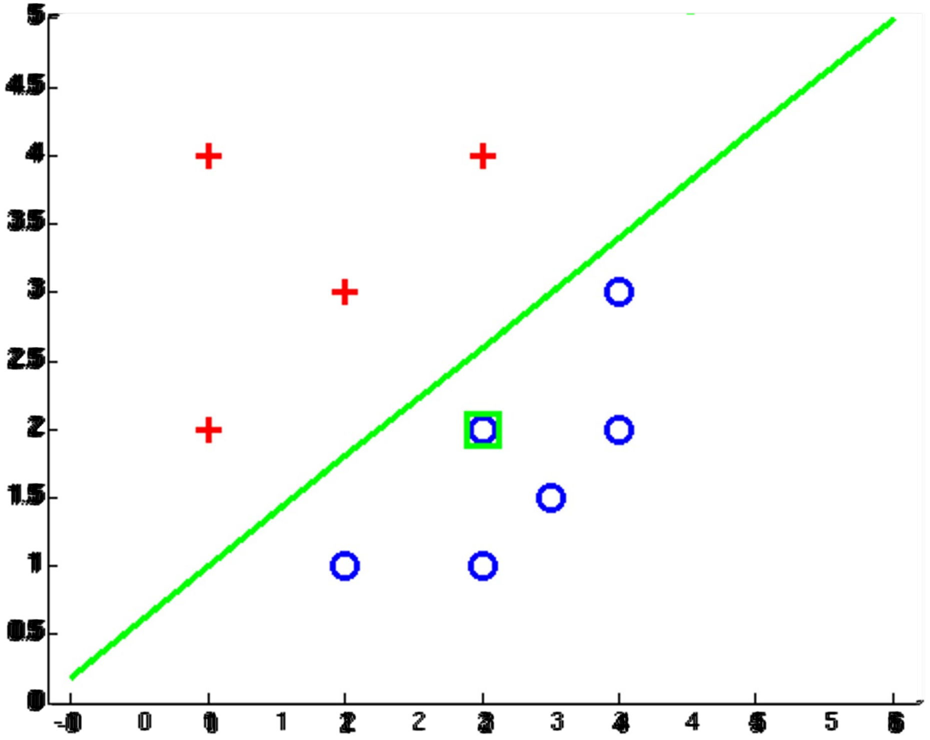
  - If correct (i.e., $\hat{y} = y$), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y is -1.

$$w = w + y \cdot x$$



Examples: Perceptron

- Separable Case



Multiclass Decision Rule

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

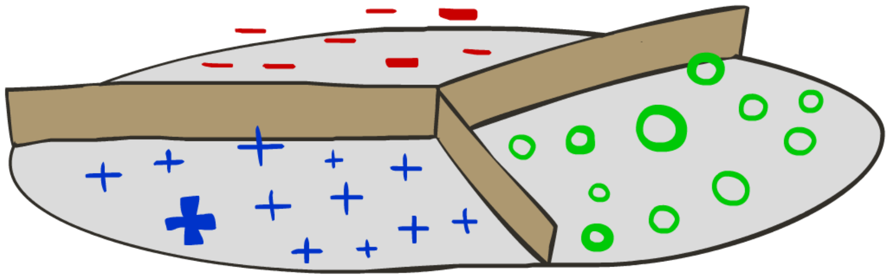
$$\boxed{w_y}$$

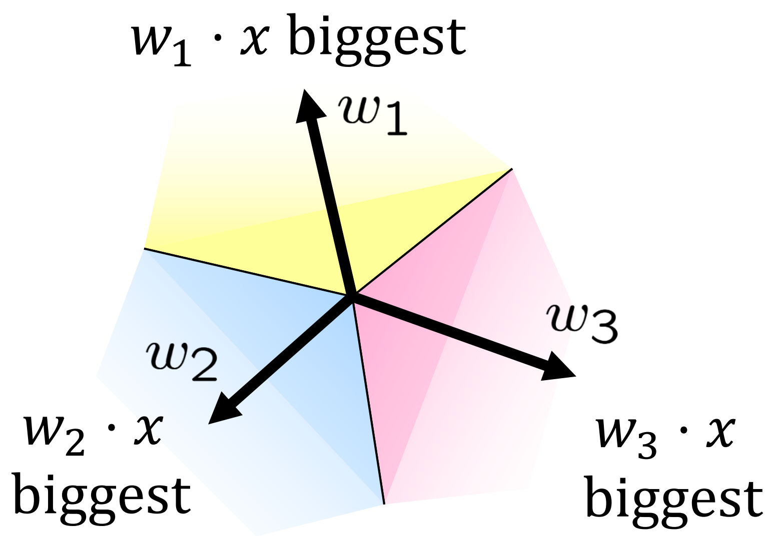
  - Score (activation) of a class y:

$$\boxed{w_y \cdot x}$$

  - Prediction highest score wins

$$\boxed{y = \arg\max_y w_y \cdot x}$$



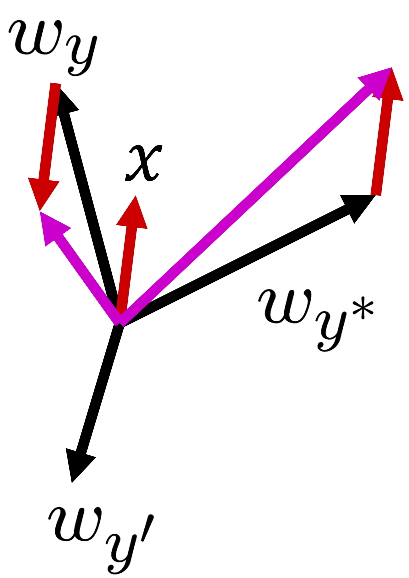


Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

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

$$\boxed{y = \arg\max_{y} w_y \cdot x}$$

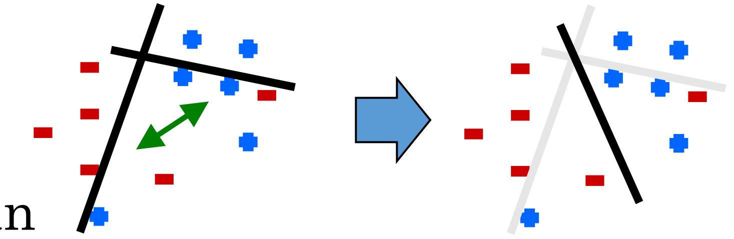


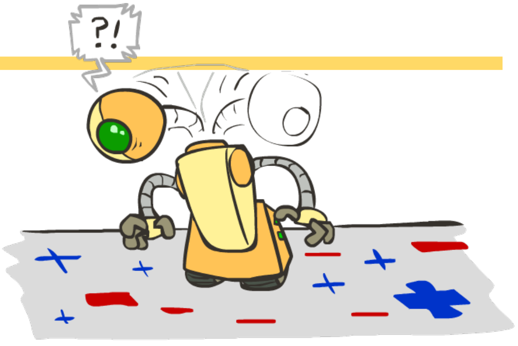
- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - x$$
$$w_{y^*} = w_{y^*} + x$$

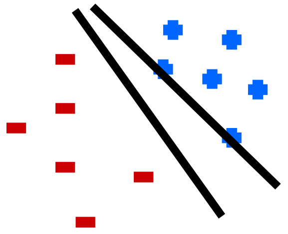
Problems with the Perceptron

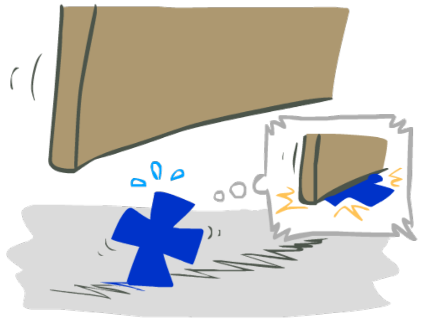
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



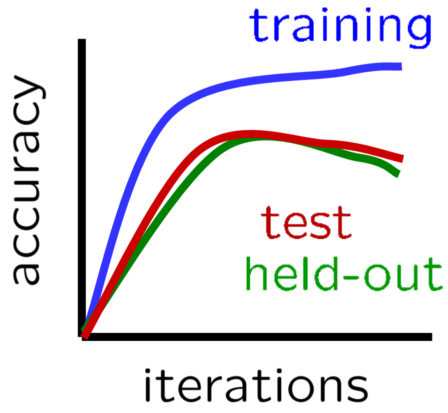


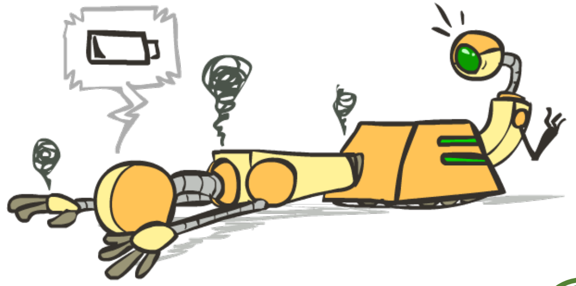
- Mediocre generalization: finds a "barely" separating solution



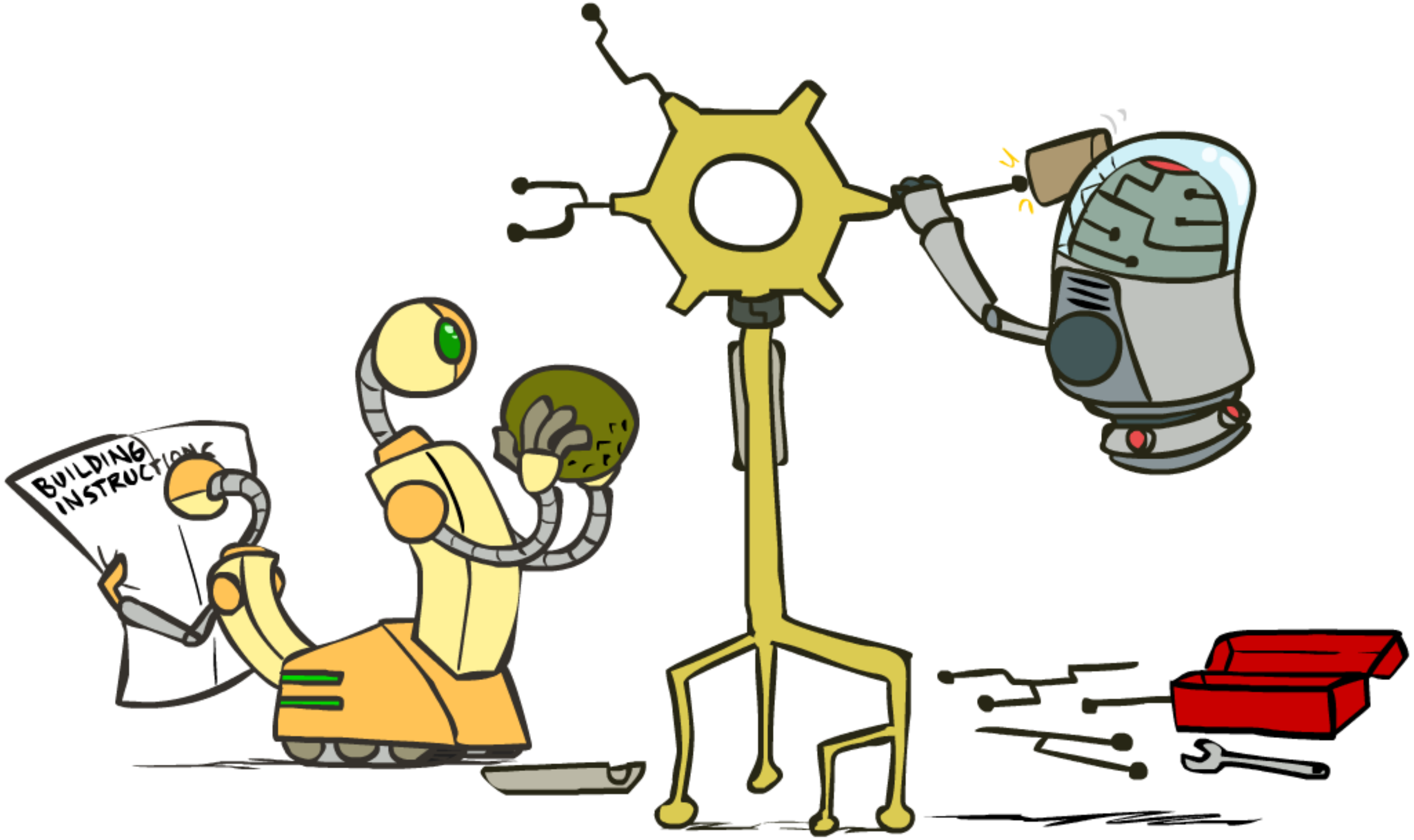


- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



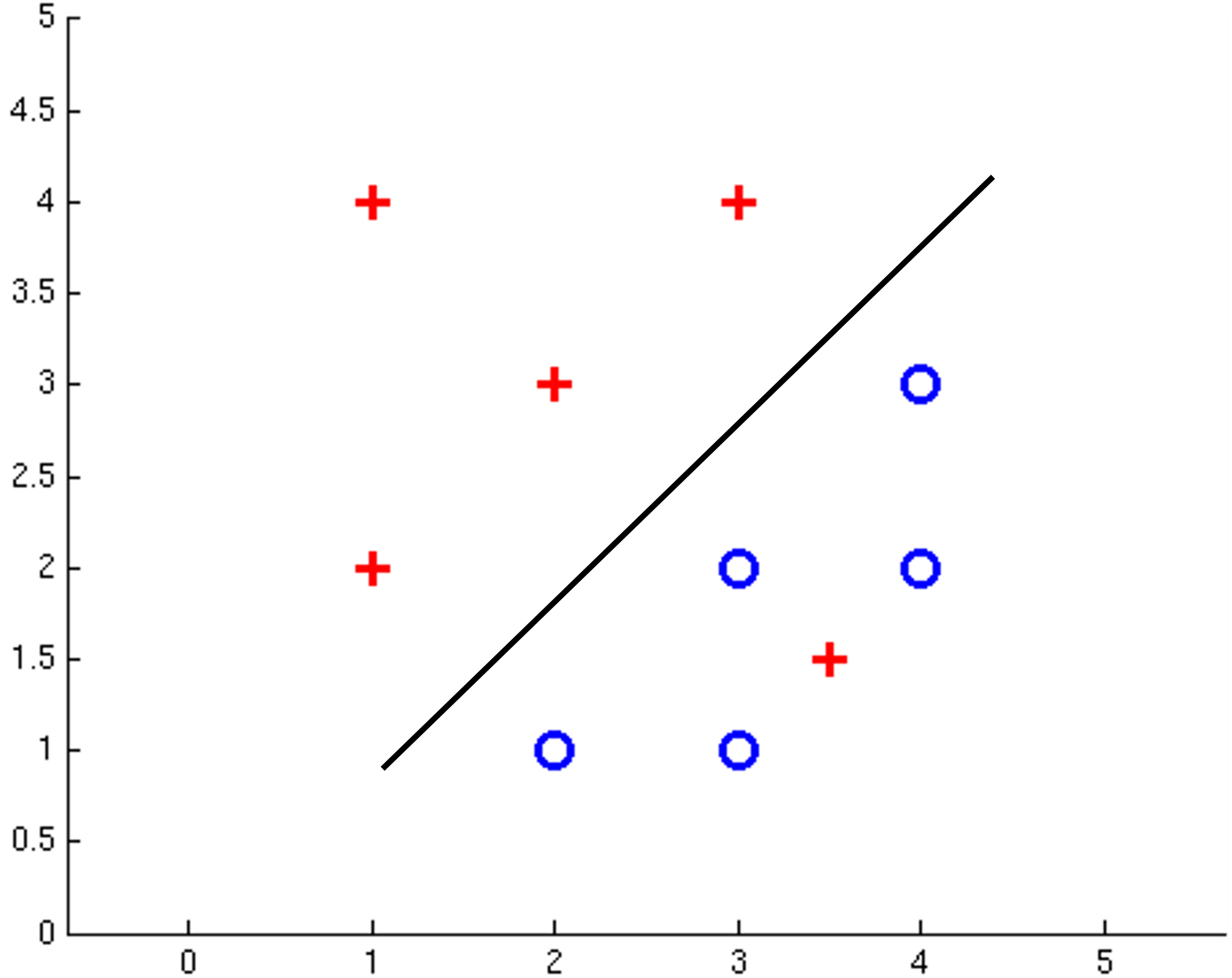


Logistic Regression

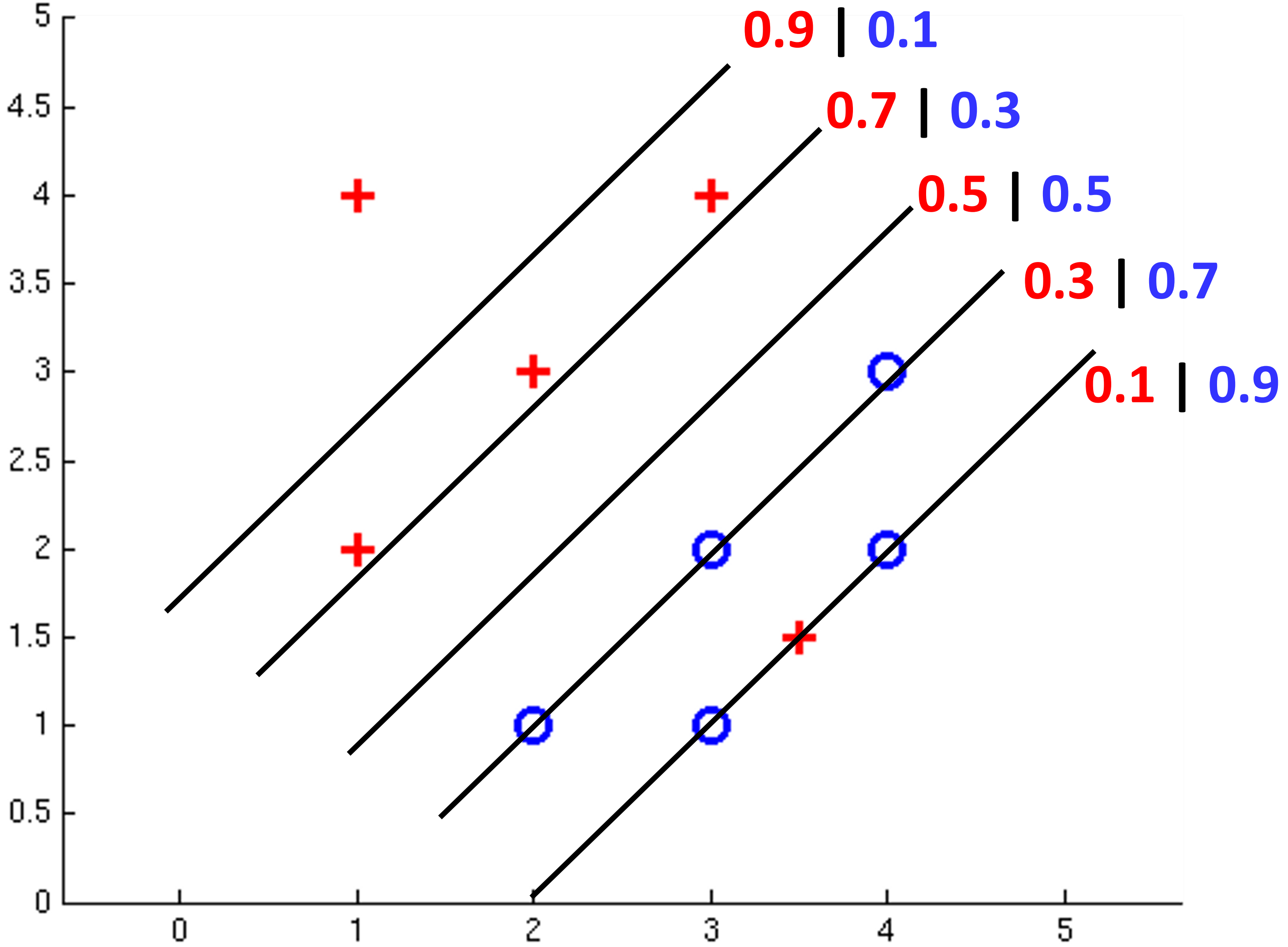


Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake



Non-Separable Case: Probabilistic Decision

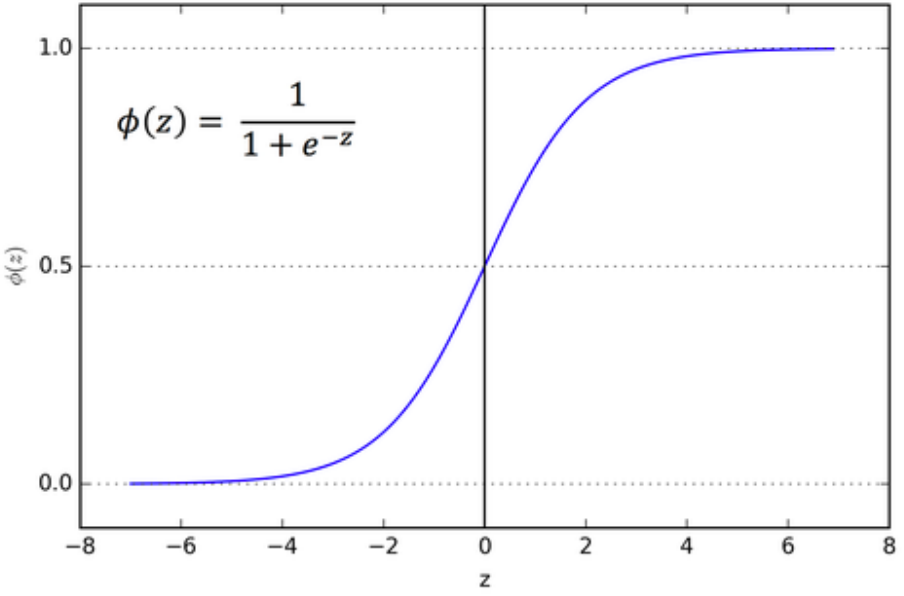


How to get probabilistic decisions?

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

- Sigmoid function

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

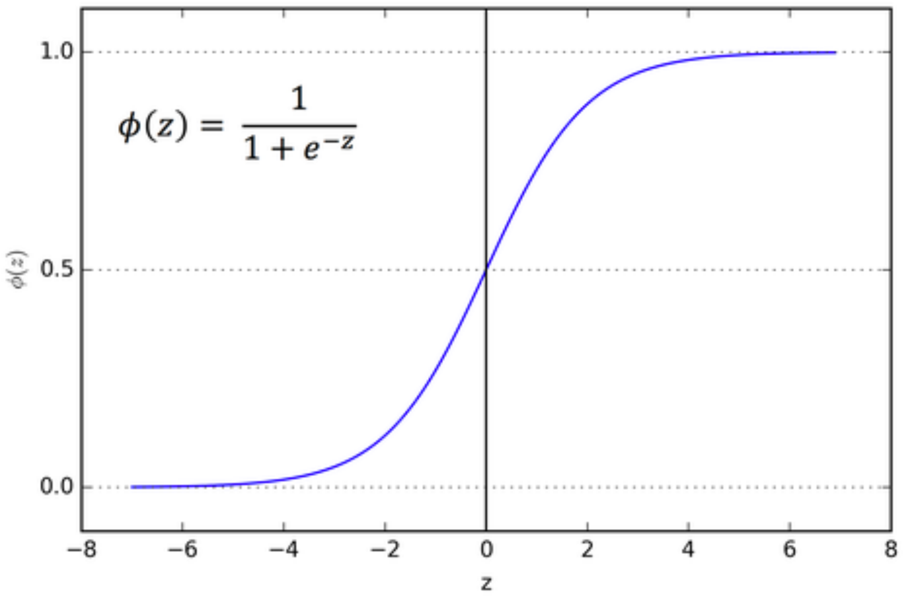


How to get probabilistic decisions?

- Sigmoid function

$$z = w \cdot x$$

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

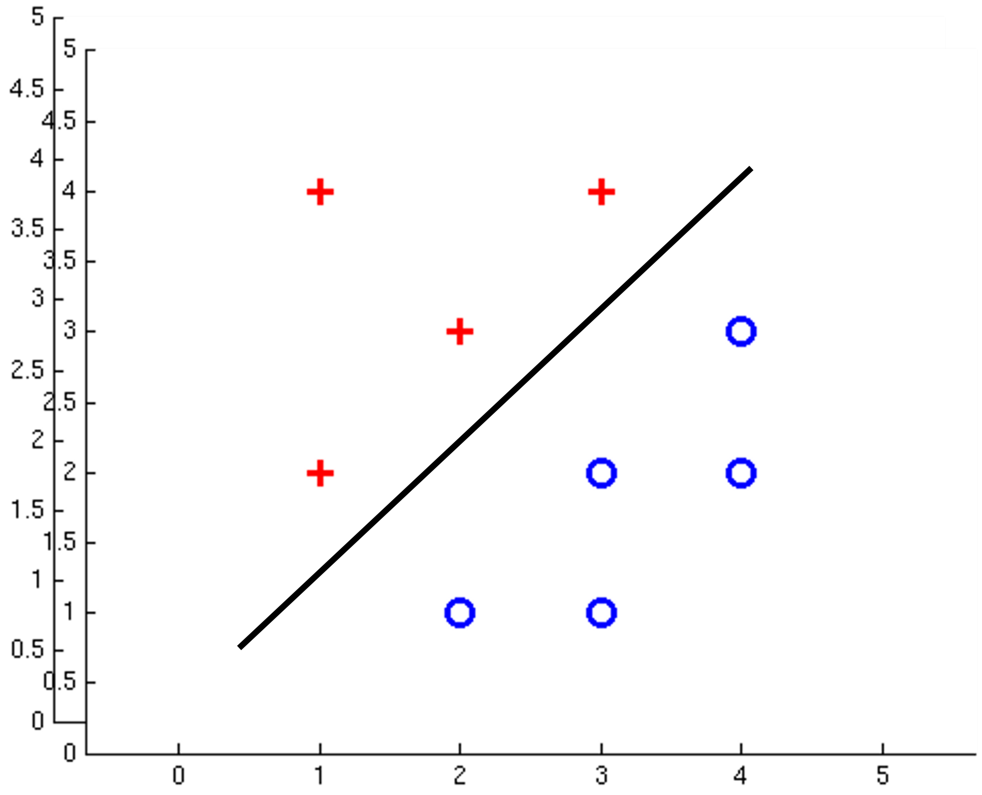


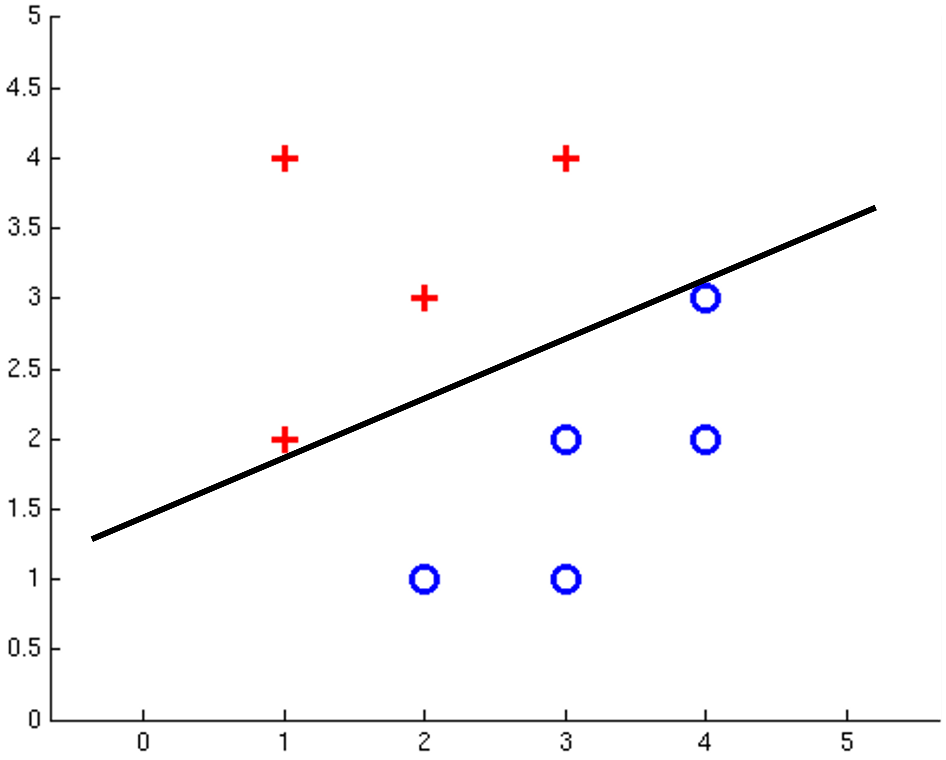
- Probabilistic decision

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

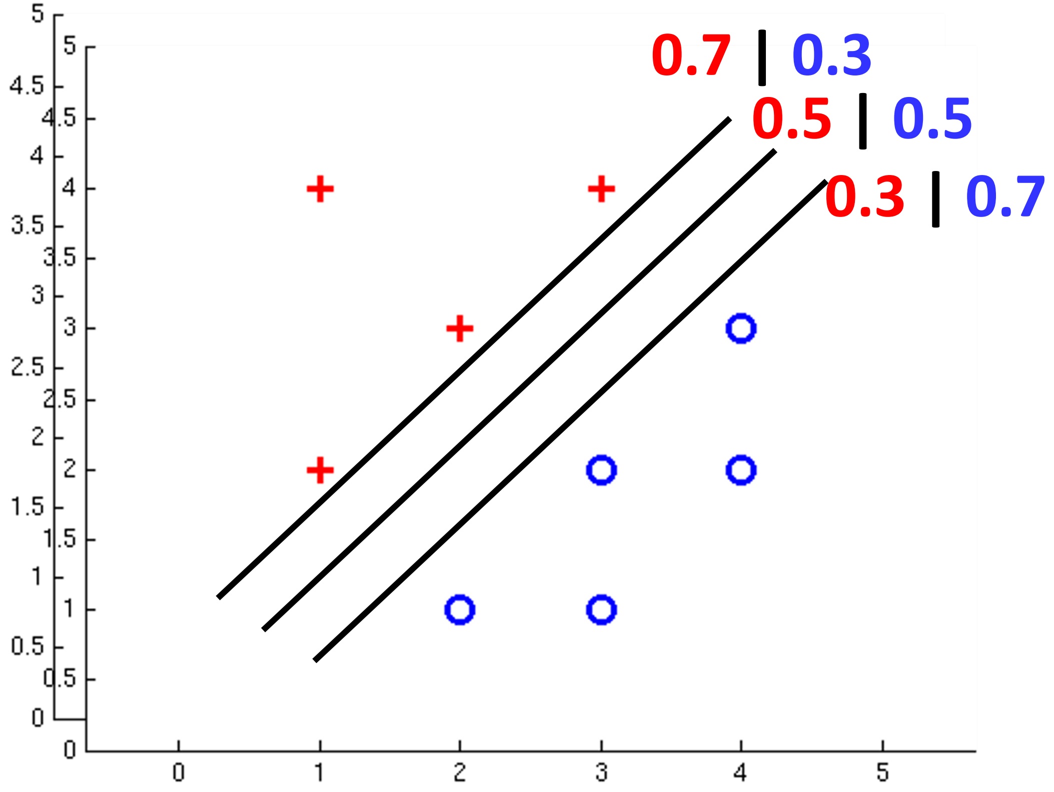
$$P\left(y^{(i)} = -1 | x^{(i)}; w\right) = 1 - \frac{1}{1 + e^{-w \cdot x^{(i)}}}$$

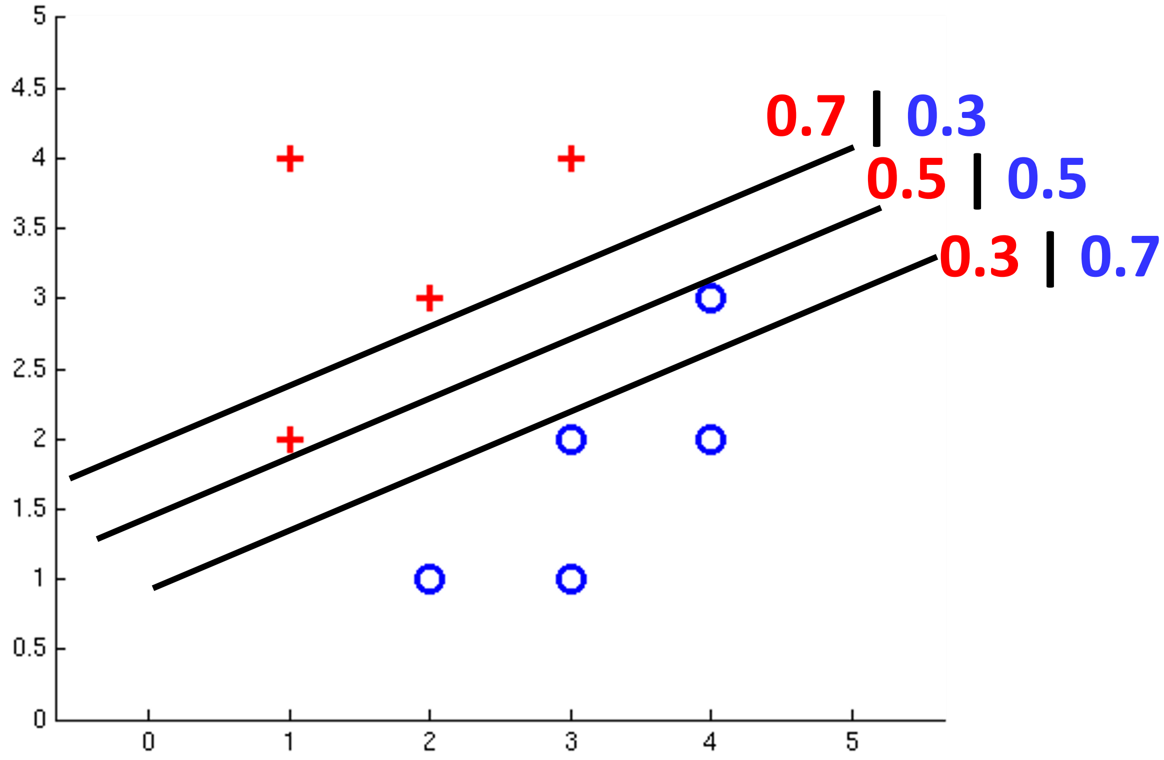
Separable Case: Deterministic Decision – Many Options





Separable Case: Probabilistic Decision – Clear Preference





Determine Best Weights: Maximum Likelihood Estimation

- Maximum likelihood estimation: choose w so as to make the data as high probability as possible.

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

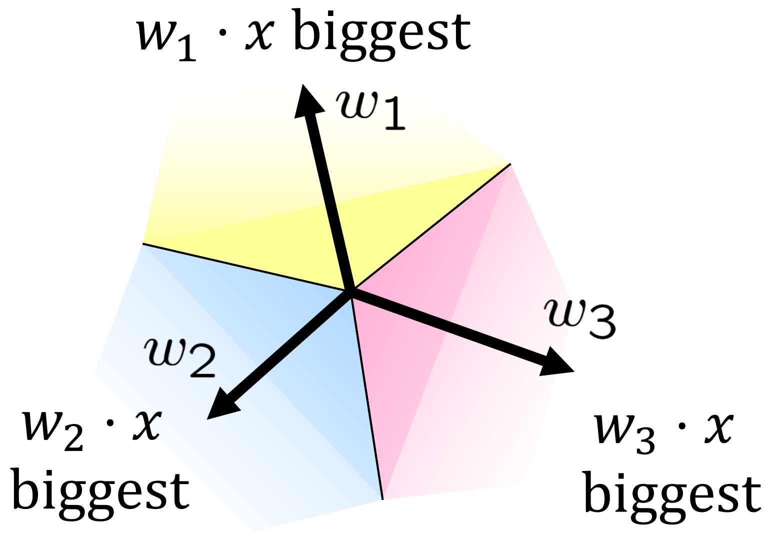
with:

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

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

Multiclass Logistic Regression

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



- 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:

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

with the likelihood of each data point:

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

Best w?

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

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

  - No closed-form expression of optimal *w*

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:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

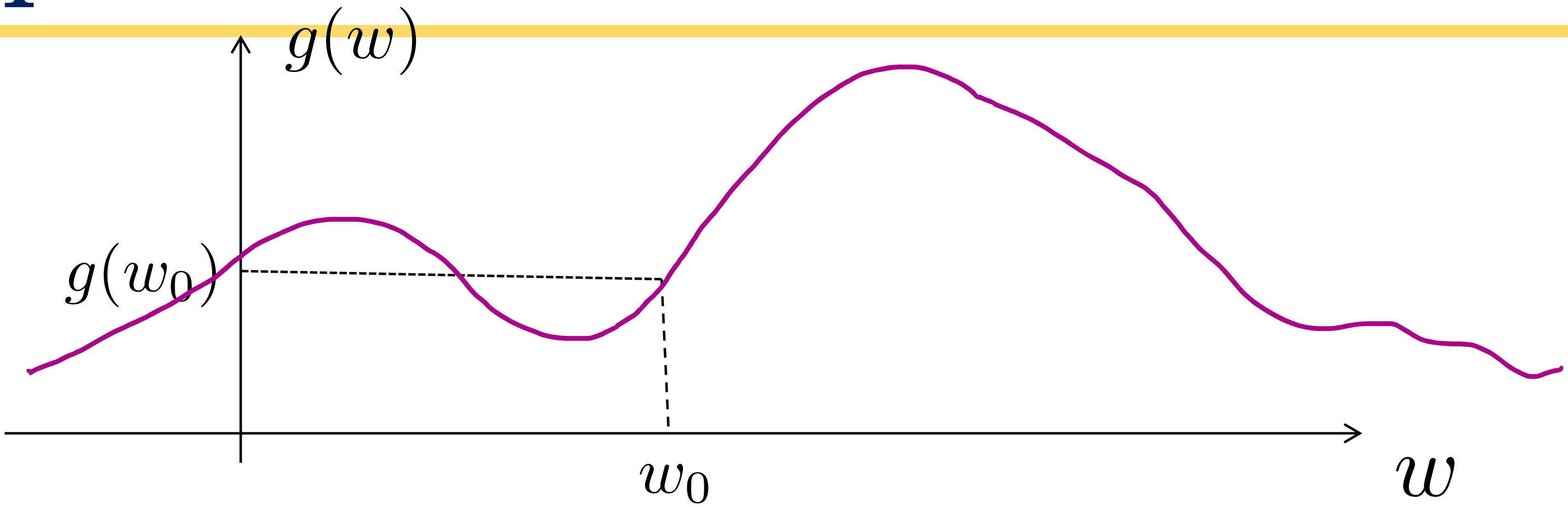
$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

- 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**

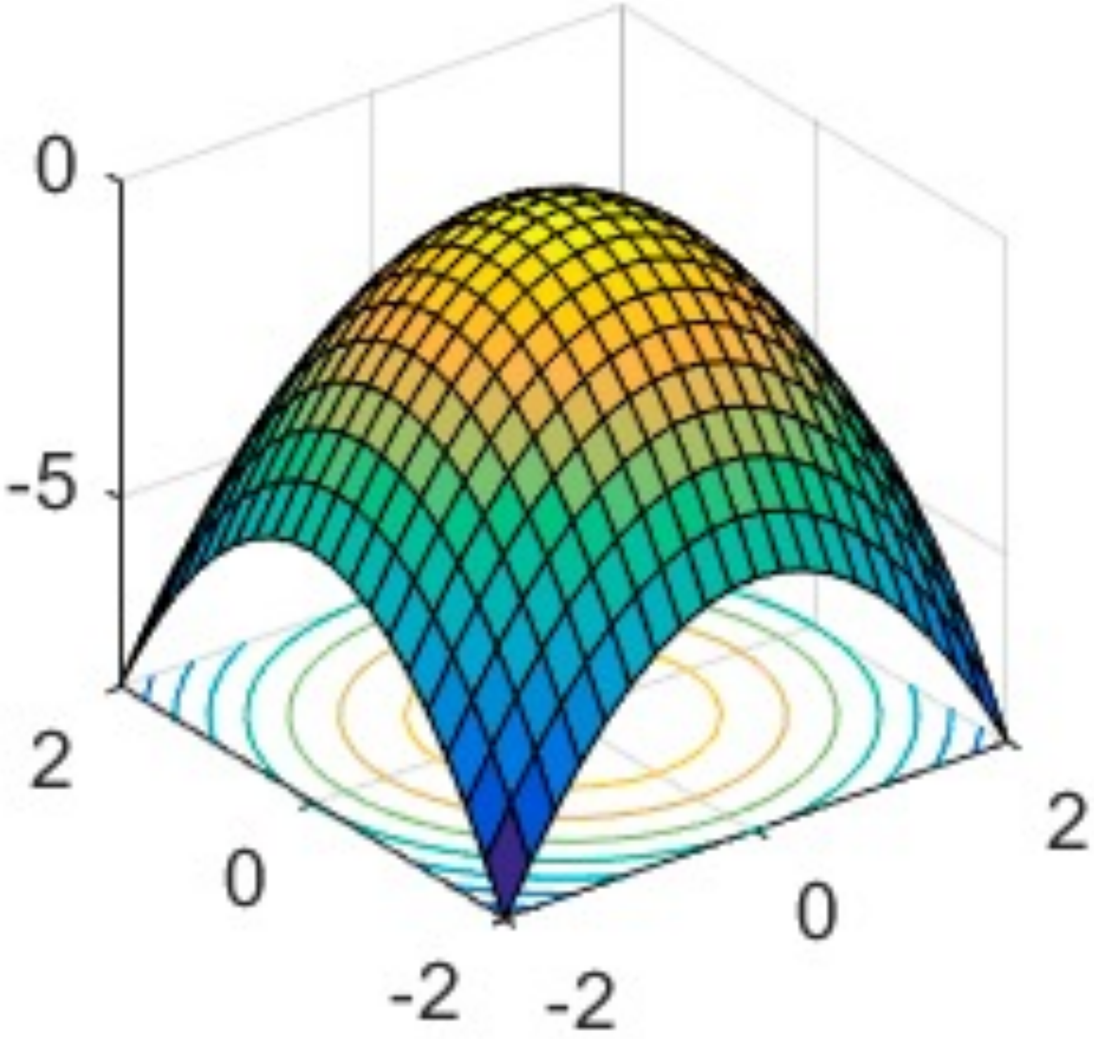
1-D Optimization



- 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**