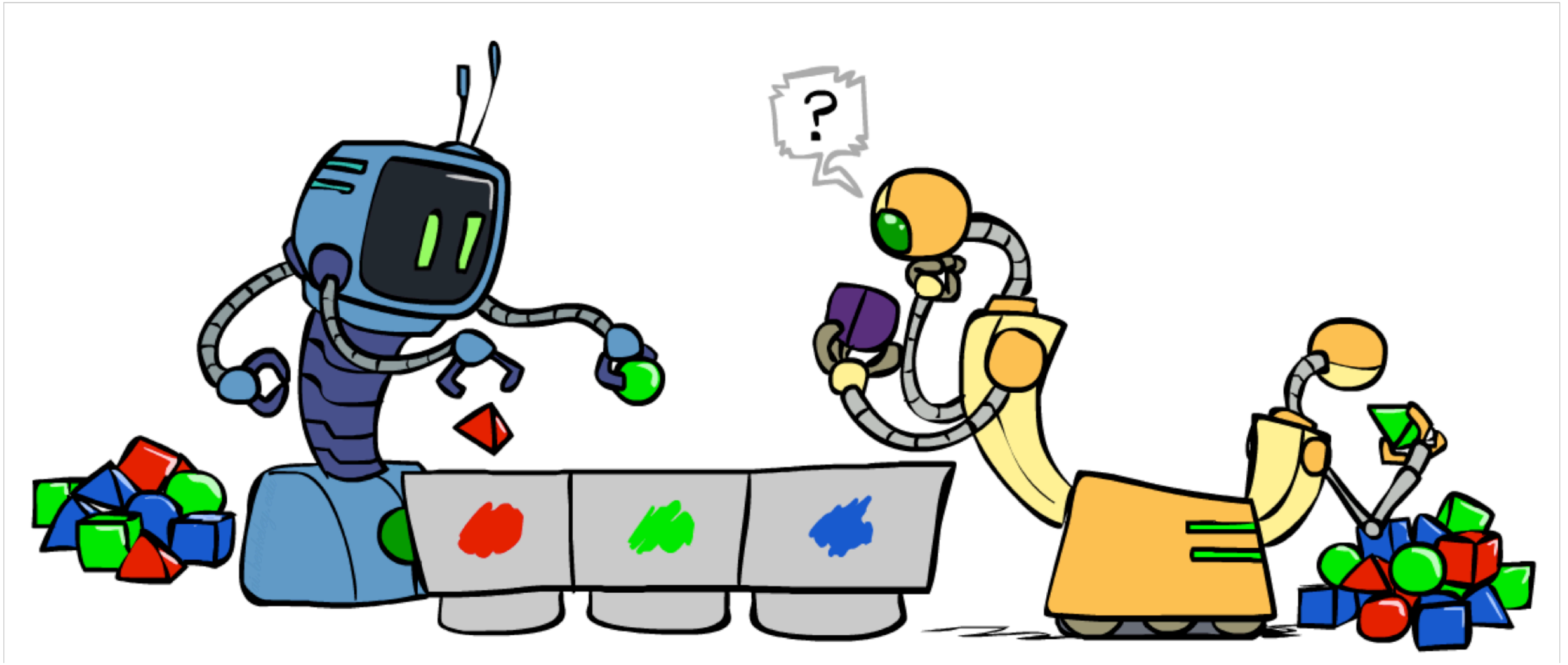
Clustering

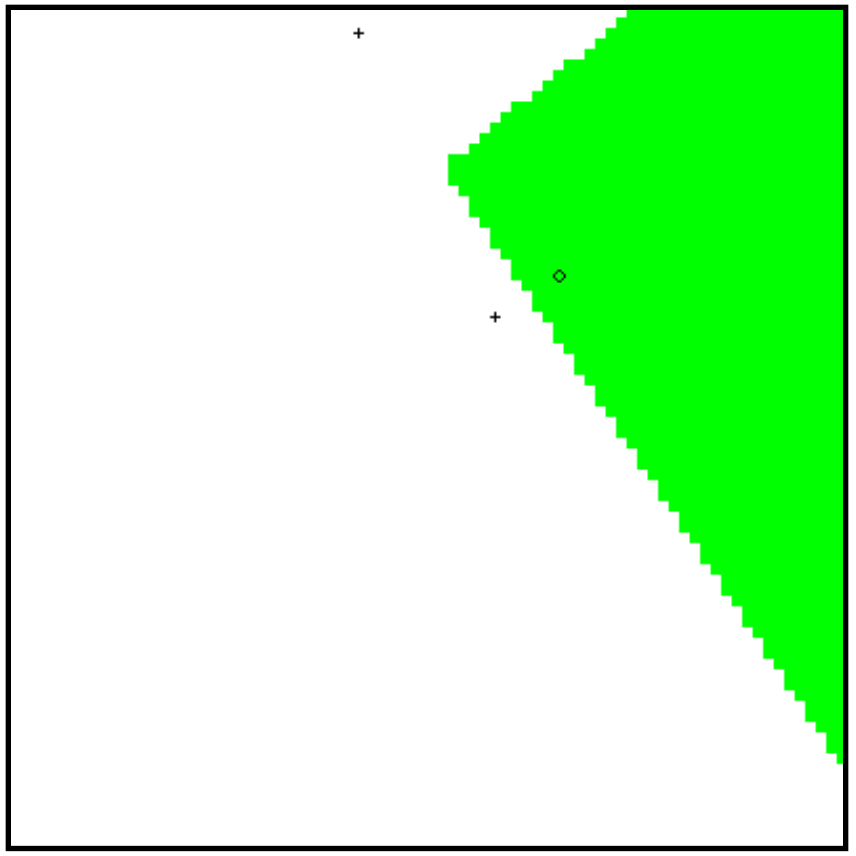


Slides Courtesy of Dan Klein and Pieter Abbeel --- University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Recap: similarity based classification

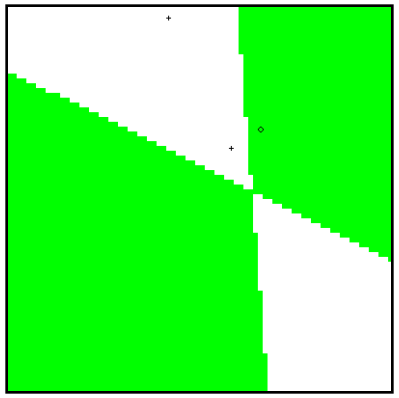
- Classification from similarity
  - Case-based reasoning
  - Predict an instance's label using similar instances
- Nearest-neighbor classification
  - 1-NN: copy the label of the most similar data point
  - K-NN: vote the k nearest neighbors (need a weighting scheme)
  - Key issue: how to define similarity
  - Trade-offs: Small k gives relevant neighbors, Large k gives smoother functions



http://www.cs.cmu.edu/~zhuxj/courseproject/knndemo/KNN.html

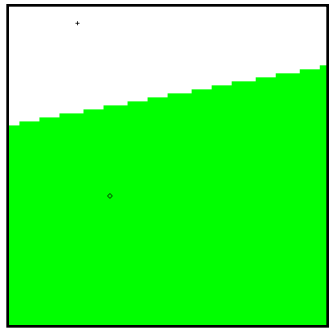
Recap: Parametric / Non-Parametric

- Parametric models:
  - Fixed set of parameters
  - More data means better settings
- Non-parametric models:
  - Complexity of the classifier increases with data
  - Better in the limit, often worse in the non-limit
- (K)NN is non-parametric

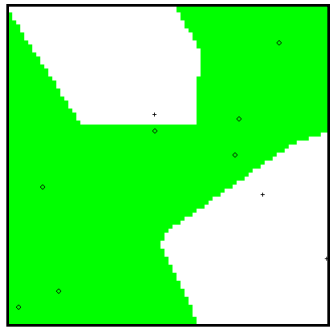


Truth

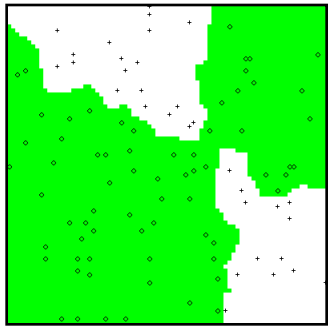
2 Examples



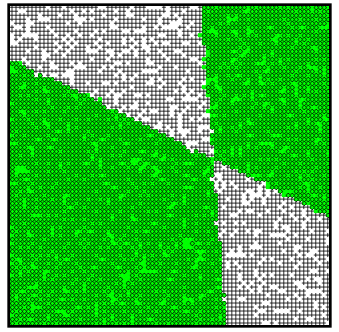
10 Examples



100 Examples



10000 Examples



Recap: Nearest-Neighbor Classification

- Nearest neighbor for digits:
  - Take new image
  - Compare to all training images
  - Assign based on closest example


0


1


2


0


1


2

- Encoding: image is vector of intensities:

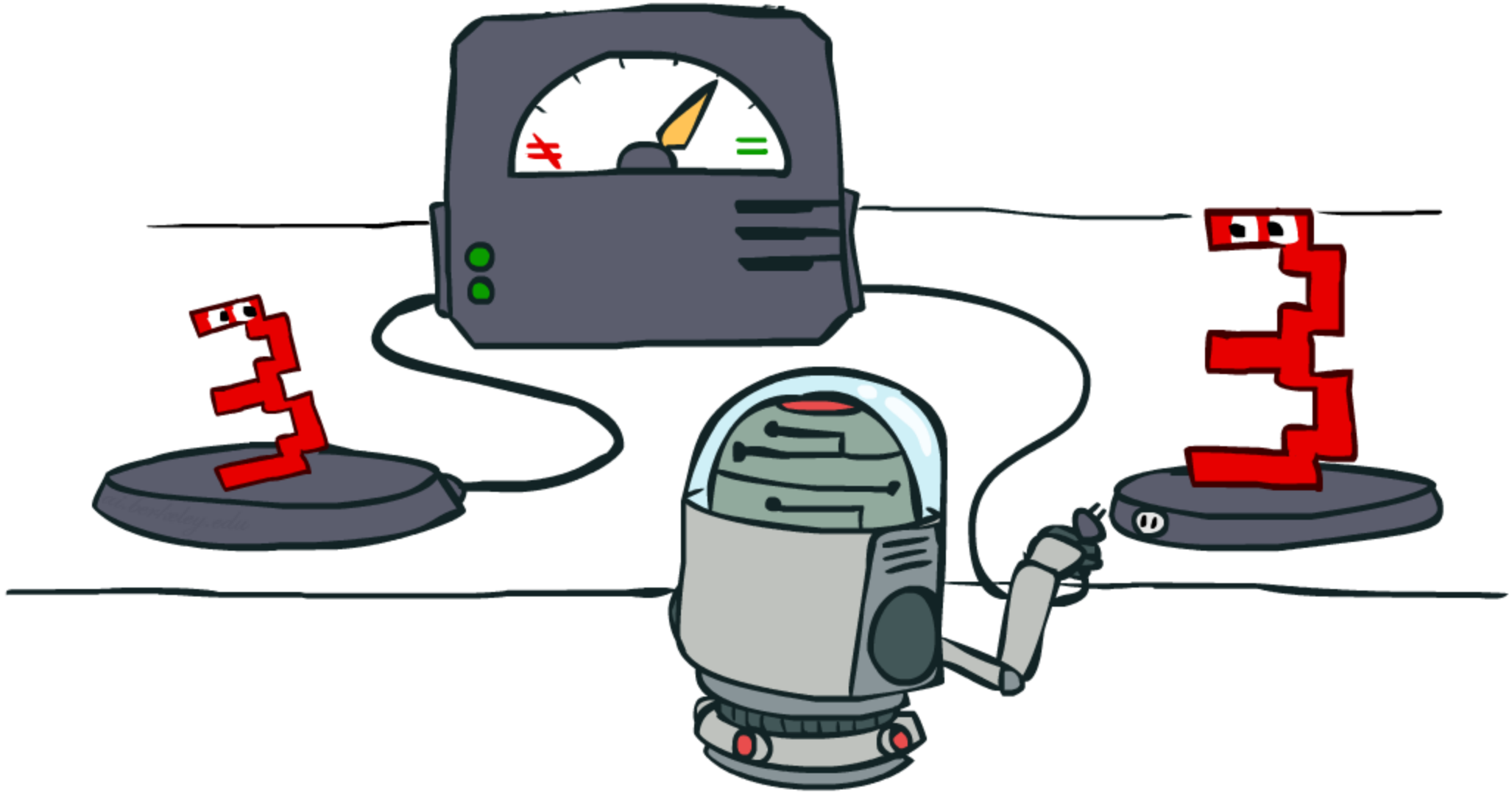
 $= \langle 0.0 \;\; 0.0 \;\; 0.3 \;\; 0.8 \;\; 0.7 \;\; 0.1 \ldots 0.0 \rangle$

- What's the similarity function?
  - Dot product of two images vectors?

$$\text{sim}(x, x') = x \cdot x' = \sum_i x_i x'_i$$

  - Usually normalize vectors so ||x|| = 1
  - min = 0 (when?), max = 1 (when?)

Recap: Similarity Functions



Recap: Dual Perceptron

- What is the final value of a weight w_y of a perceptron?
  - Can it be any real vector?
  - No! It's built by adding up inputs.

$$w_y = \mathbf{0} + f(x_1) - f(x_5) + \dots$$

$$w_y = \sum_i \alpha_{i,y} f(x_i)$$

- Can reconstruct weight vectors (the primal representation) from update counts (the dual representation)

$$\alpha_y = \langle \alpha_{1,y} \;\; \alpha_{2,y} \;\; \dots \;\; \alpha_{n,y} \rangle$$

Recap: Dual Perceptron

- How to classify a new example x?

$$\begin{aligned}
\text{score}(y, x) &= w_y \cdot f(x) \\
&= \left(\sum_i \alpha_{i,y} f(x_i) \right) \cdot f(x) \\
&= \sum_i \alpha_{i,y} \left(f(x_i) \cdot f(x) \right) \\
&= \sum_i \alpha_{i,y} K(x_i, x)
\end{aligned}$$

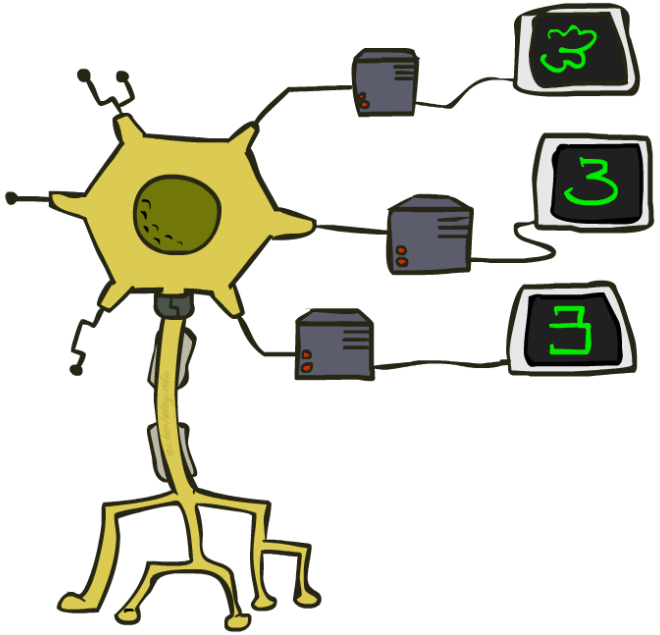
- If someone tells us the value of K for each pair of examples, never need to build the weight vectors (or the feature vectors)!

Recap: Kernelized Perceptron

- If we had a black box (kernel) K that told us the dot product of two examples x and x':
  - Could work entirely with the dual representation
  - No need to ever take dot products ("kernel trick")

$$\text{score}(y, x) = w_y \cdot f(x)$$

$$= \sum_i \alpha_{i,y} \, K(x_i, x)$$



- Like nearest neighbor – work with black-box similarities

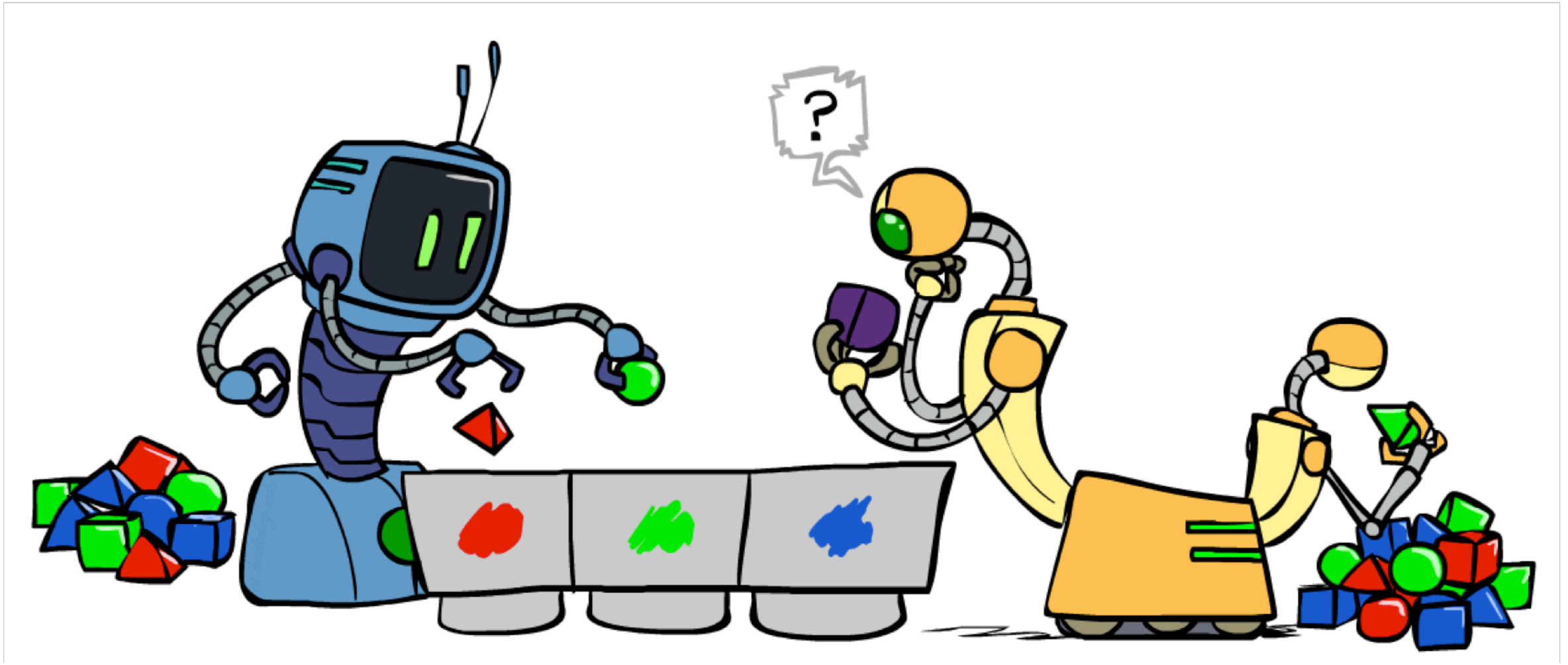
Recap: Classification

- Classification systems:
  - Supervised learning
  - Make a prediction given evidence
  - We've seen several methods for this
  - Useful when you have labeled data

Clustering

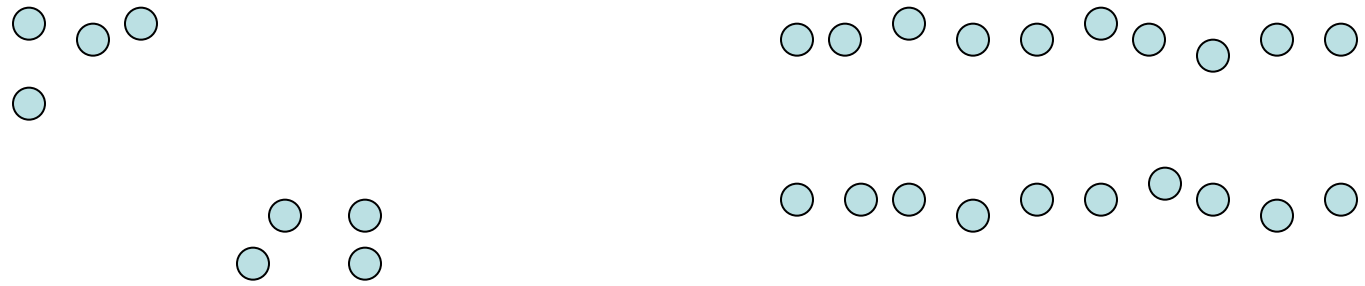
- Clustering systems:
  - Unsupervised learning
  - Detect patterns in unlabeled data
    - E.g. group emails or search results
    - E.g. find categories of customers
    - E.g. detect anomalous program executions
  - Useful when don't know what you're looking for
  - Requires data, but no labels
  - Often get gibberish

Clustering



Clustering

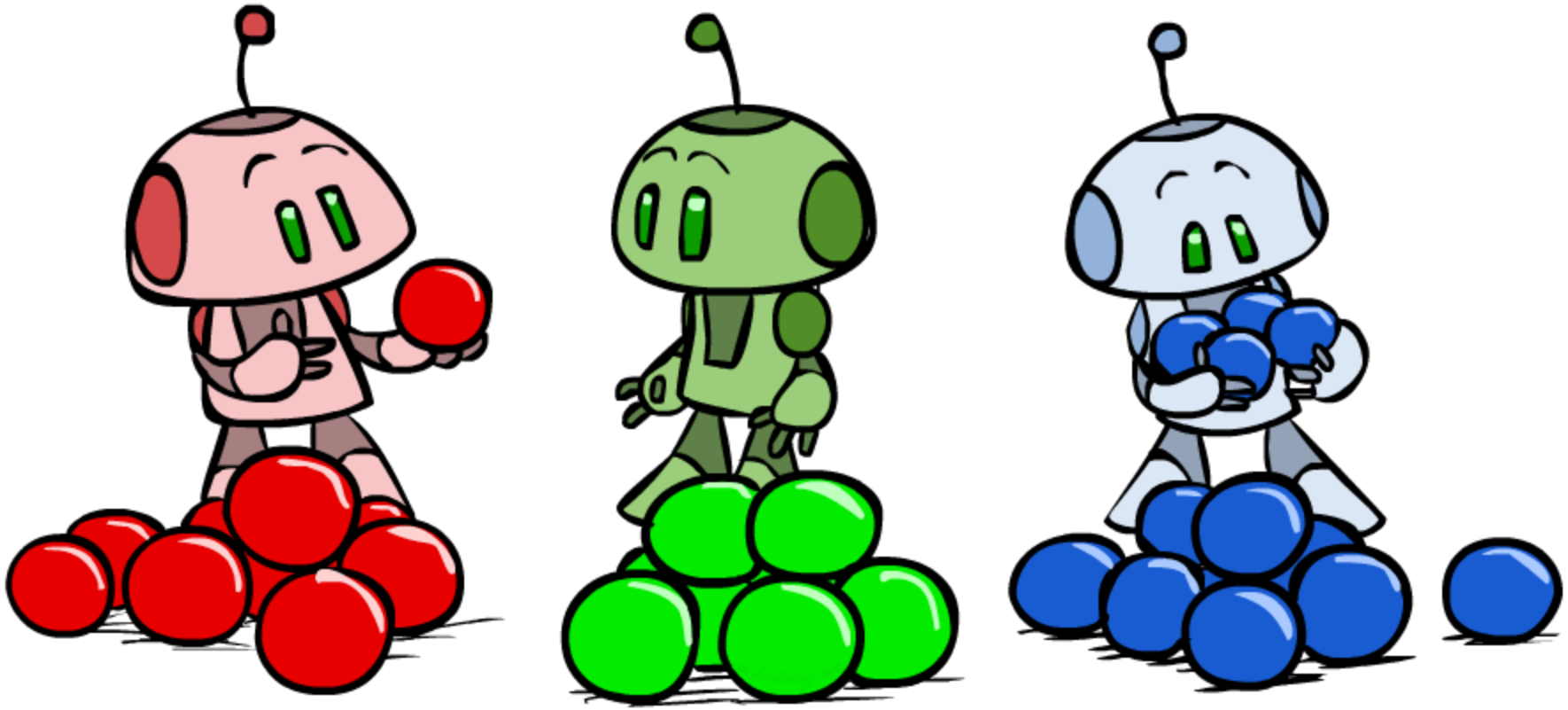
- Basic idea: group together similar instances
- Example: 2D point patterns



- What could "similar" mean?
  - One option: small (squared) Euclidean distance

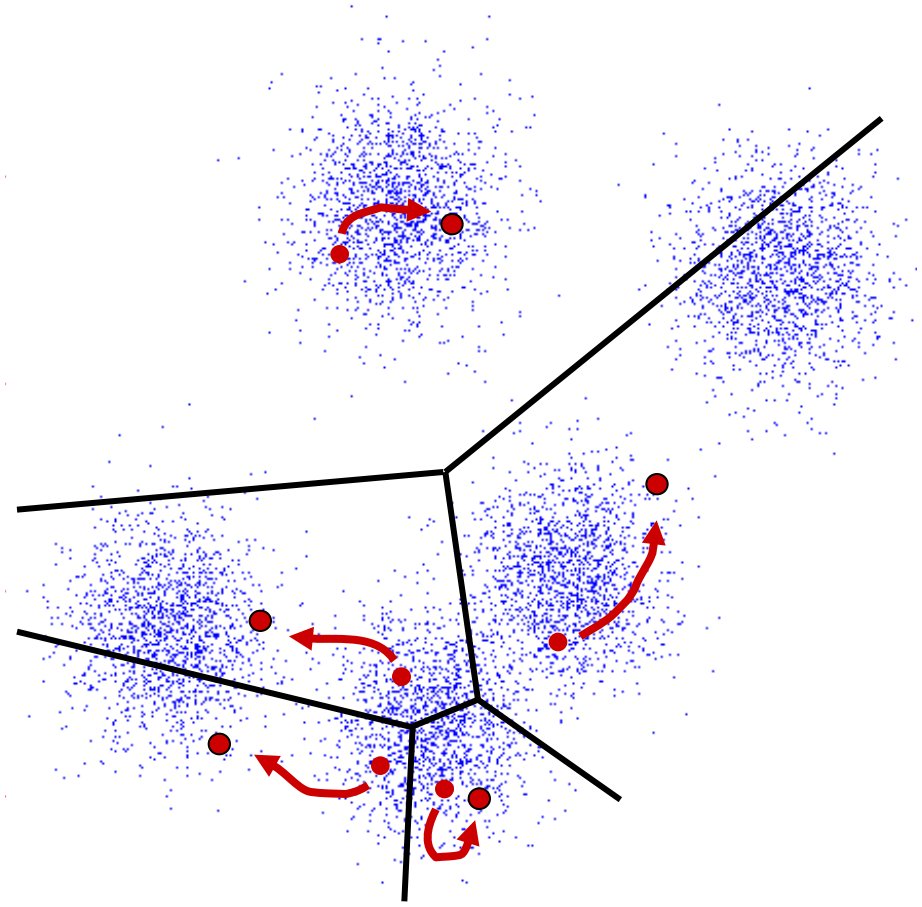
$$\text{dist}(x, y) = (x - y)^{\mathsf{T}}(x - y) = \sum_i (x_i - y_i)^2$$

K-Means

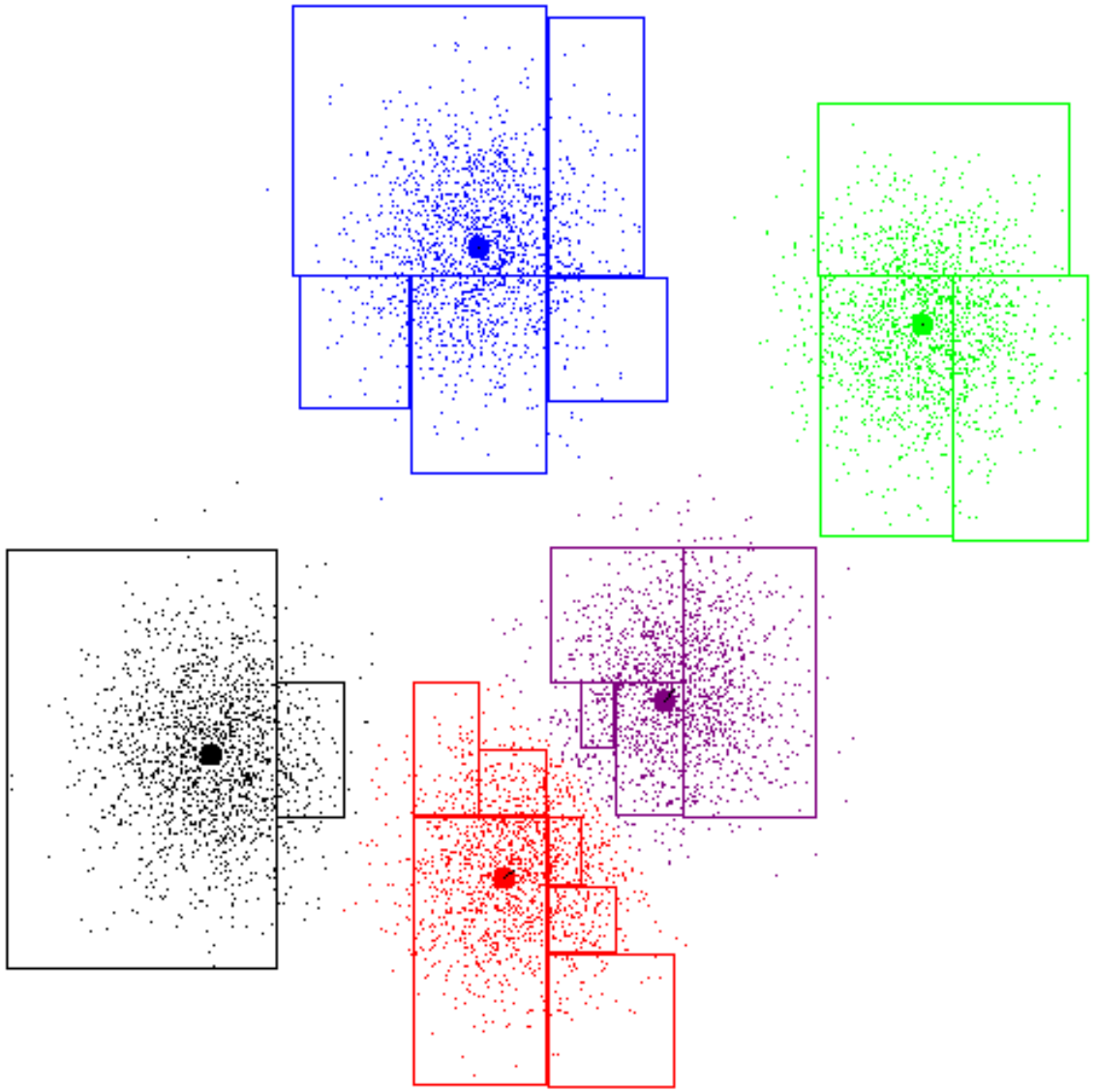


K-Means

- An iterative clustering algorithm
  - Pick K random points as cluster centers (means)
  - Alternate:
    - Assign data instances to closest mean
    - Assign each mean to the average of its assigned points
  - Stop when no points' assignments change



K-Means Example

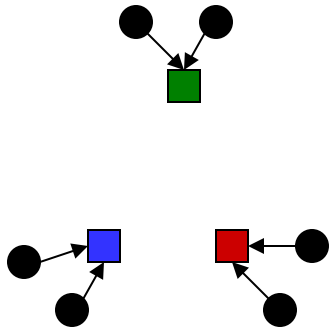


K-Means as Optimization

- Consider the total distance to the means:

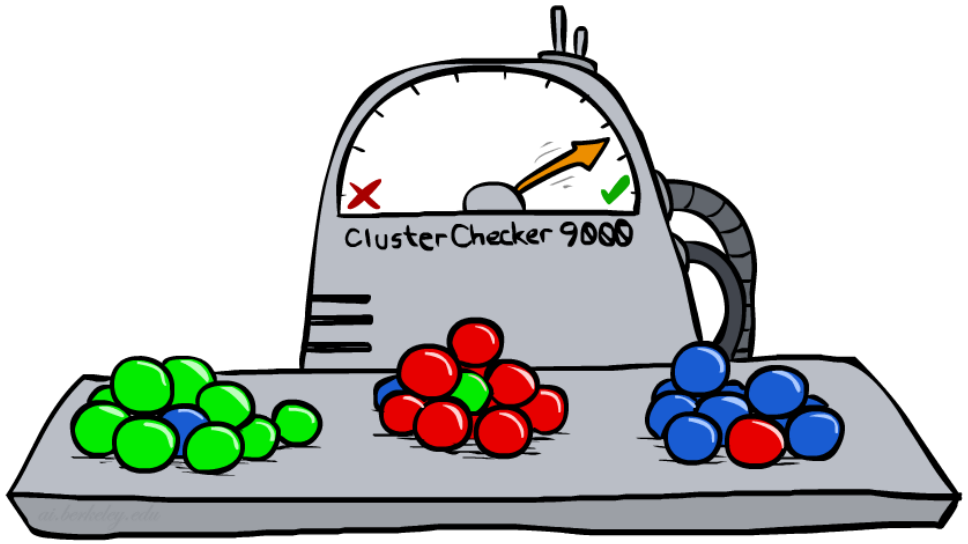
$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$

points assignments means



- Each iteration reduces phi

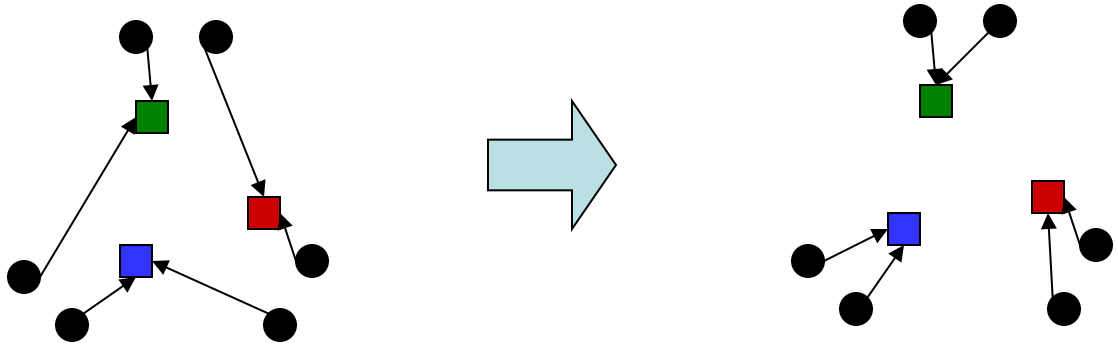
- Two stages each iteration:
  - Update assignments: fix means c, change assignments a
  - Update means: fix assignments a, change means c



Phase I: Update Assignments

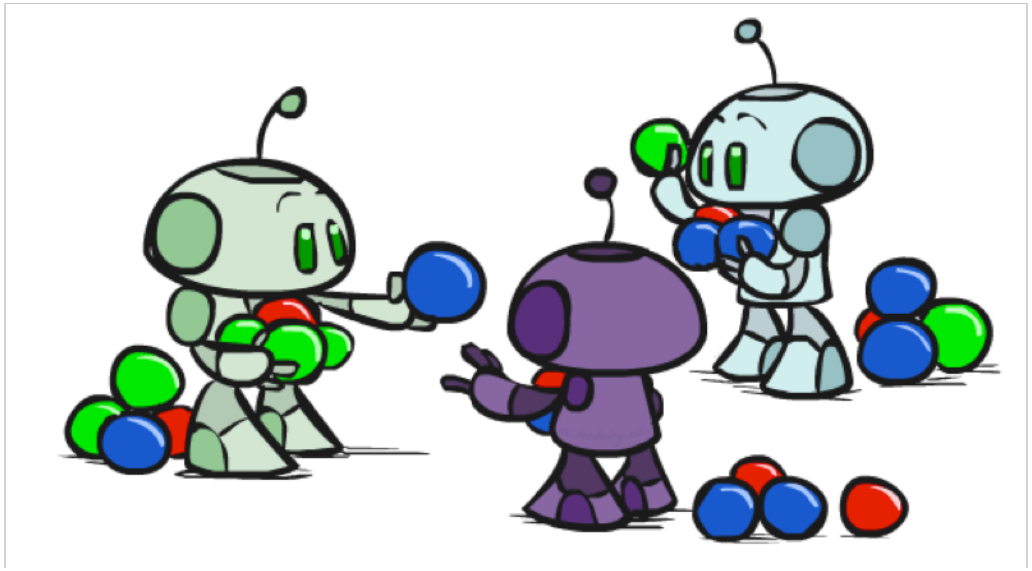
- For each point, re-assign to closest mean:

$$a_i = \arg\min_k \text{dist}(x_i, c_k)$$



- Can only decrease total distance phi!

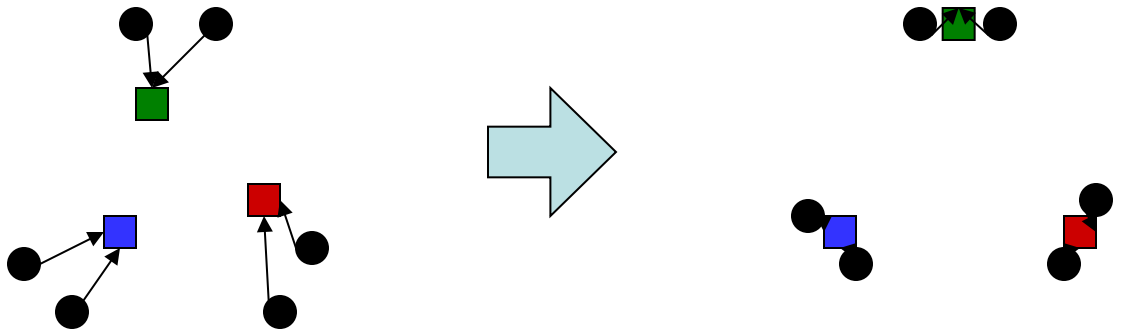
$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$



Phase II: Update Means

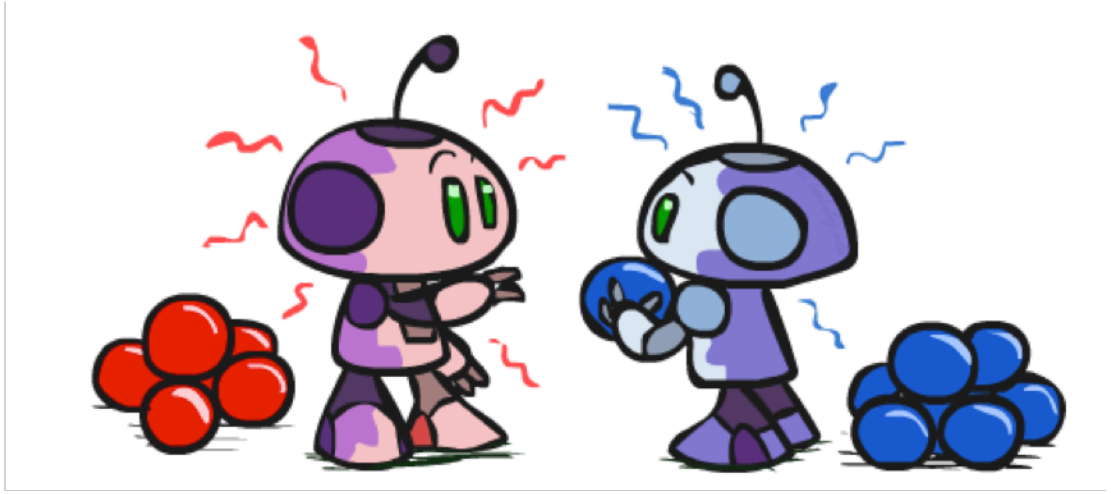
- Move each mean to the average of its assigned points:

$$c_k = \frac{1}{|\{i : a_i = k\}|} \sum_{i:a_i=k} x_i$$



- Also can only decrease total distance… (Why?)

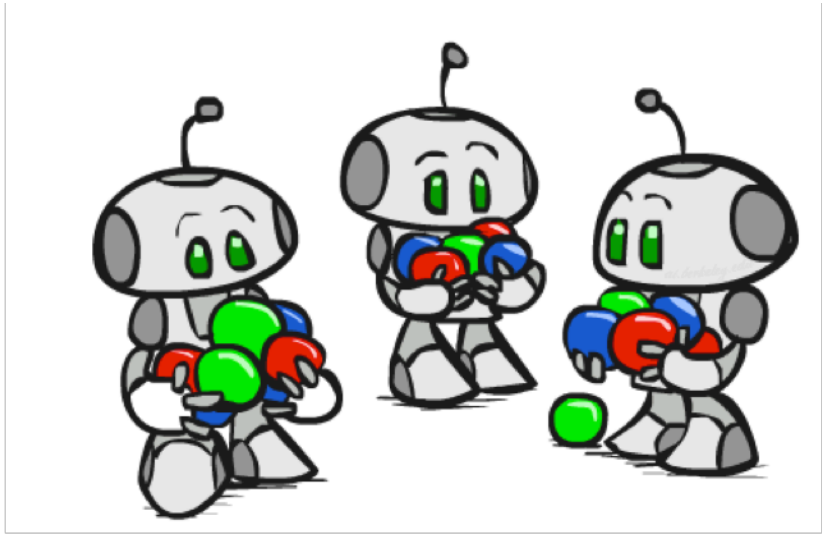
- Fun fact: the point y with minimum squared Euclidean distance to a set of points {x} is their mean



Initialization

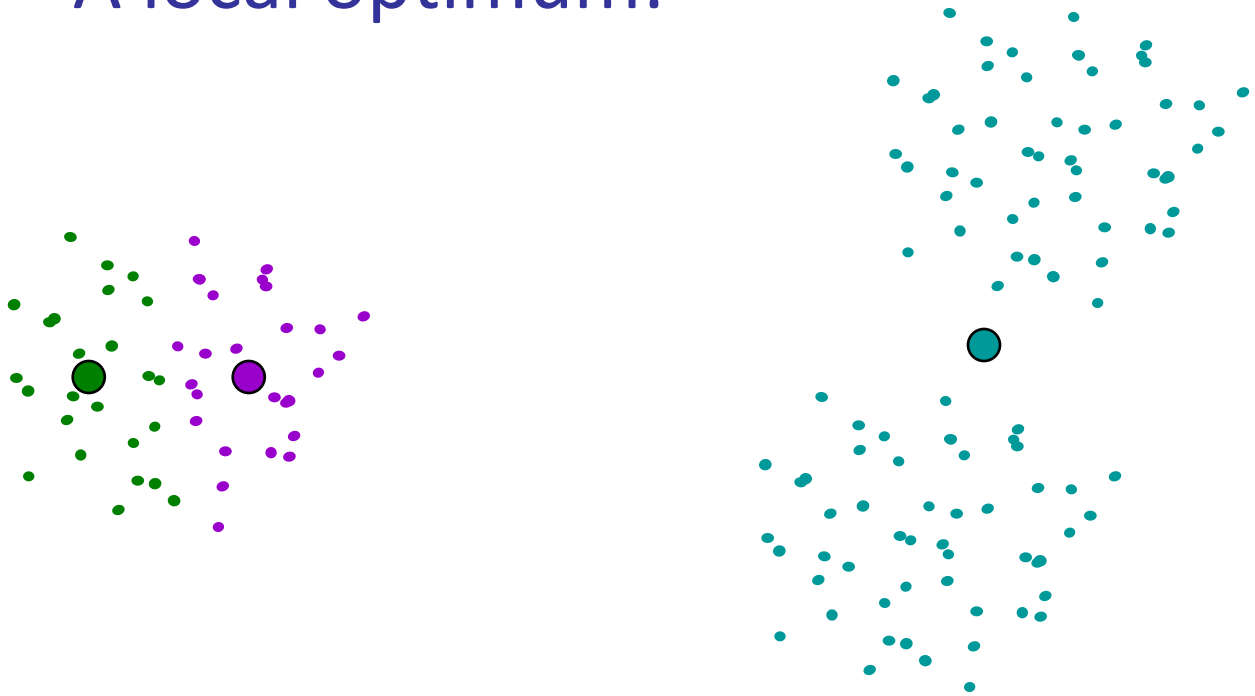
- K-means is non-deterministic
  - Requires initial means
  - It does matter what you pick!
  - What can go wrong?

- Various schemes for preventing this kind of thing: variance-based split / merge, initialization heuristics

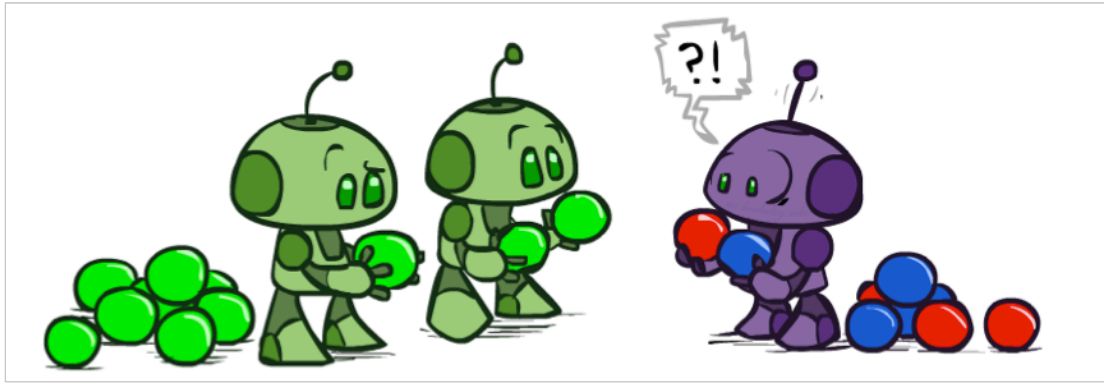


K-Means Getting Stuck

- A local optimum:

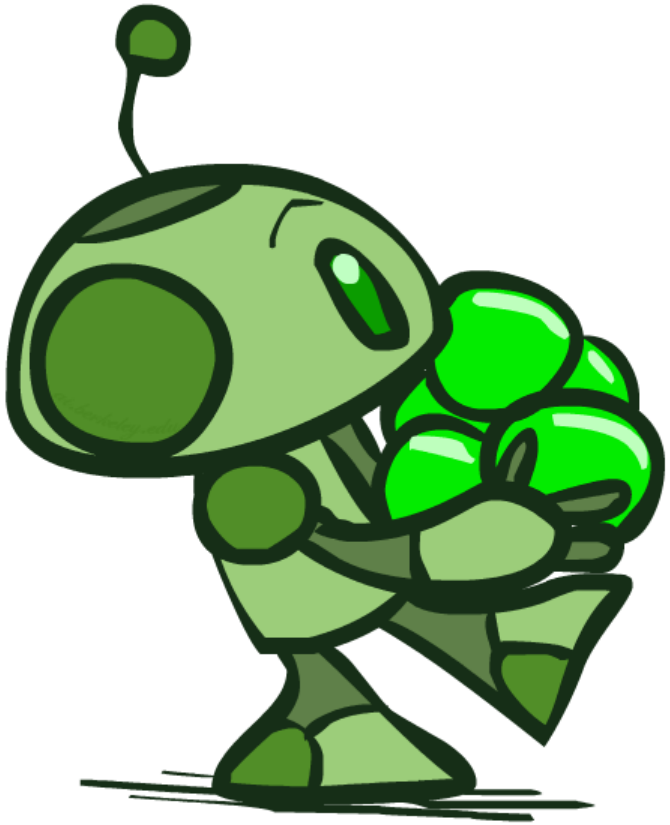


Why doesn't this work out like the earlier example, with the purple taking over half the blue?

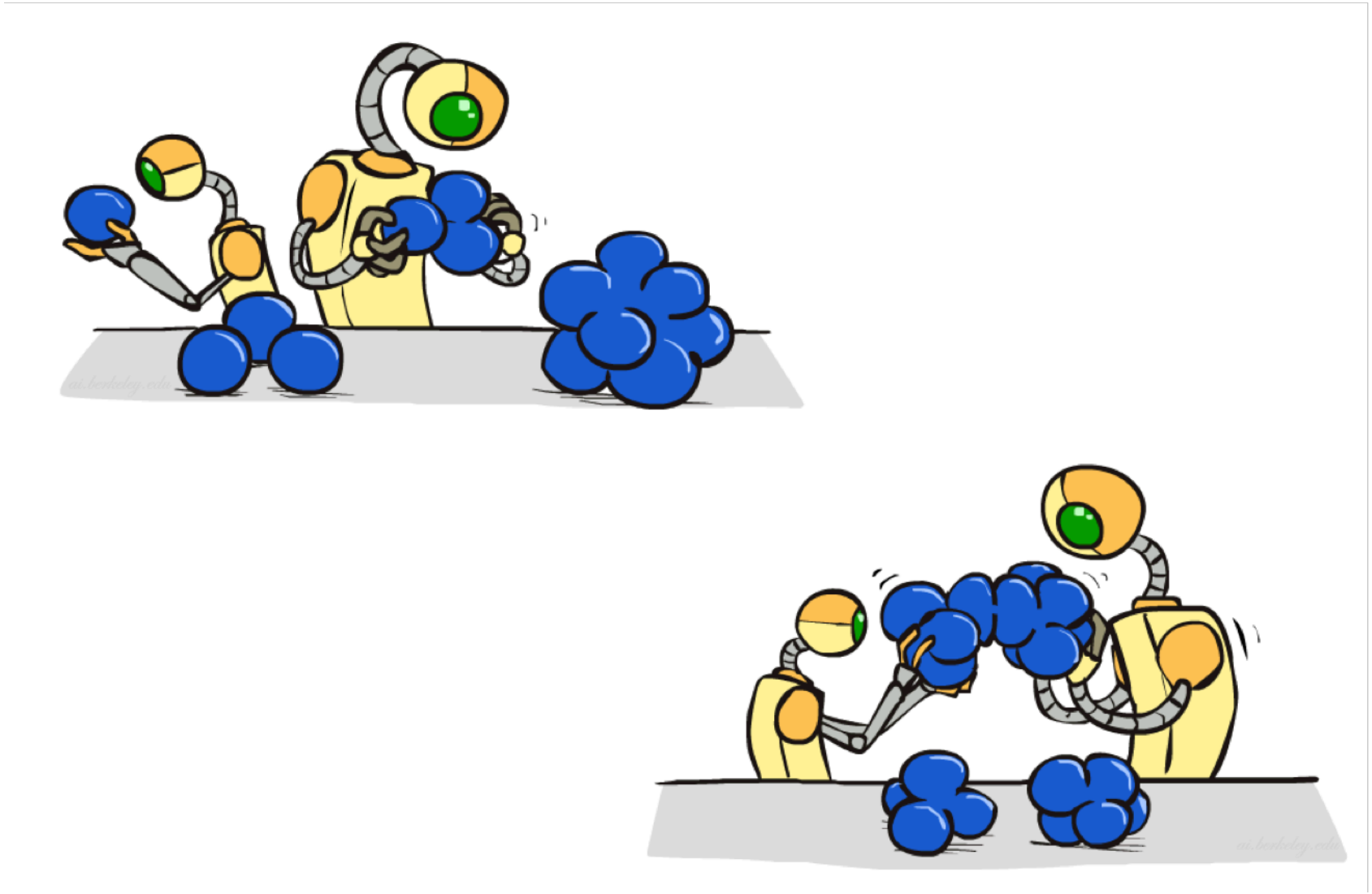


K-Means Questions

- Will K-means converge?
  - To a global optimum?
- Will it always find the true patterns in the data?
  - If the patterns are very very clear?
- Will it find something interesting?
- Do people ever use it?
- How many clusters to pick?

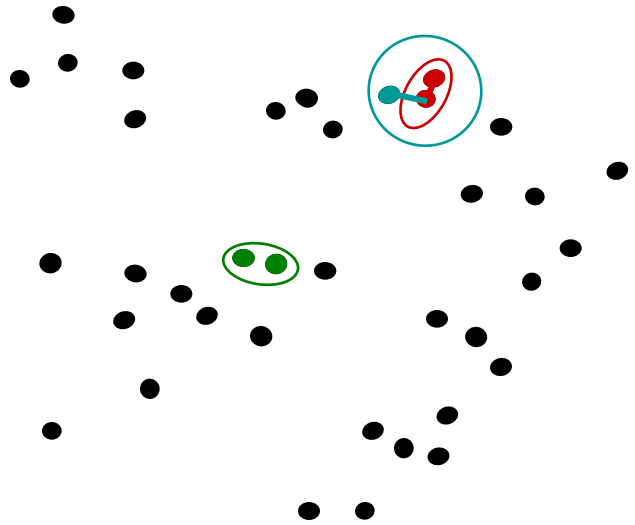


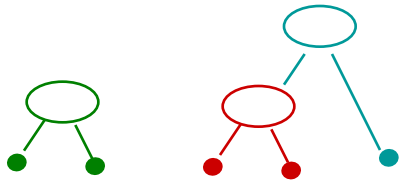
Agglomerative Clustering



Agglomerative Clustering

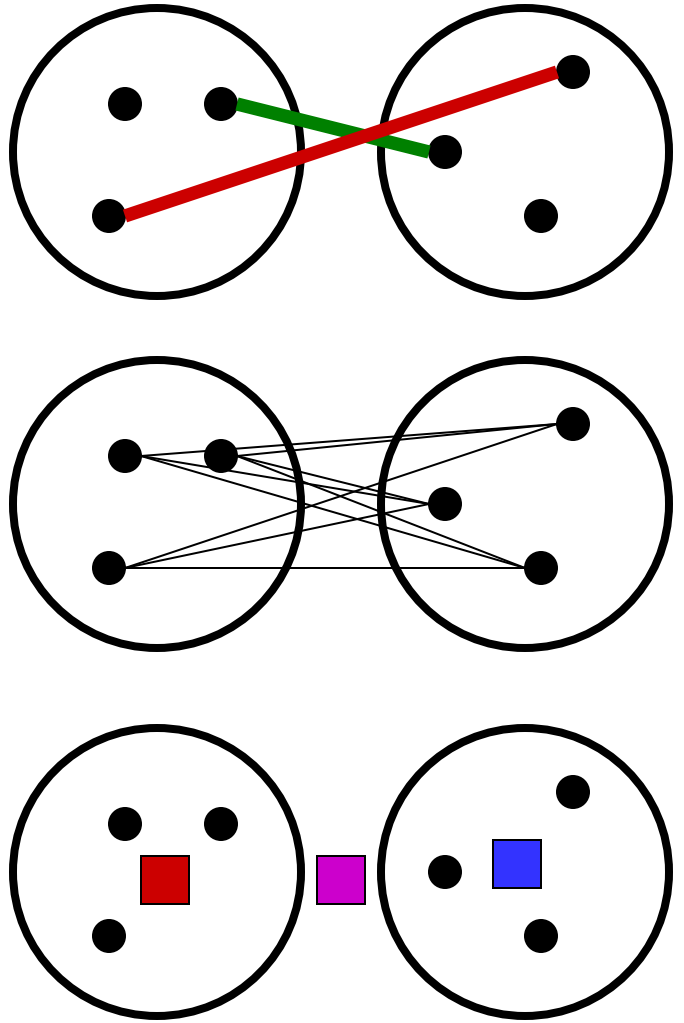
- Agglomerative clustering:
  - First merge very similar instances
  - Incrementally build larger clusters out of smaller clusters
- Algorithm:
  - Maintain a set of clusters
  - Initially, each instance in its own cluster
  - Repeat:
    - Pick the two closest clusters
    - Merge them into a new cluster
    - Stop when there's only one cluster left
- Produces not one clustering, but a family of clusterings represented by a dendrogram



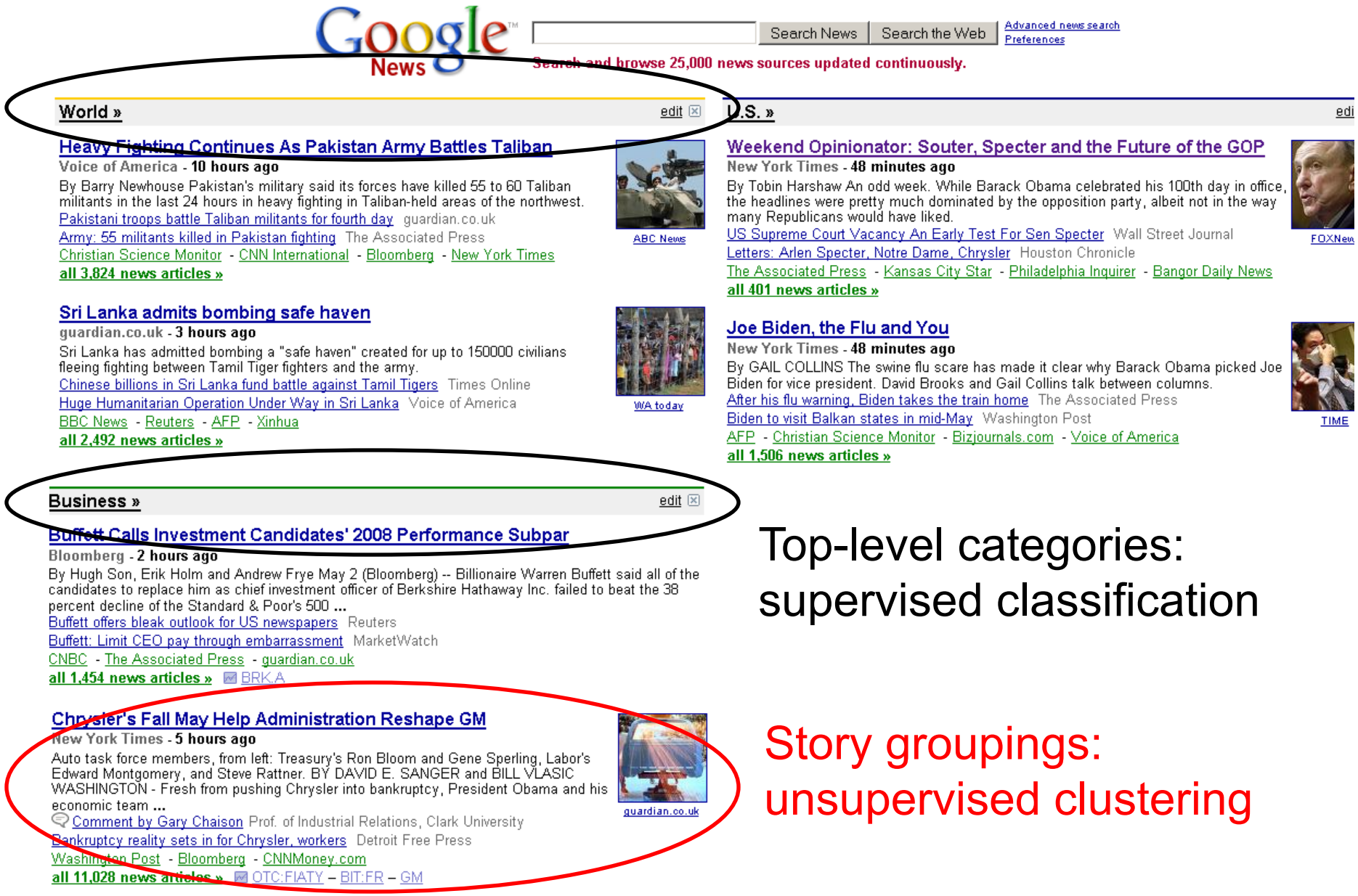


Agglomerative Clustering

- How should we define “closest” for clusters with multiple elements?
- Many options
  - Closest pair (single-link clustering)
  - Farthest pair (complete-link clustering)
  - Average of all pairs
  - Ward’s method (min variance, like k-means)
- Different choices create different clustering behaviors



Example: Document clustering



Top-level categories:
supervised classification

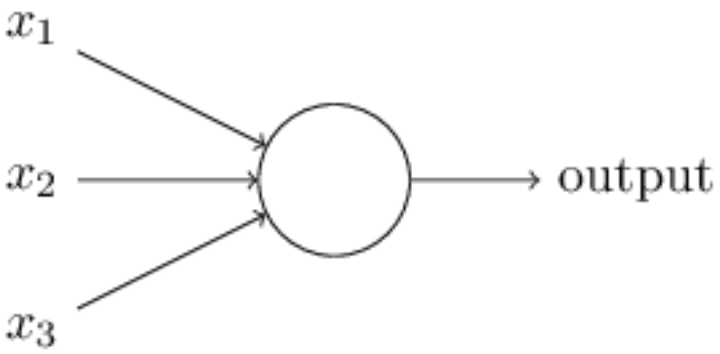
Story groupings:
unsupervised clustering

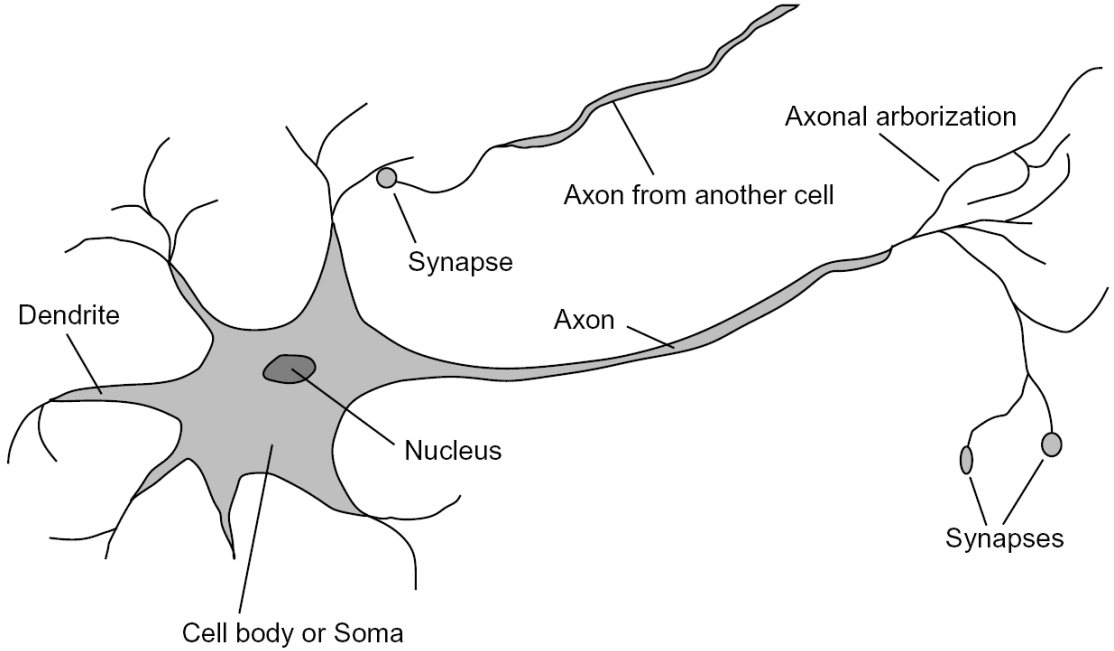
Neural Networks

This lecture is based on Michael Nielsen's *Neural Networks and Deep Learning,* a free online book (with code!).
Please read chapters 1-3.

Review: Perceptron

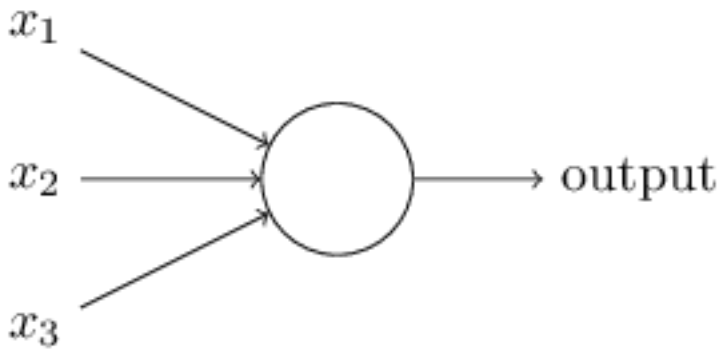
- Perceptrons were developed in the 1950s and 1960s loosely inspired by the neuron.





Review: Perceptron

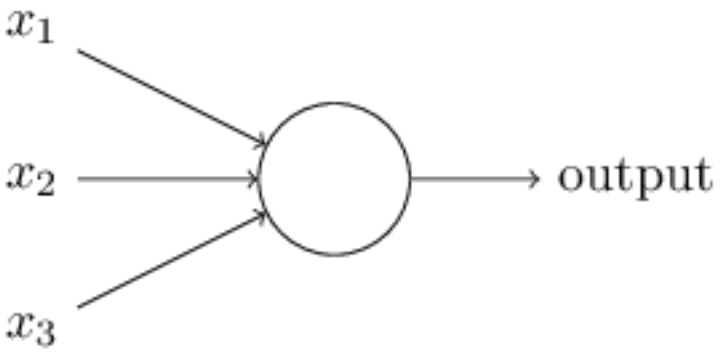
- Perceptron has inputs, x_1,x_2, .. x_N, and weights w_1,w_2, .. w_N
- The perceptron outputs 0 or 1, based on the weighted sum is less than or greater than a threshold value

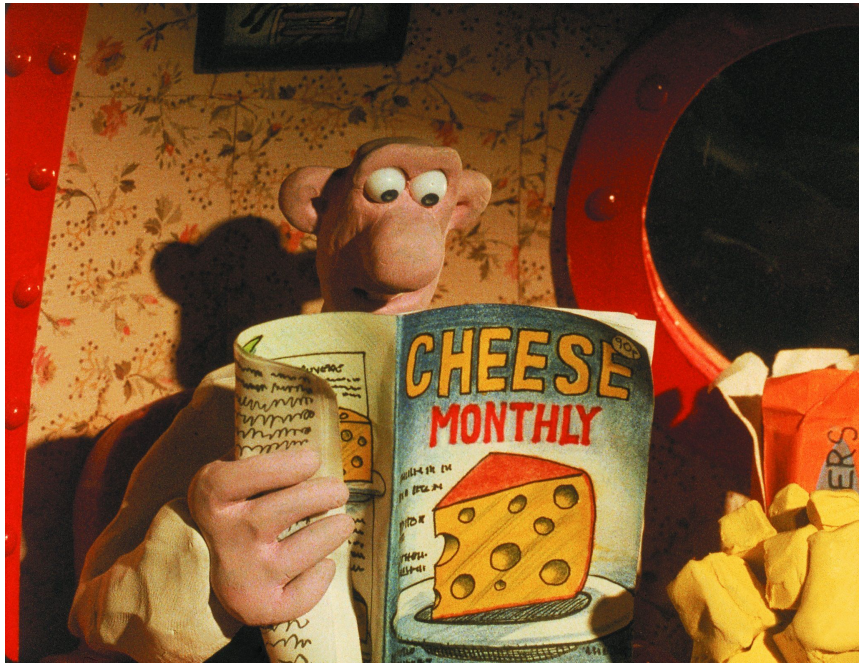


$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Perceptrons for decision making

- We can think about the perceptron as a device that makes decisions by weighing up evidence.
- Example: Suppose there's a cheese festival in your town. You like cheese.



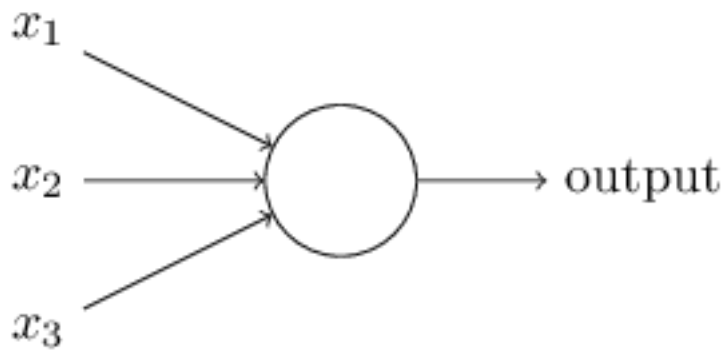


Perceptrons for decision making

- You might use 3 factors to decide whether to go.

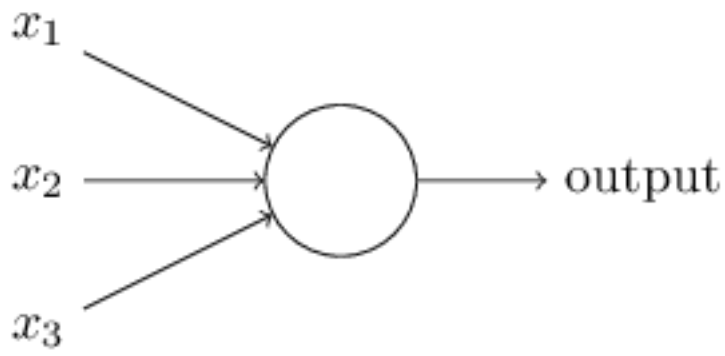
1. Is the weather good?
2. Can your loyal companion come with you?
3. Is the festival near public transit?

- These can be the binary input values to a perceptron



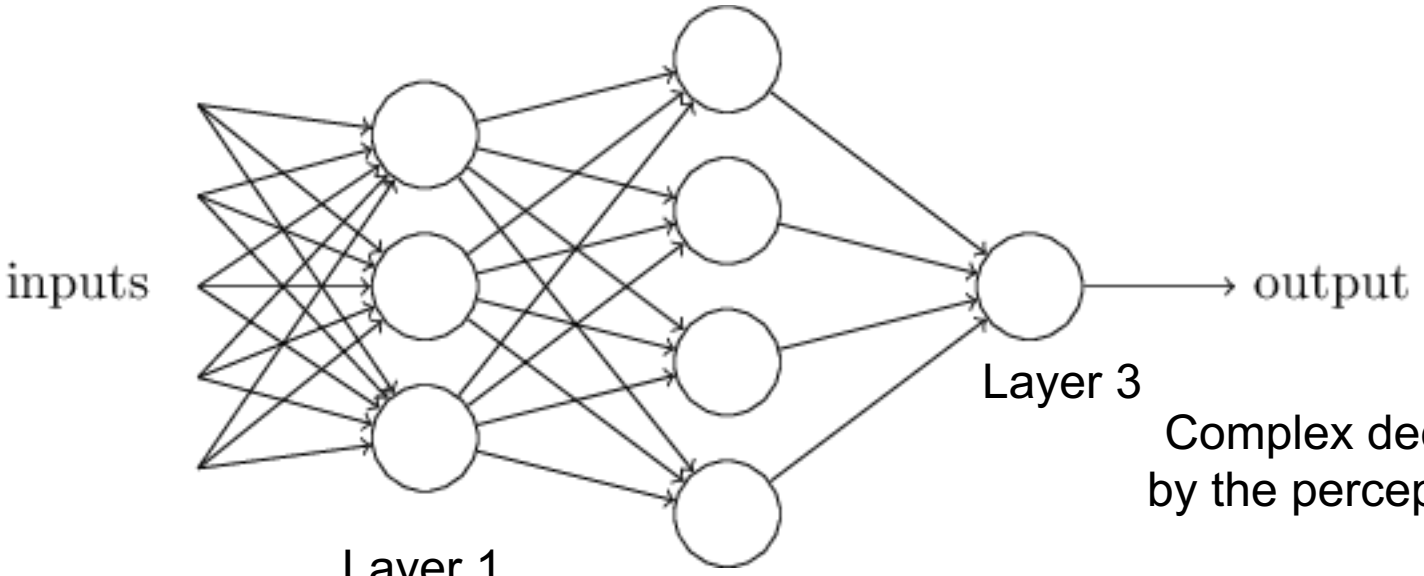
Perceptrons for decision making

- By varying weights and the threshold we get different models of decision making
- Example 1: $w_1 = 6$ $w_2 = 2$ $w_3 = 2$, threshold = 5
- Example 2: $w_1 = 6$ $w_2 = 2$ $w_3 = 2$, threshold = 3



Perceptrons for decision making

- A complex network of perceptrons could make quite subtle decisions:



1st layer makes 3 simple decisions by weighing the input evidence

2nd layer makes a decision by weighing up the results from the 1st layer of decision-making

Complex decisions can be made by the perceptron in the third layer

Weights, bias and dot products

- Two notational changes simplify the way that perceptrons are described.
- The first change is to replace the weighted sum as a dot product

$$w \cdot x \equiv \sum_j w_j x_j$$

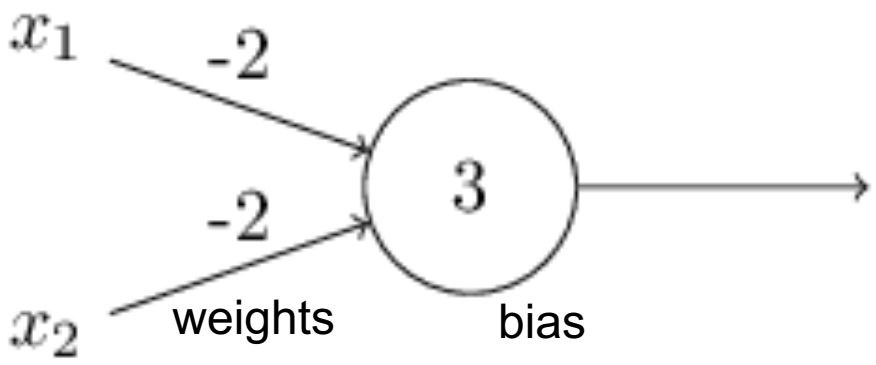
- The second change is to move the threshold to the other side of the inequality, and to replace it by a *bias*, b≡−threshold

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Decision making OR logical functions

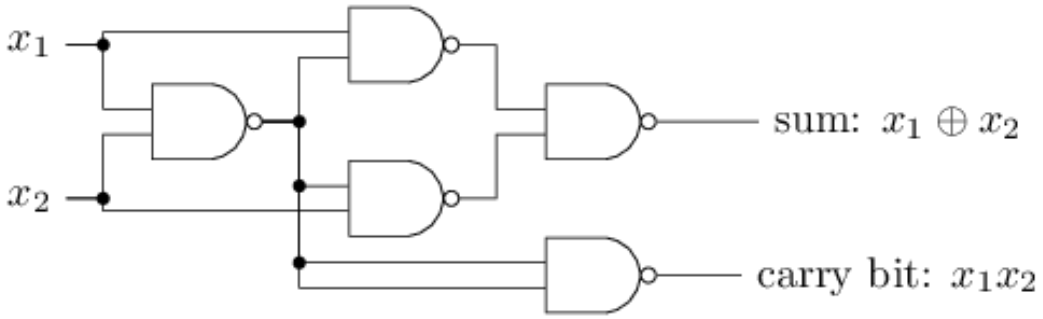
- Perceptrons can be used is to compute logical functions like AND, OR and NAND
- Example:

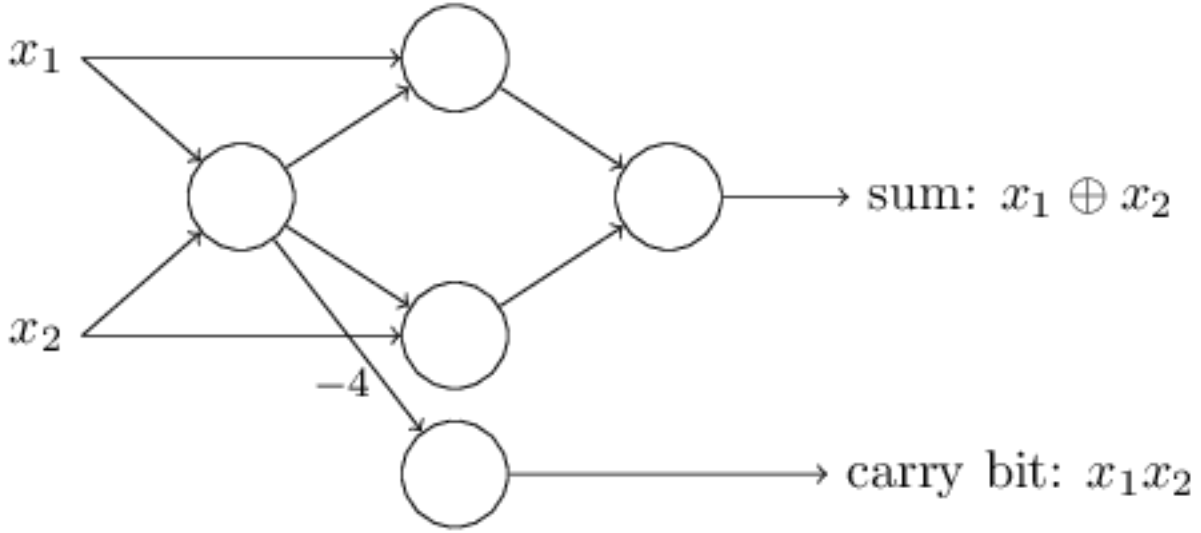


Input	Weighted sum	Output
00	-2*0 + -2*0 +3 = 3	1
10 or 01	-2*1 + -2*0 +3 = 1	1
11	-2*1 + -2*1 +3 = -1	0

Logical functions

- Networks of perceptrons to compute *any* logical function
- We can build any computation up out of NAND gates.
- For example, a circuit which adds two bits x_1 and x_2





All unlabeled weights are -2, all biases =3.

Power of Perceptrons

- Networks of Perceptrons are universal for computation, like NAND gates
- Perceptrons can be as powerful as any other computing device!

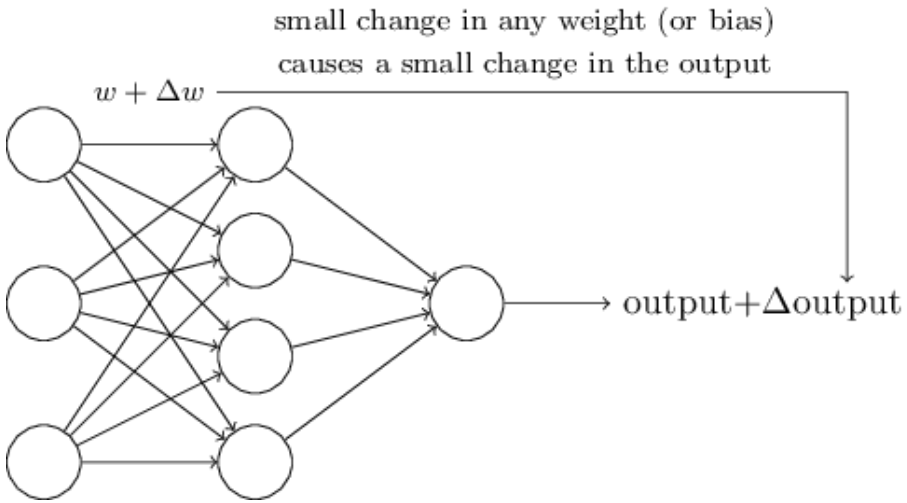
- We can devise *learning algorithms* to automatically tune the weights and biases of a network of artificial neurons
- Instead of laying out a circuit of NAND and other gates, neural networks can simply learn to solve problems

Learning to classify digits

- Input: image encoded input as a vector of intensities:

 $= \langle 0.0 \;\; 0.0 \;\; 0.3 \;\; 0.8 \;\; 0.7 \;\; 0.1 \ldots 0.0 \rangle$

- Output: is this a one or not?
- Goal: set the parameters of the network to correctly classify the digits
- Learning = changing the weights an biases in the network.



 0

 1

 0

 0

 1

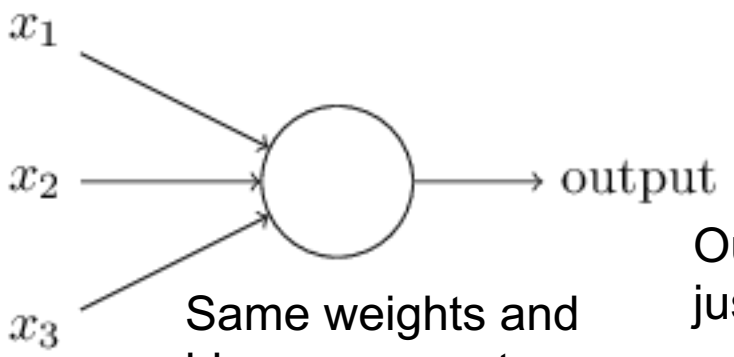
 0

Sigmoid neurons

- Problem: a small change in the weights or bias of any single perceptron in the network can causes the output to completely flip from 0 to 1.
- Solution: sigmoid neuron

Perceptron

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



Same weights and bias as perceptron

Output is no longer just 1 or 0.

Sigmoid neuron

output = $\sigma(w \cdot x + b)$

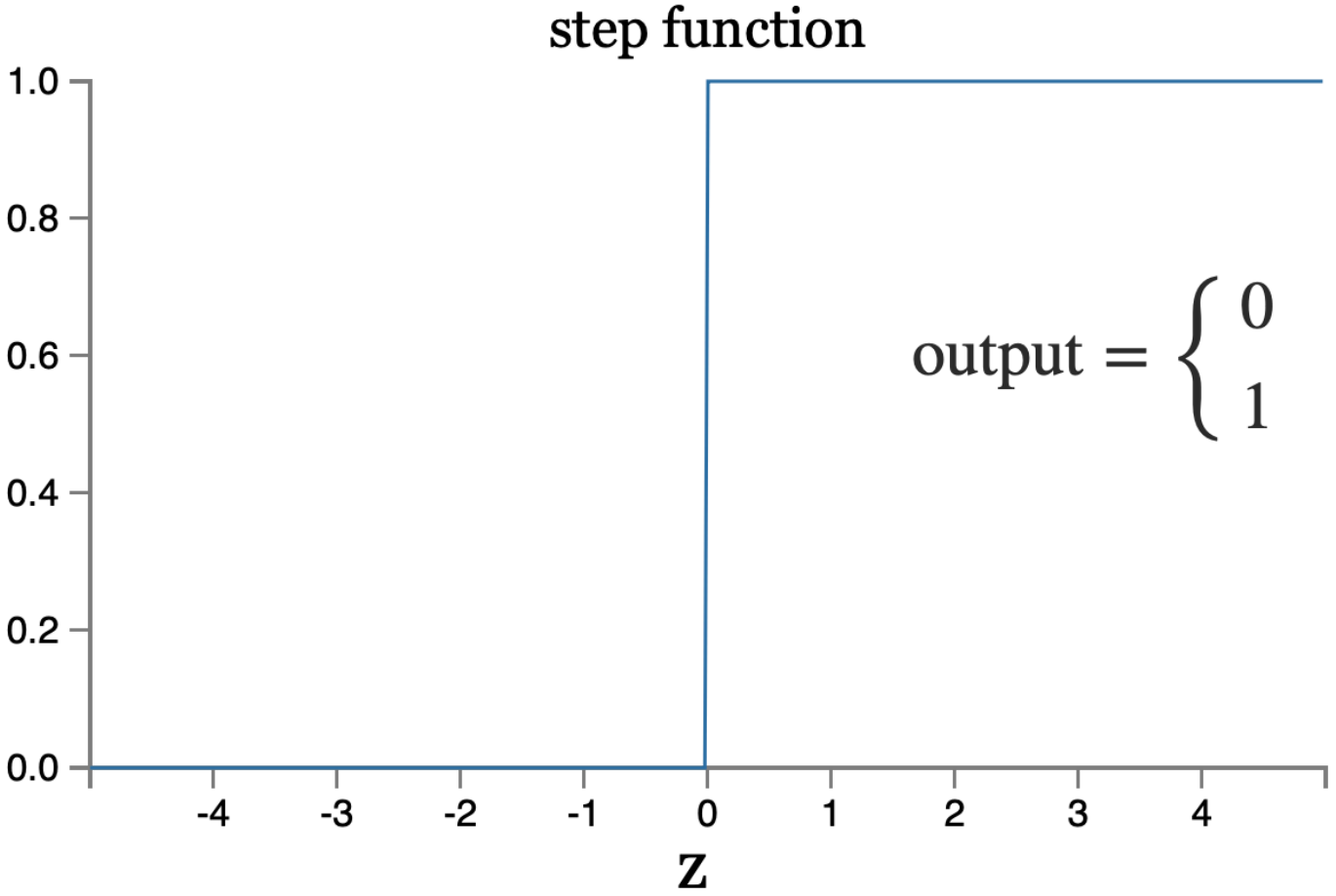
Sigmoid function:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

Inputs: any real-valued number

Perceptron

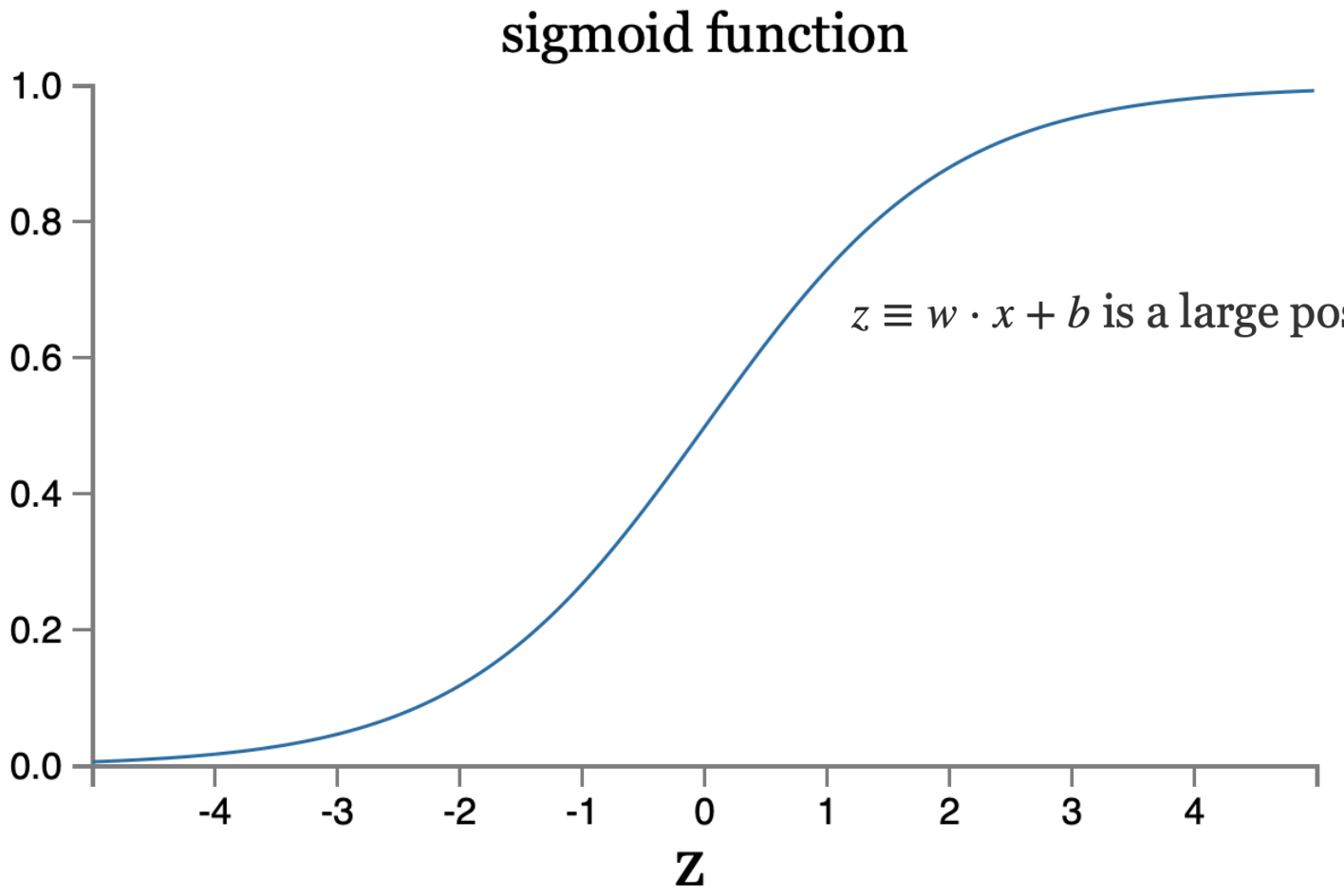
$$z \equiv w \cdot x + b$$



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Sigmoid neuron

$z \equiv w \cdot x + b$



$z \equiv w \cdot x + b$ is a large positive number. Then $e^{-z} \approx 0$

$z = w \cdot x + b$ is very negative. Then $e^{-z} \rightarrow \infty$, and $\sigma(z) \approx 0$

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

Smoothness is crucial

- Smoothness of σ means that small changes in the weights w_j and in the bias b will produce a small change the output from the neuron

$$\Delta \text{output} \approx \sum_j \frac{\partial \, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \, \text{output}}{\partial b} \Delta b$$

- Δoutput is a *linear function* of the changes Δwj and Δb
- This makes it easy to choose small changes in the weights and biases to achieve any desired small change in the output

Next time: Neural Nets and Gradient Descent