Announcement

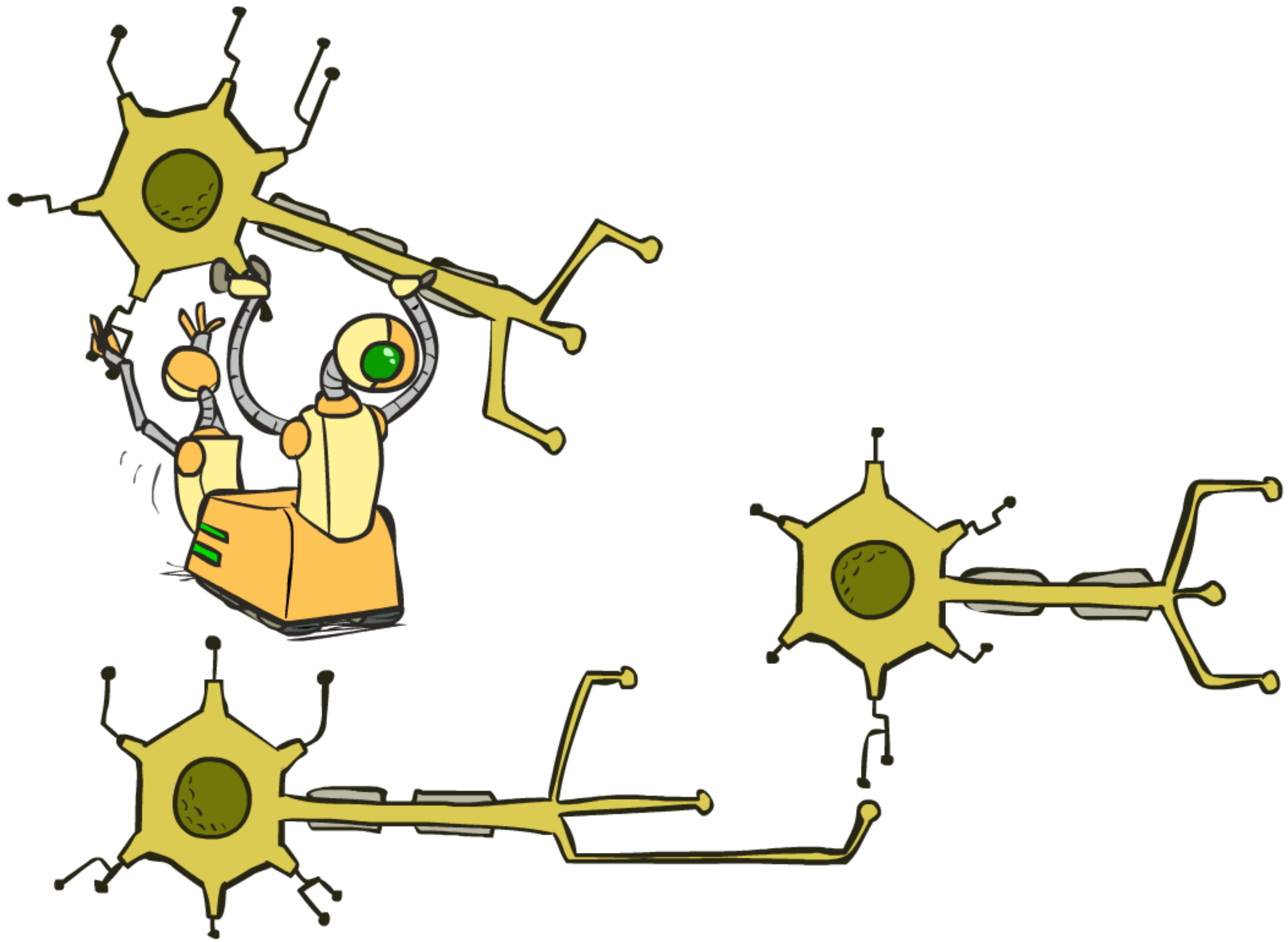
- Proposal presentation on Dec. 14 & 16
- Problem set 5 is out, due Dec 9
- Homework 5 is out

Proposal Presentation

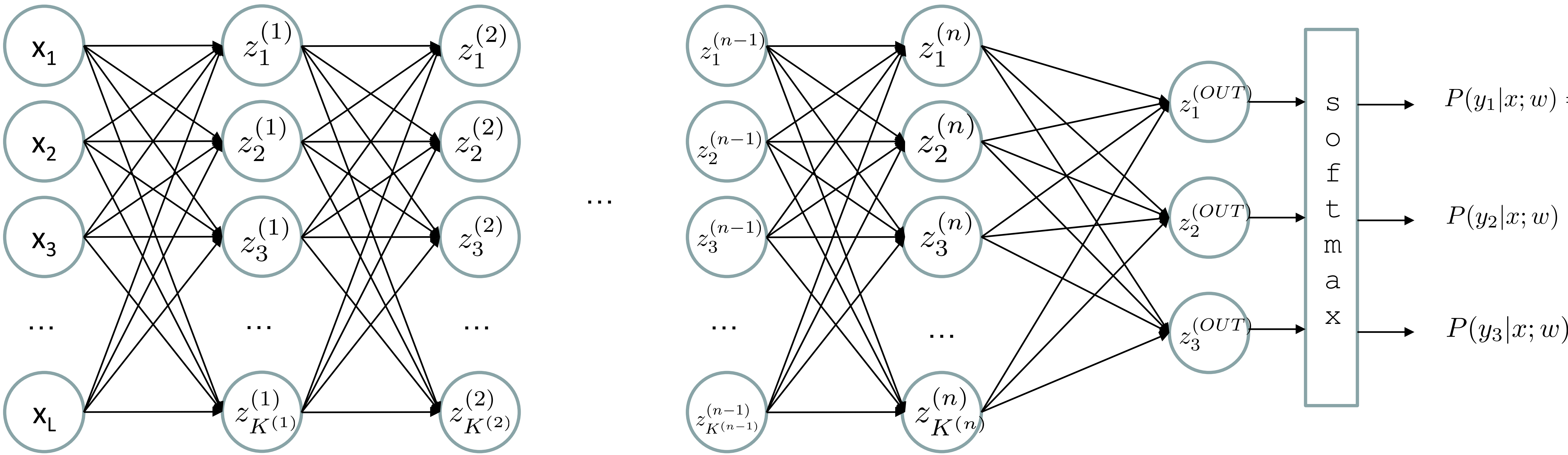
- Proposal presentation
  - 4-5 min presentation: topic, motivation, possible methods
  - Dec. 14, 16, in class
  - Presentation schedule will be sent out later

- If you have not formed/joined a group, please do so ASAP
  - Group registration: https://wj.qq.com/s2/7551413/2fd0/

Neural Networks



Deep Neural Network = Also learn the features!

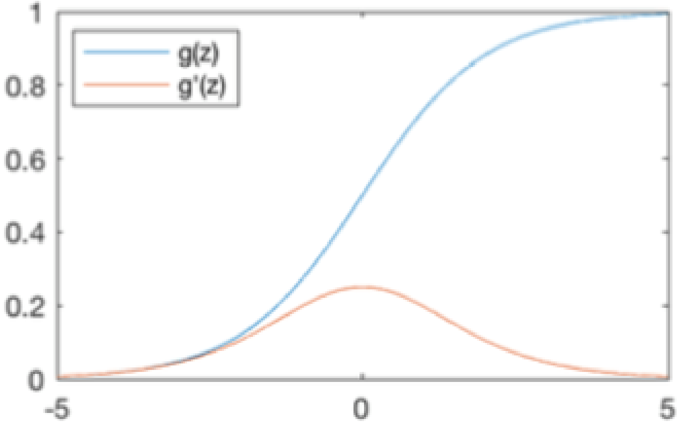


$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Common Activation Functions

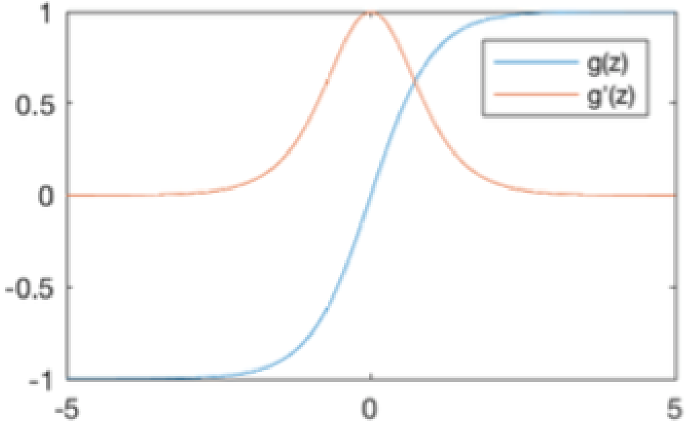
Sigmoid Function



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

$$g'(z) = g(z)(1 - g(z))$$

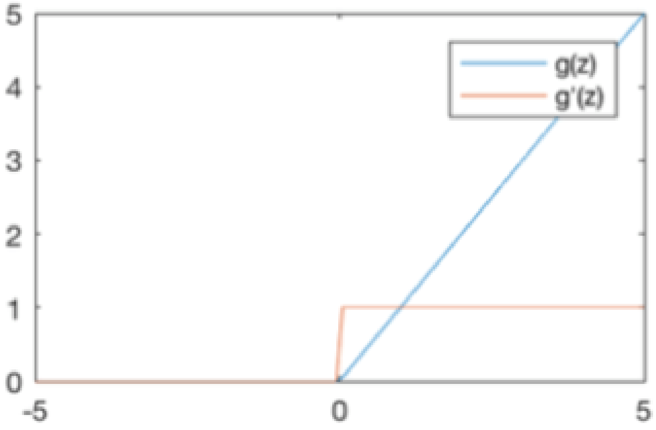
Hyperbolic Tangent



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

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[source: MIT 6.S191 introtodeeplearning.com]

Deep Neural Network: Also Learn the Features!

- Training the deep neural network is just like logistic regression:

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

just w tends to be a much, much larger vector ☺

→just run gradient ascent

+ stop when log likelihood of hold-out data starts to decrease

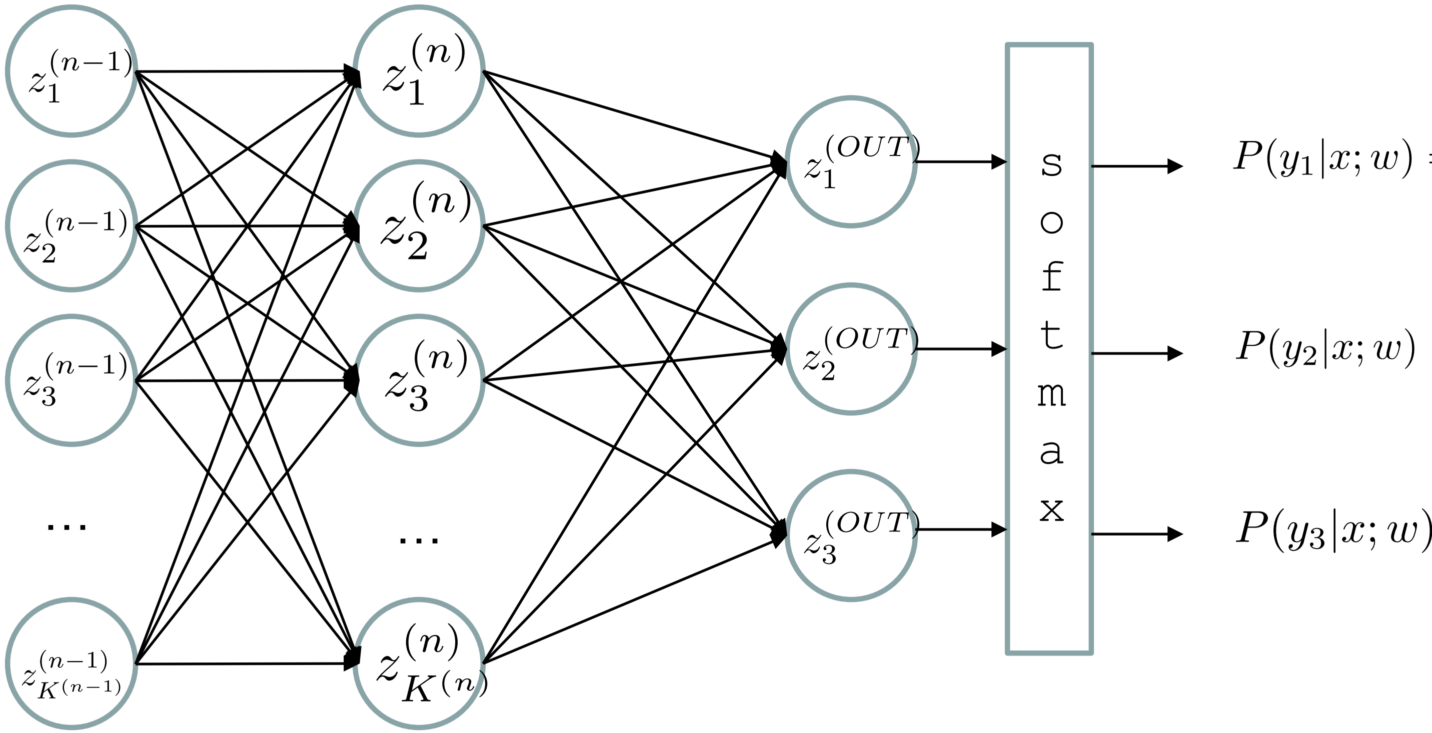
Neural Networks Properties

- Theorem (Universal Function Approximators). A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.
- Practical considerations
  - Can be seen as learning the features
  - Large number of neurons
    - Danger for overfitting
    - (hence early stopping!)

Training a Network

Key words:

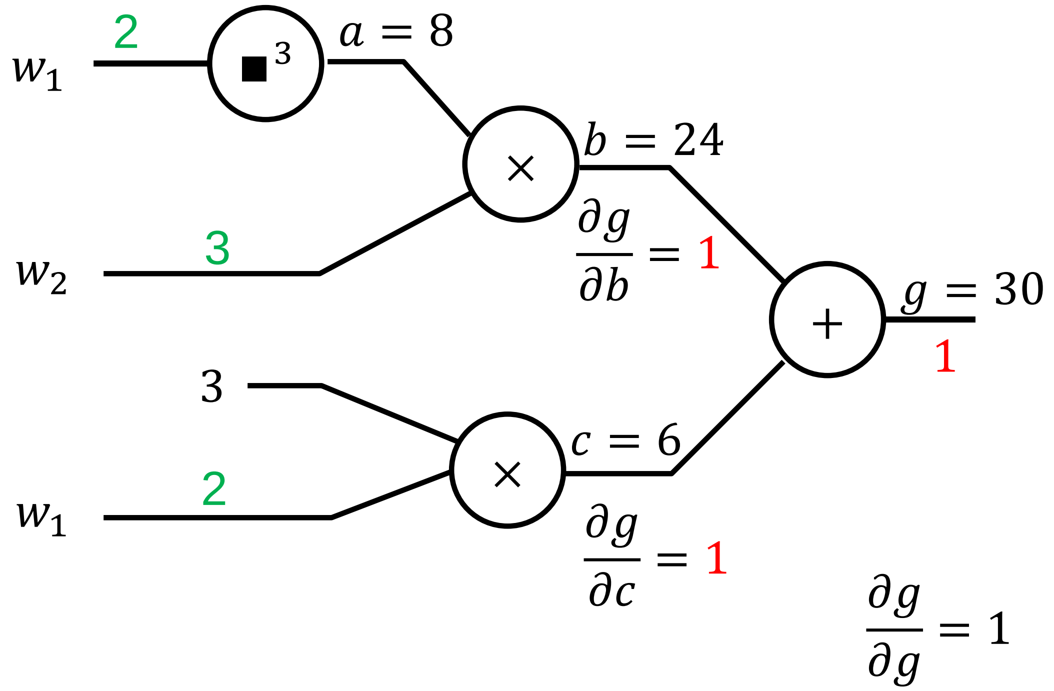
- Forward
- Backwards
- Gradient
- Backprop



g = nonlinear activation function

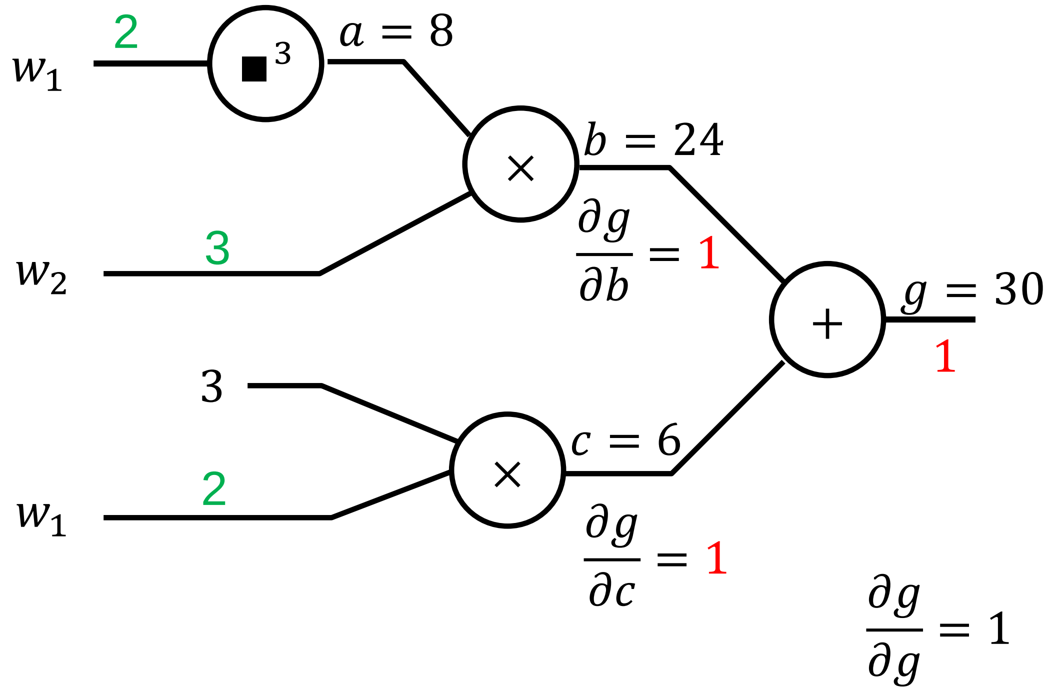
Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

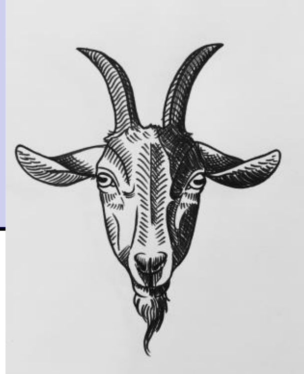


Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

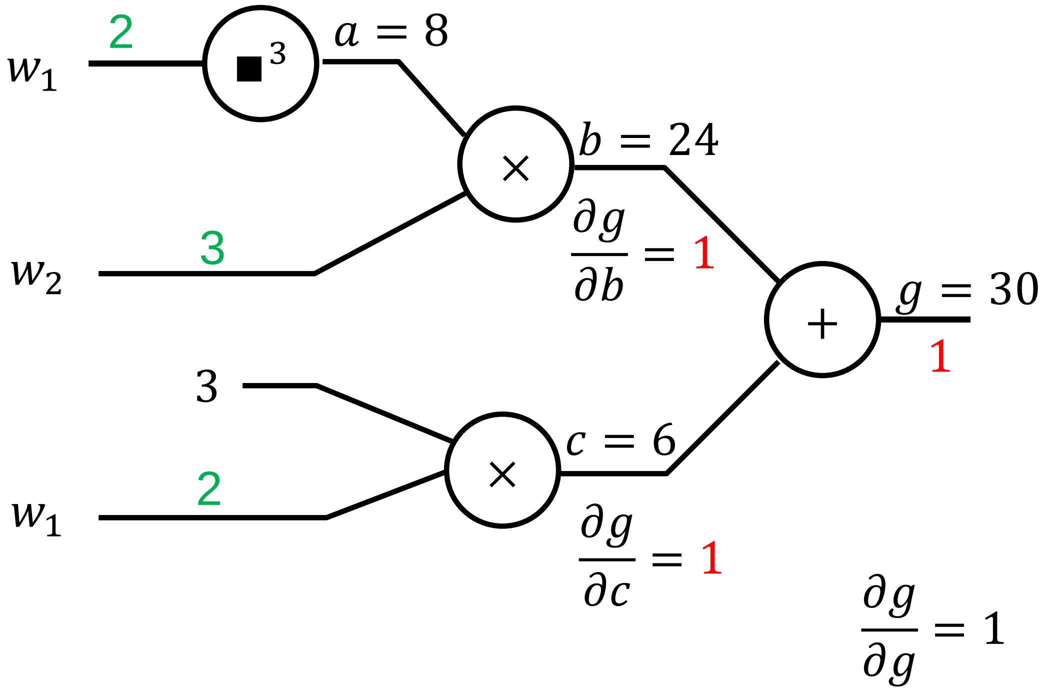
- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a}$



Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

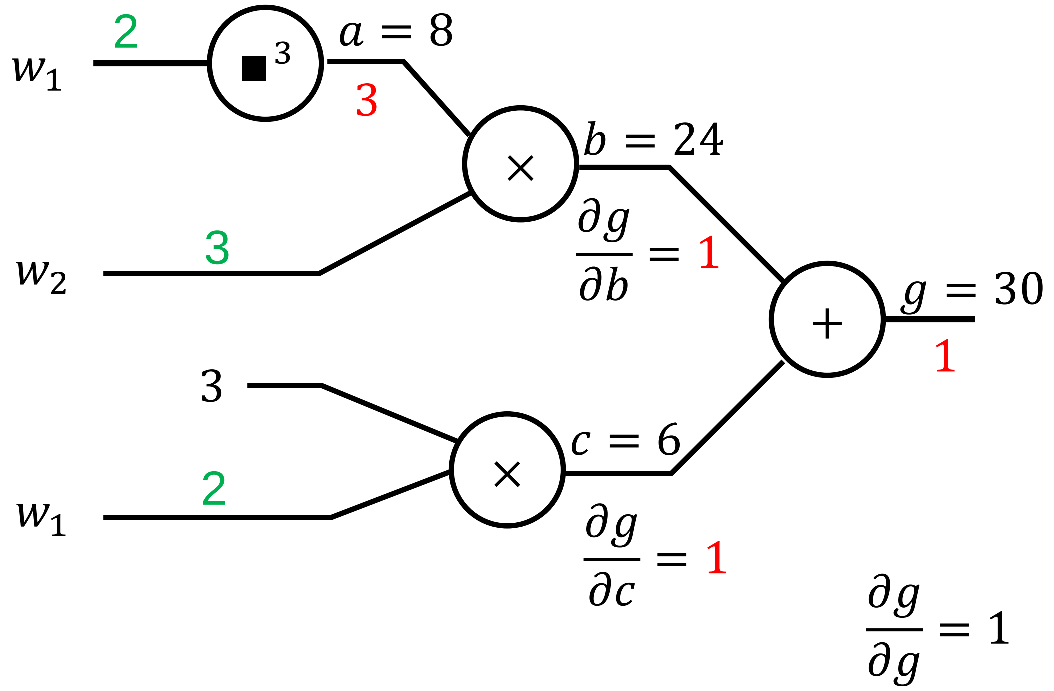


- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} =?????$

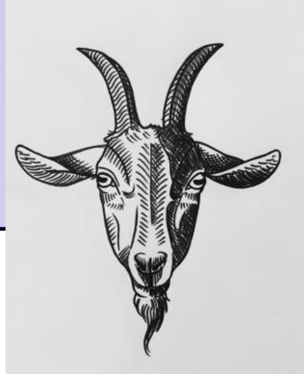


Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

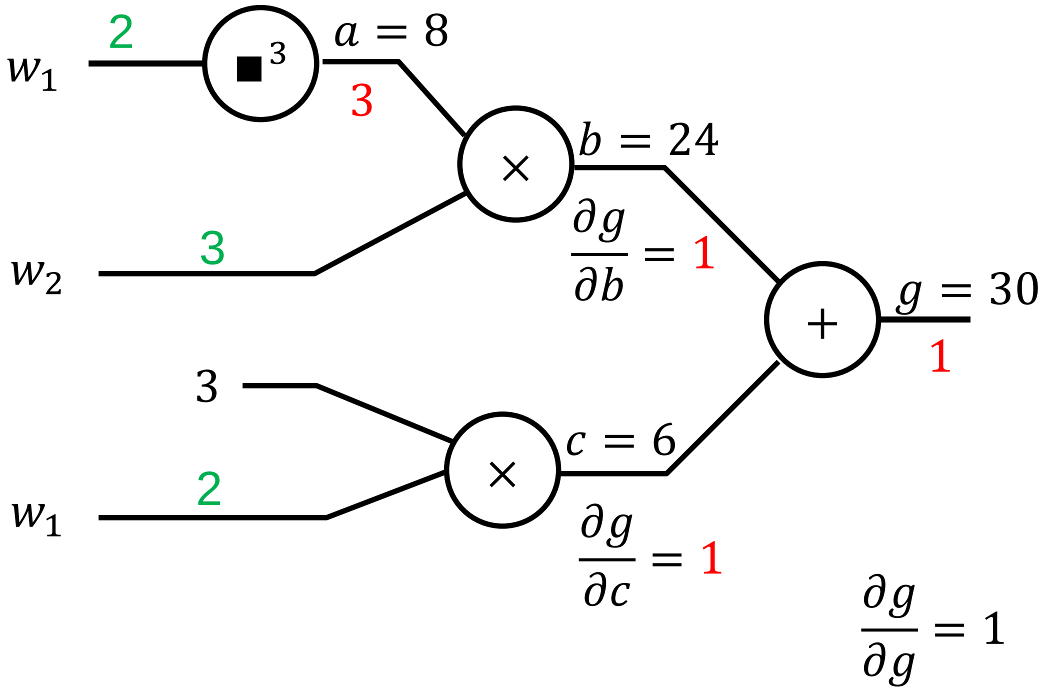
- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$



Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

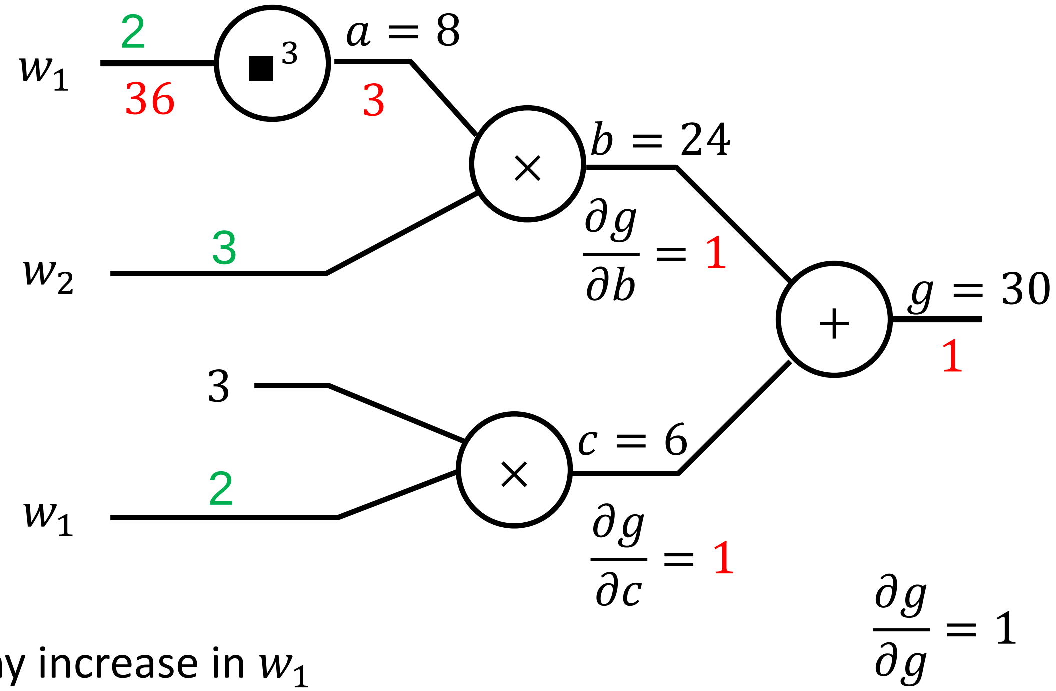


- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2,3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = ?????$



Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g/\partial w_1$ and $\partial g/\partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$

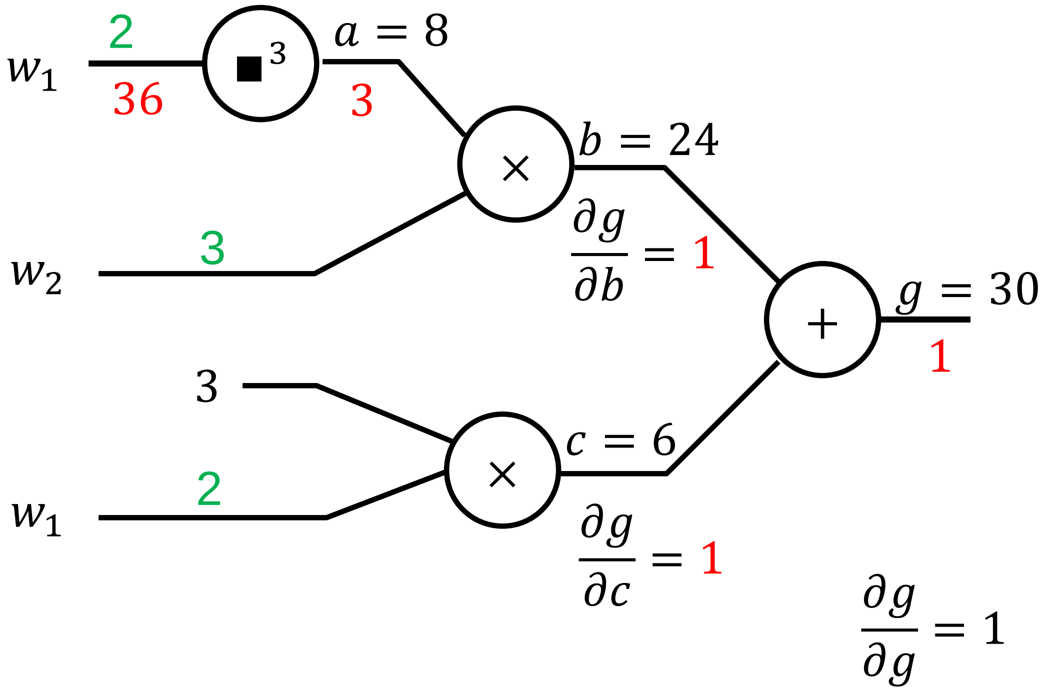


Interpretation: A tiny increase in w_1 will result in an approximately $36w_1$ increase in g due to this cube function.

Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

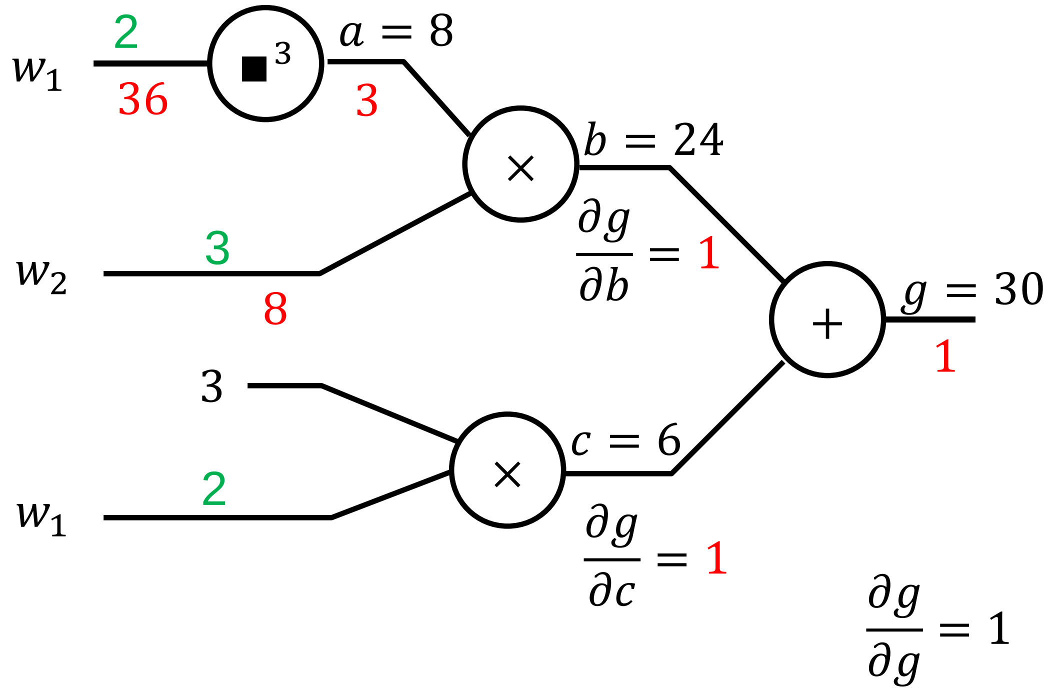
- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$
- $\frac{\partial g}{\partial w_2} =???$

Hint: $b = a \times 3$ may be useful.



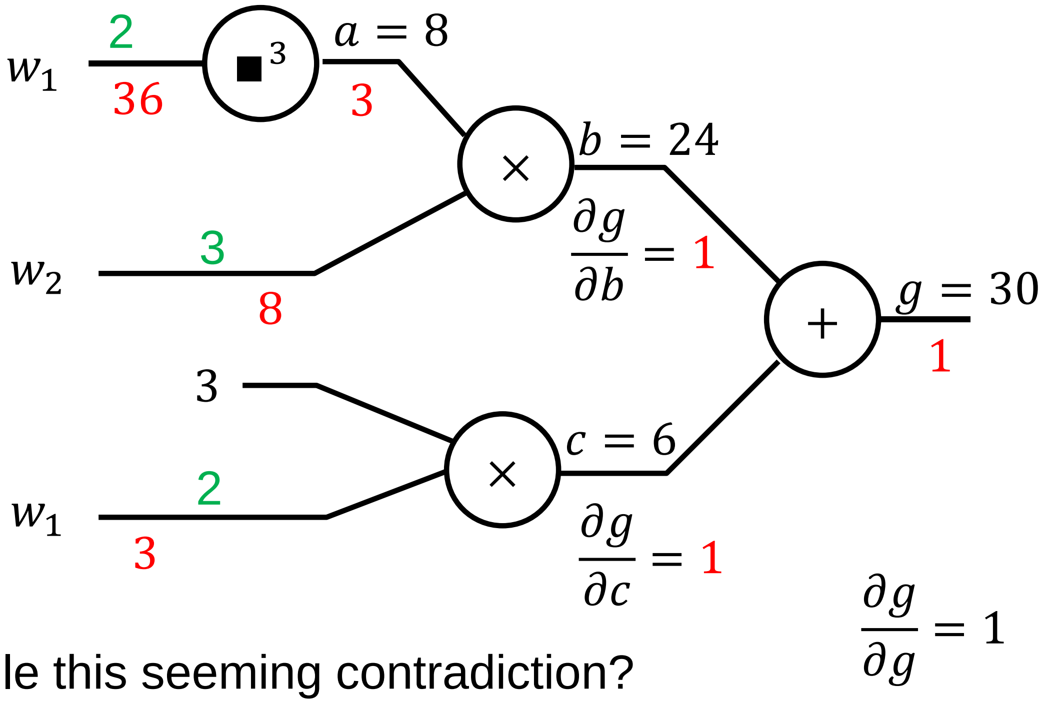
Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1\frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
  - $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial w_2} = 1\frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$



Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

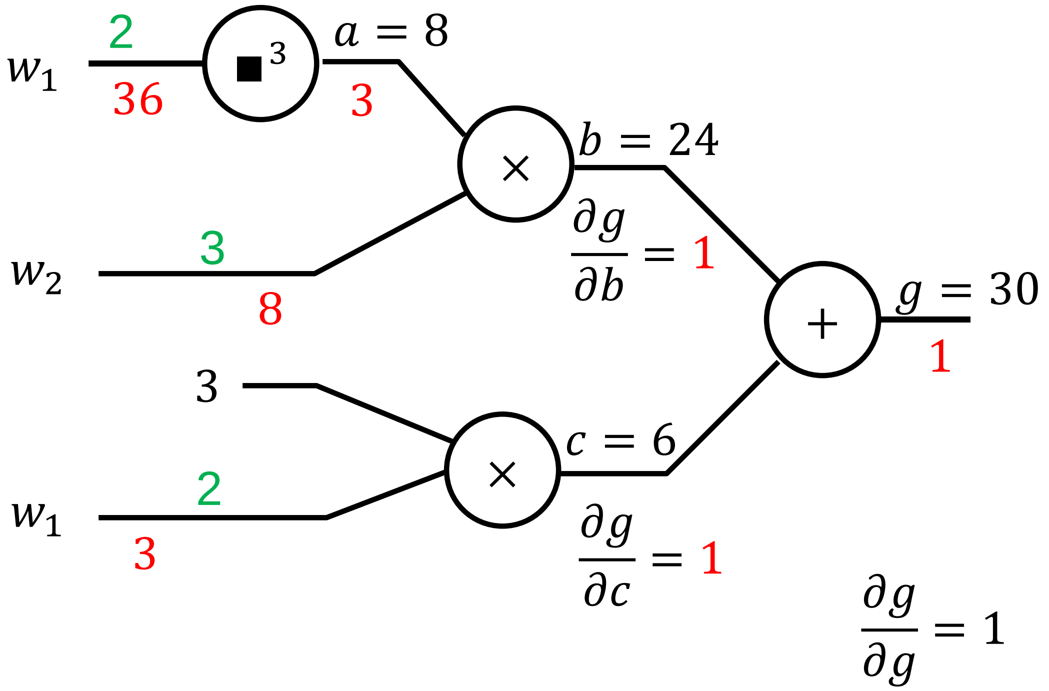
- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1\frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
  - $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial w_2} = 1\frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$
- $c = 3w_1$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c}\frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$



How do we reconcile this seeming contradiction? Top partial derivative means cube function contributes $36w_1$ and bottom p.d. means product contributes $3w_1$ so add them.

Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1\frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
  - $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial w_2} = 1\frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$
- $c = 3w_1$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c}\frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$



$\nabla g = \left[\frac{\partial g}{\partial w_1}, \frac{\partial g}{\partial w_2}\right] = [39, 8]$

Gradient Descent

- Punchline: If we can somehow compute our gradient, we can use gradient descent.
- How do we compute the gradient?
  - Purely analytically.
    - Gives exact symbolic answer. Infeasible for functions of lots of parameters or input values.
  - Finite difference approximation.
    - Gives approximation, very easy to implement.
    - Runtime for ll: $O(NM)$, where N is the number of parameters, and M is number of data points.
  - Back propagation.
    - Gives exact answer, difficult to implement.
    - Runtime for ll: $O(NM)$

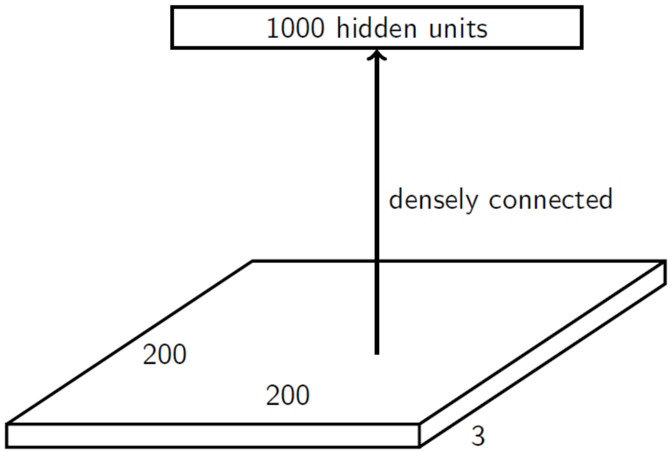
$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}); w)$$

Automatic Differentiation

- Automatic differentiation software
  - e.g. Theano, TensorFlow, PyTorch, Chainer
  - Only need to program the function g(x,y,w)
  - Can automatically compute all derivatives w.r.t. all entries in w
  - This is typically done by caching info during forward computation pass of f, and then doing a backward pass = "backpropagation"
  - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass
- Need to know this exists

Convolutional Neural Networks

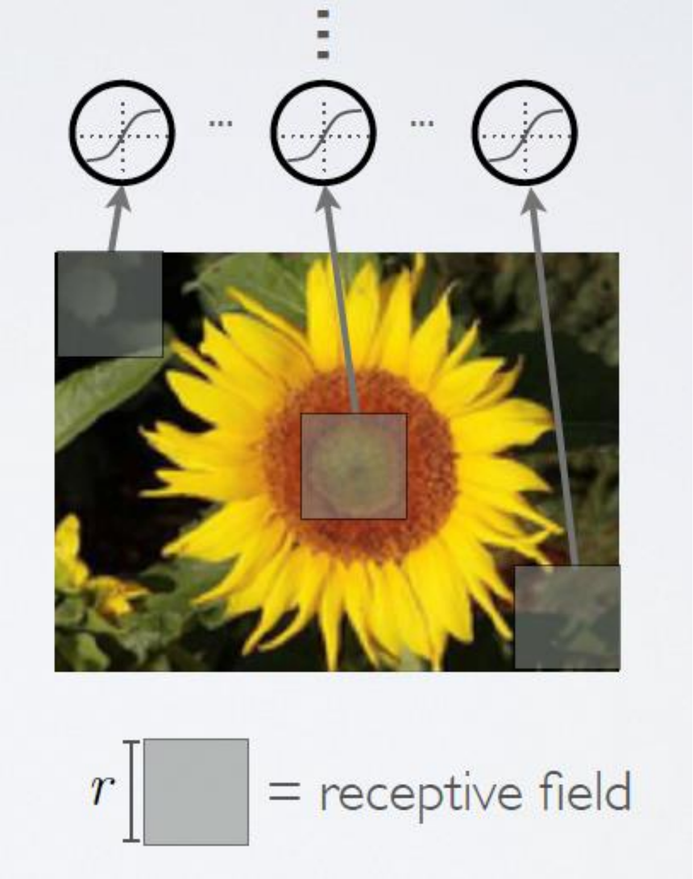
- Visual recognition
  - Suppose we aim to train a network that takes a 200x200 RGB image as input



  - What is the problem with have full connections in the first layer?
    - Too many parameters! 200x200x3x1000 = 120 million
    - What happens if the object in the image shifts a little?

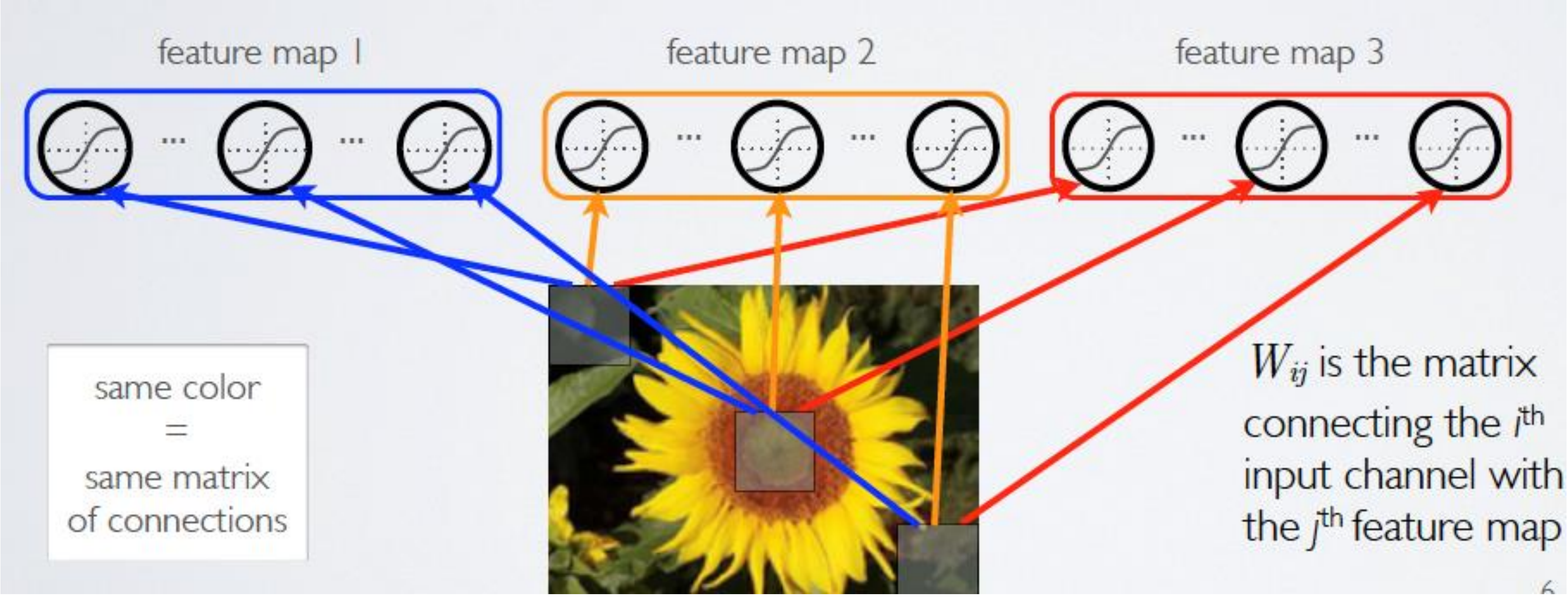
Overview of CNNs

- First idea: Use a local connectivity of hidden units
  - Each hidden unit is connected only to a subregion (patch) of the input image
  - Usually it is connected to all channels
  - Each neuron has a local receptive field



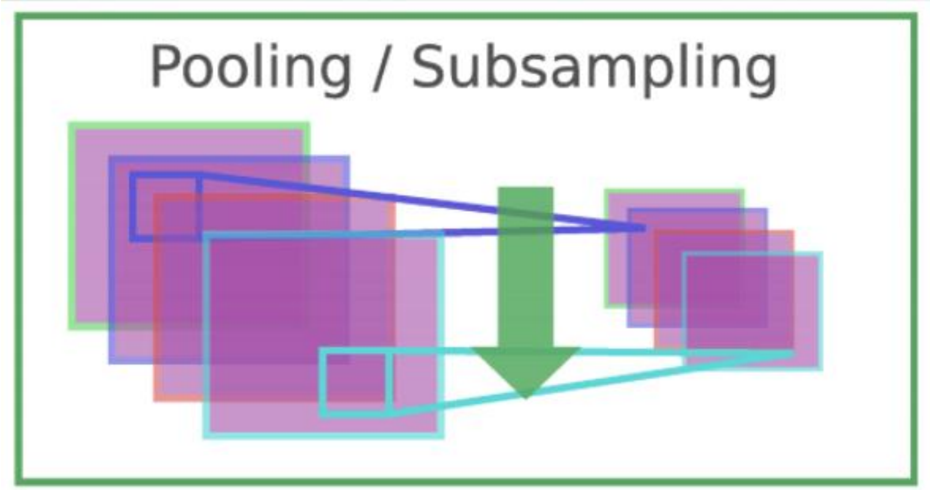
Overview of CNNs

- Second idea: share weights across certain units
  - Units organized into the same “feature map” share weight parameters
  - Hidden units within a feature map cover different positions in the image



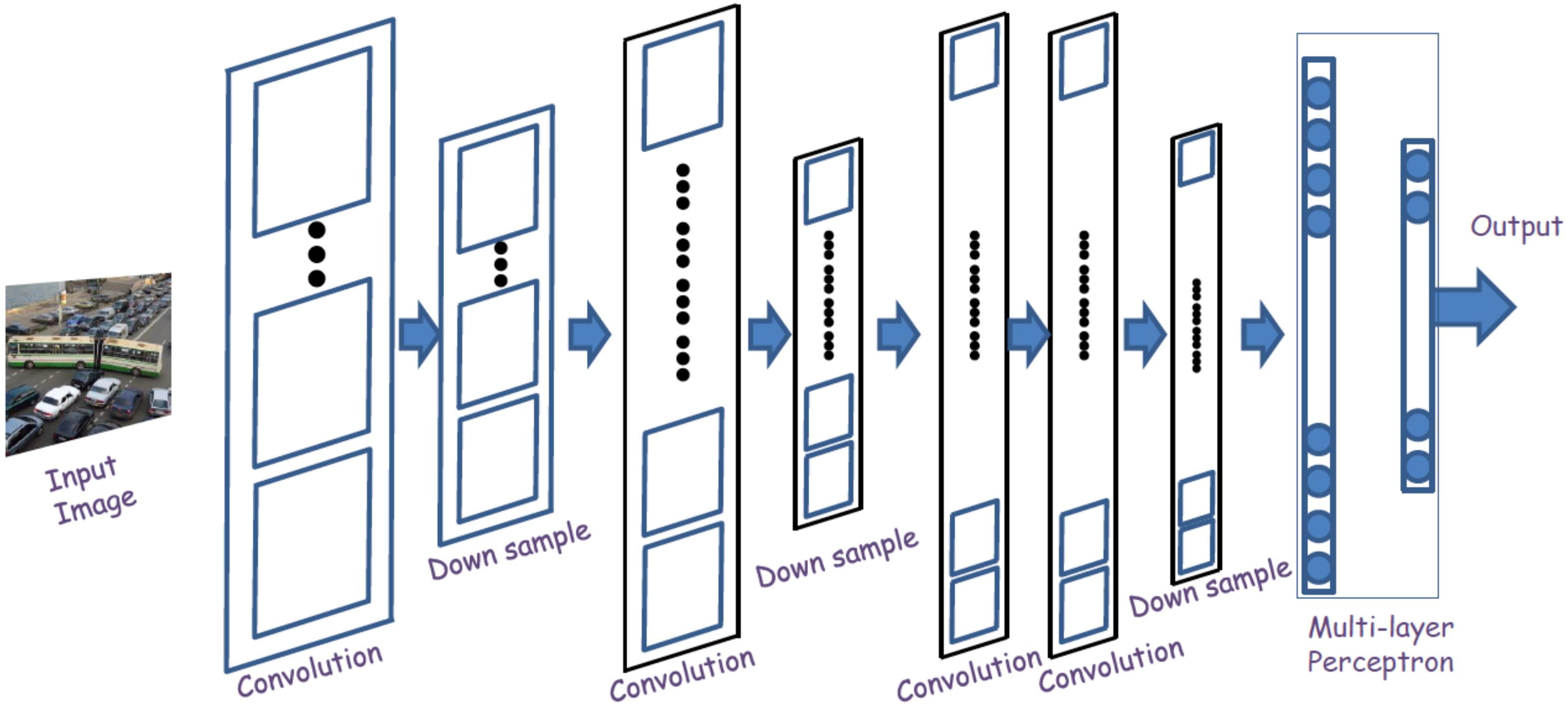
Overview of CNNs

- Third idea: pool hidden units in the same neighborhood
  - Averaging or Discarding location information in a small region
  - Robust toward small deformations in object shapes by ignoring details.



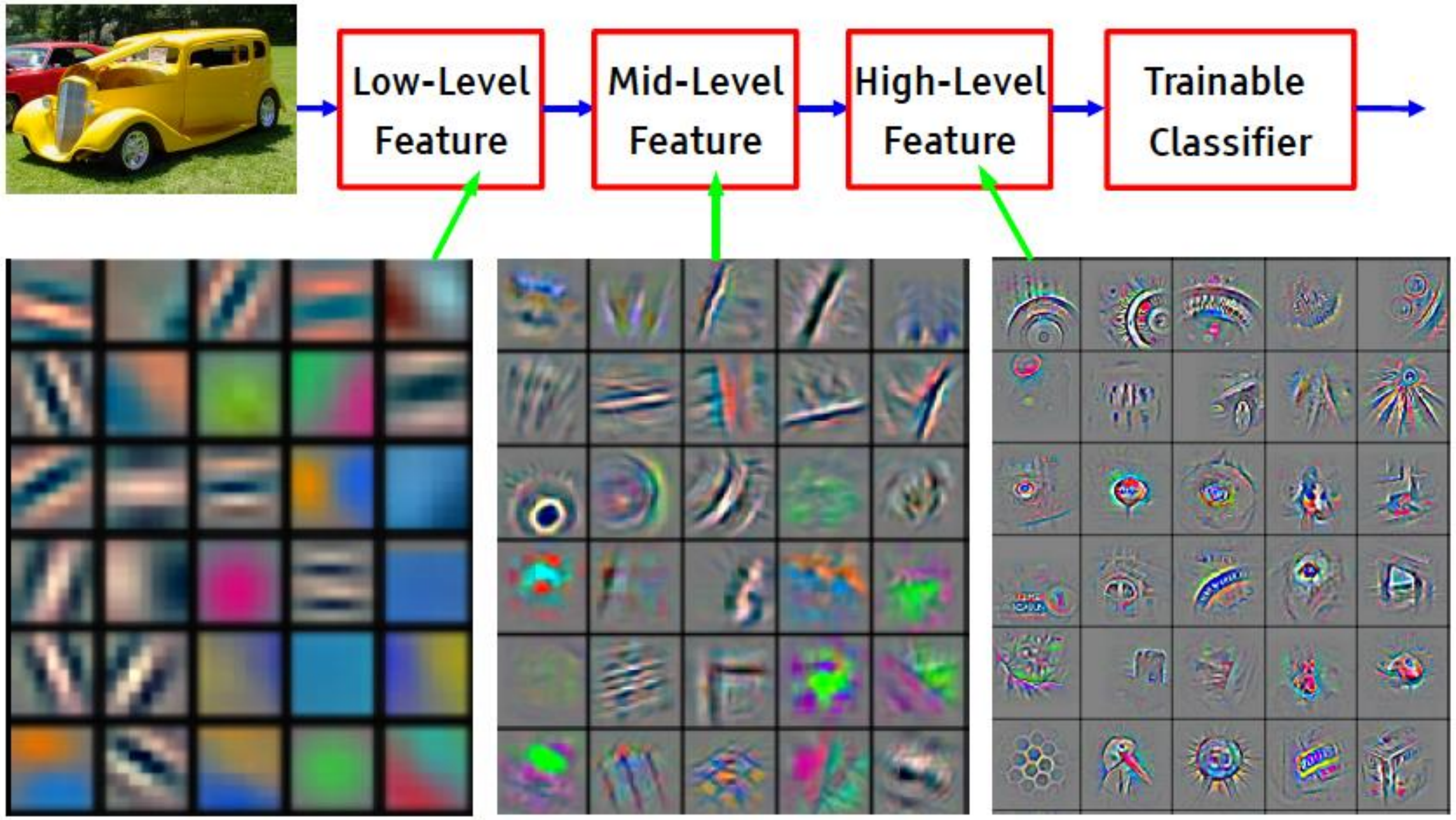
Overview of CNNs

- Fourth idea: Interleaving feature extraction and pooling operations
  - Extracting abstract, compositional features for representing semantic object classes



Overview of CNNs

- Artificial visual pathway: from images to semantic concepts (Representation learning)

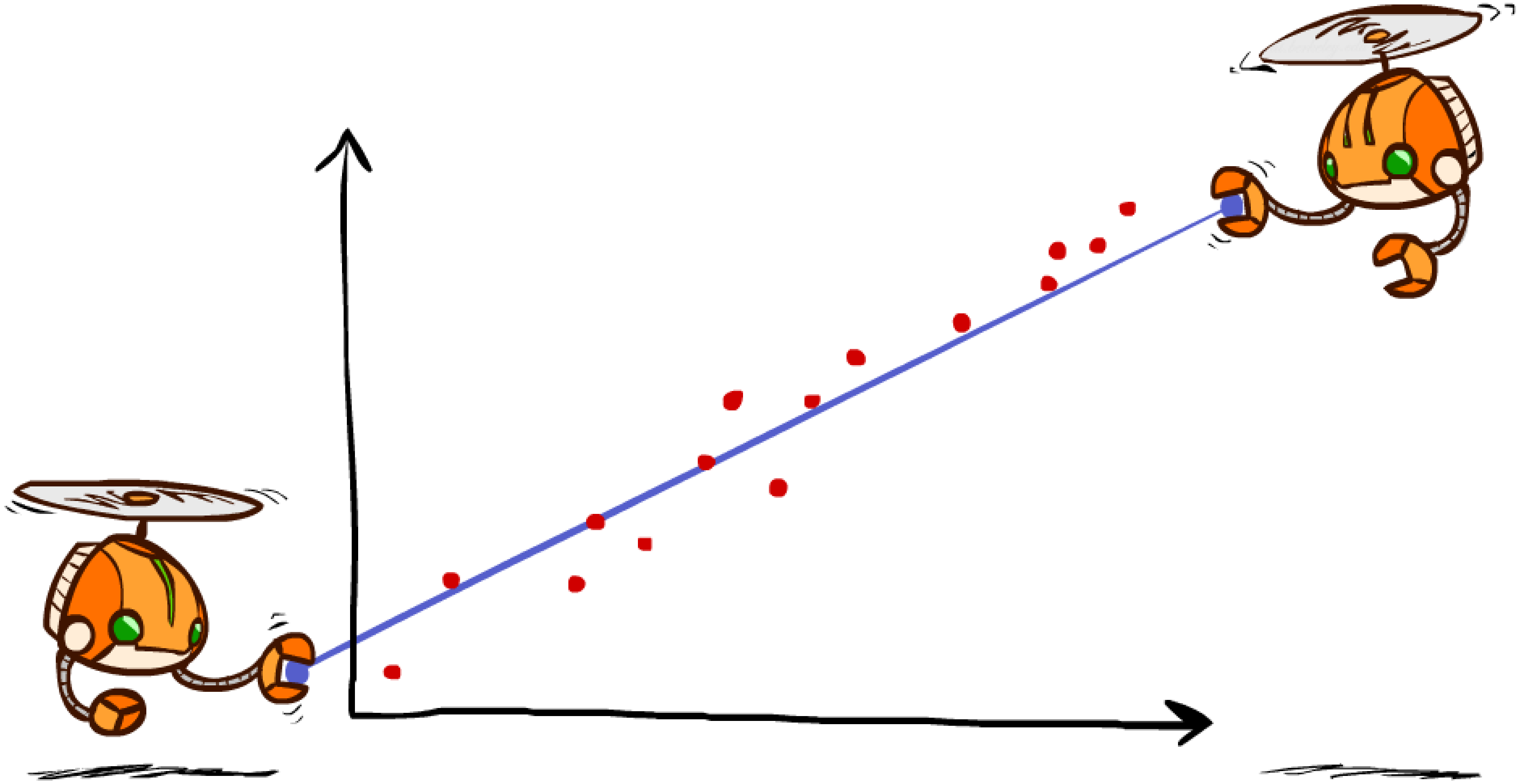


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

More classification methods

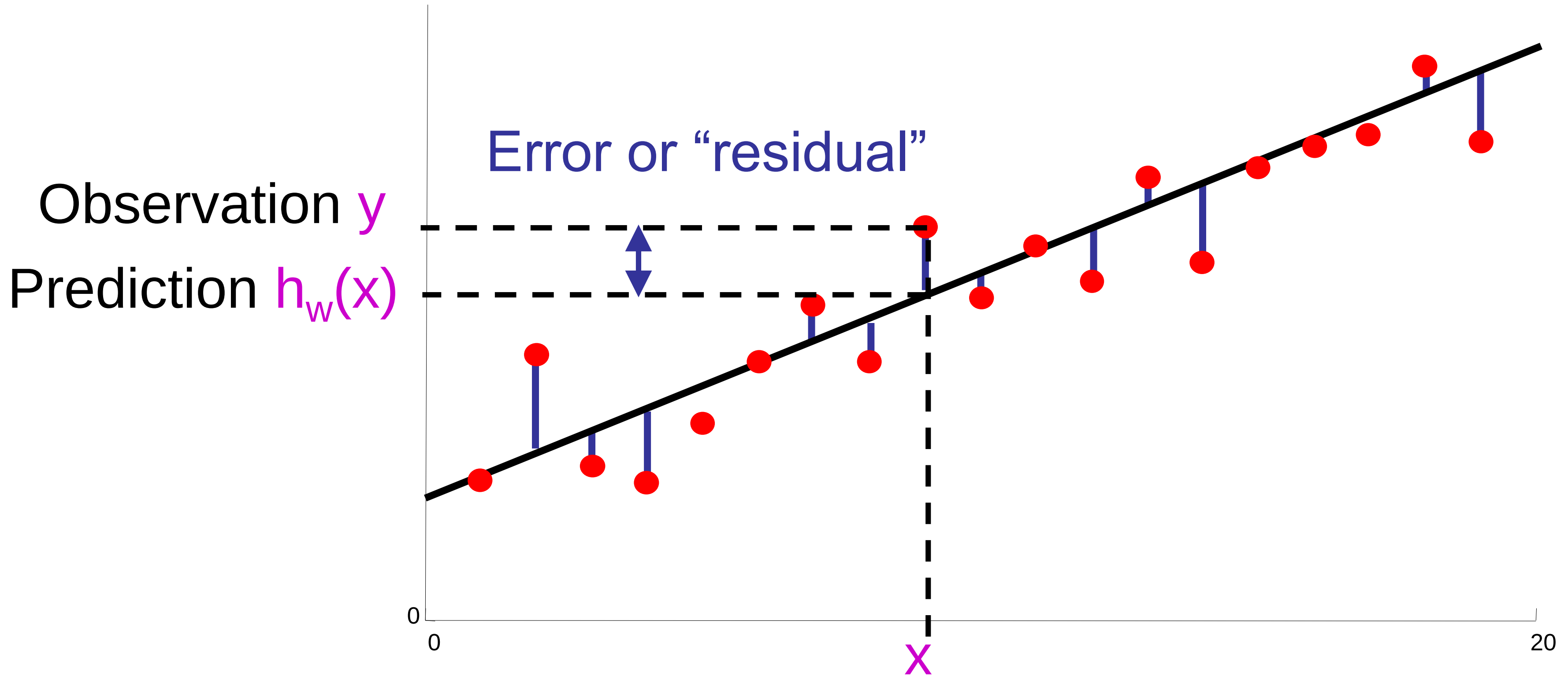
- Naive Bayes
- Perceptron / Neural networks
- Decision trees / Random forest
- Support Vector Machines
- Nearest neighbors
- Model ensembles: bagging, boosting, etc.
- ……

Regression



Linear Regression

Prediction: $h_w(x) = w_0 + w_1x$



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

Least squares: Minimizing squared error

- L2 loss function: sum of squared errors over all examples

$$L(\boldsymbol{w}) = \sum_i \left(y_i - h_w(\boldsymbol{x}_i)\right)^2 = \sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2$$

- We want the weights w* that minimize loss
- Analytical solution: at w* the derivative of loss w.r.t. each weight is zero
  - **X** is the data matrix (all the data, one example per row); **y** is the vector of labels
  - $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

Regularized Regression

- Overfitting is also possible in regression
  - Extreme case: *n* features, *n* training examples
- Regularization can be used to alleviate overfitting

- LASSO (Least Absolute Shrinkage and Selection Operator)

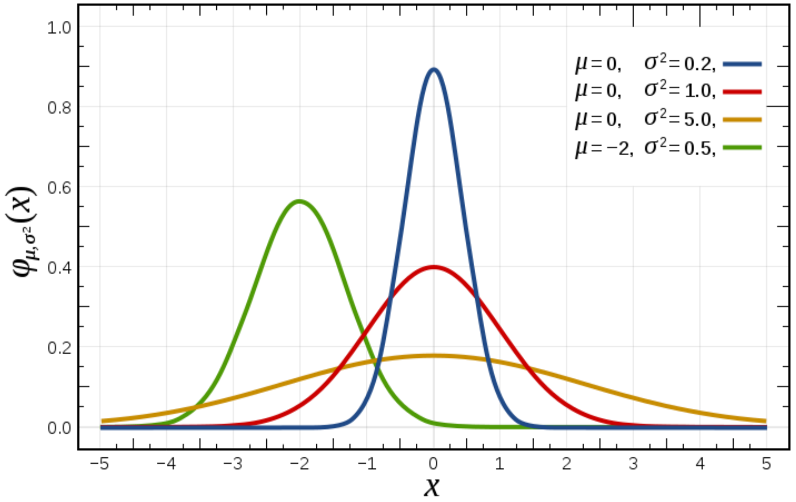
$$L(\boldsymbol{w}) = \sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 + \lambda \sum_k |w_k|$$

- Ridge Regression

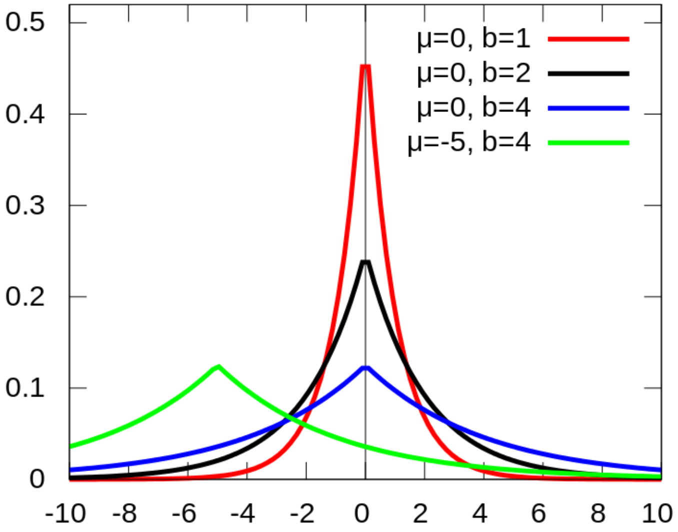
$$L(\boldsymbol{w}) = \sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 + \lambda \sum_k w_k^2$$

Regularized Regression

- L2 regularization = Gaussian distribution as weight prior
  - Small weights



- L1 regularization = Laplace distribution as weight prior
  - Long-tailed distribution
  - Zero weights + sparse large weights



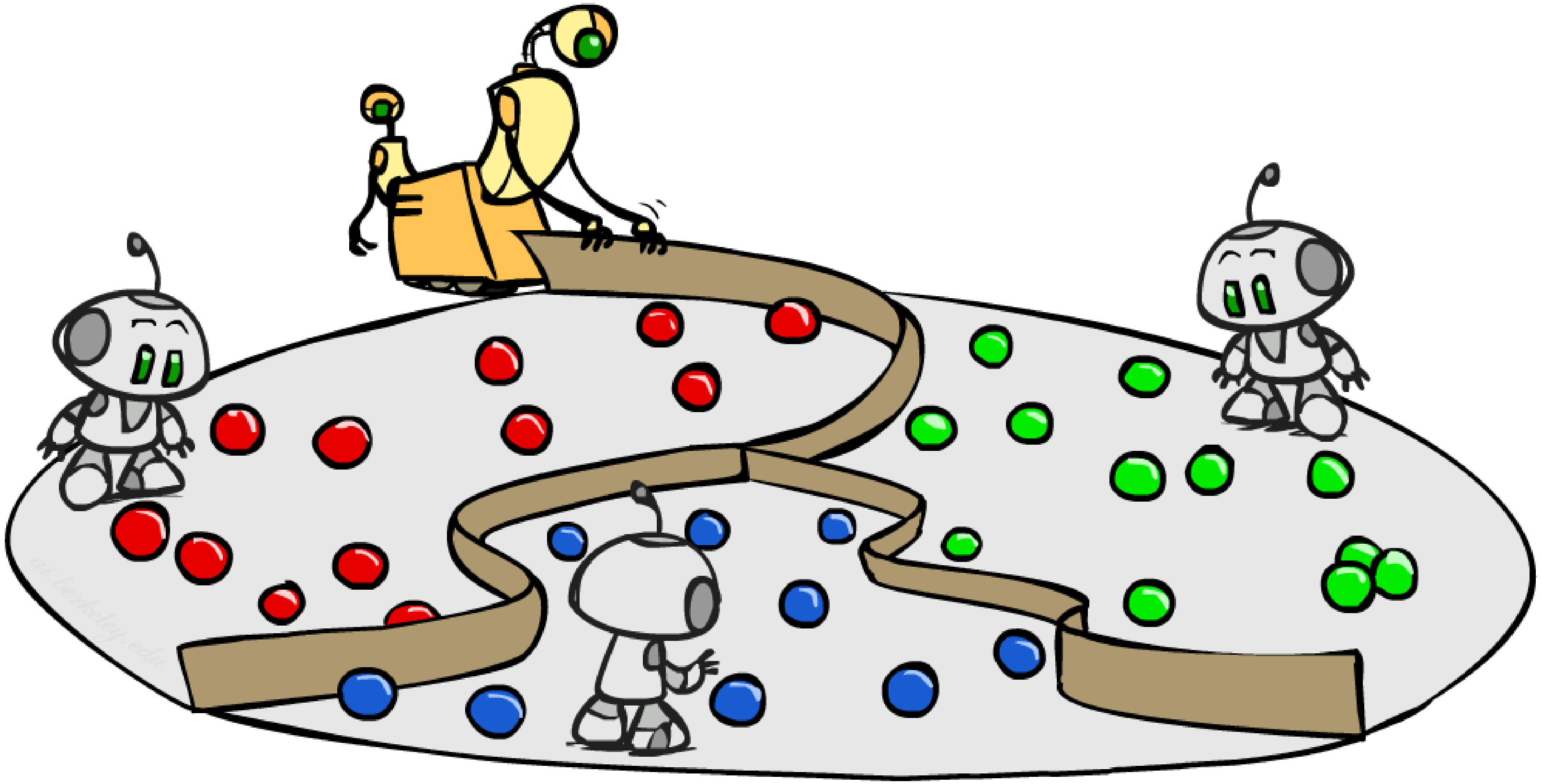
Non-linear least squares

- No closed-form solution in general
- Numerical algorithms are typically used
  - Choose initial values for the parameters and then refine the parameters iteratively
  - Gradient descent
  - Gauss–Newton method
  - Limited-memory BFGS
  - Derivative-free methods
  - etc.

Summary

- Supervised learning:
  - Learning a function from labeled examples
- Classification: discrete-valued function
  - Naïve Bayes
  - Generalization and overfitting, smoothing
  - Perceptron
- Regression: real-valued function
  - Linear regression

Unsupervised Machine Learning



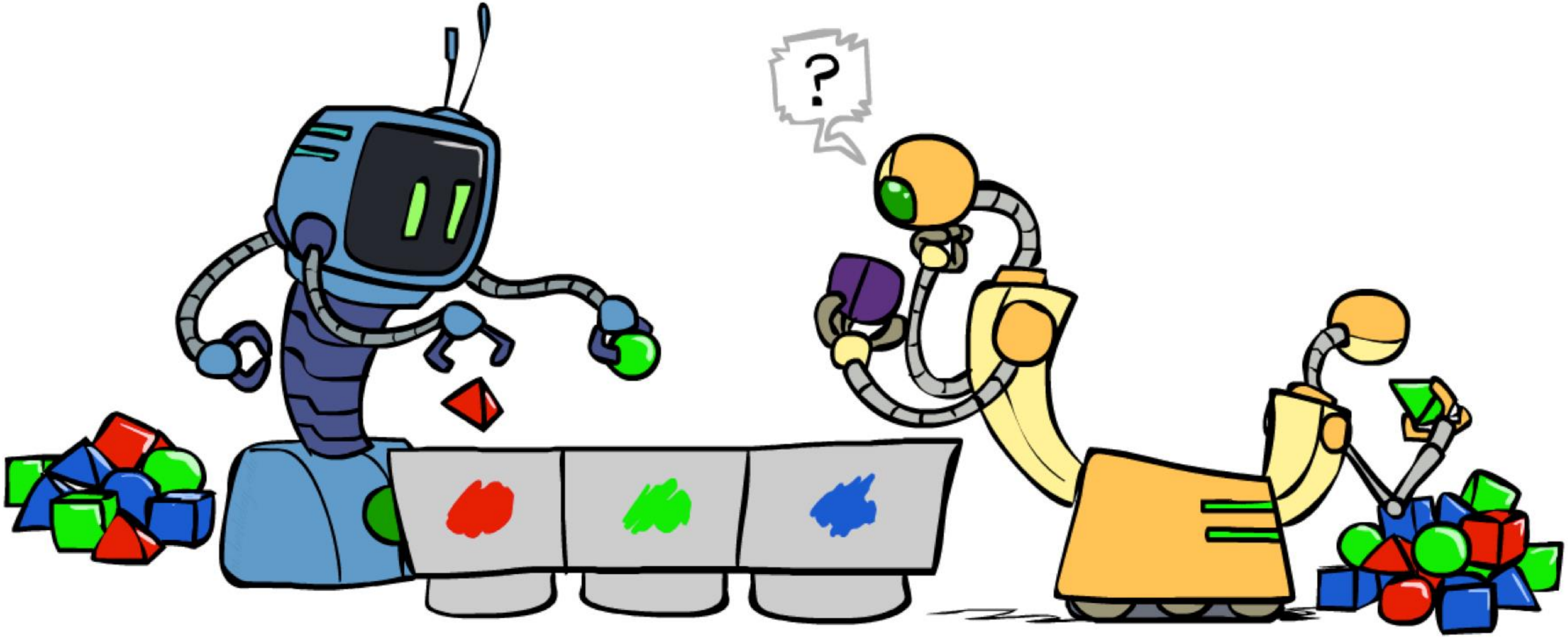
AIMA Chapter 20

[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley and from Daniel Weld, Carlos Guestrin, & Luke Zettlemoyer at U Washington]

Types of Learning

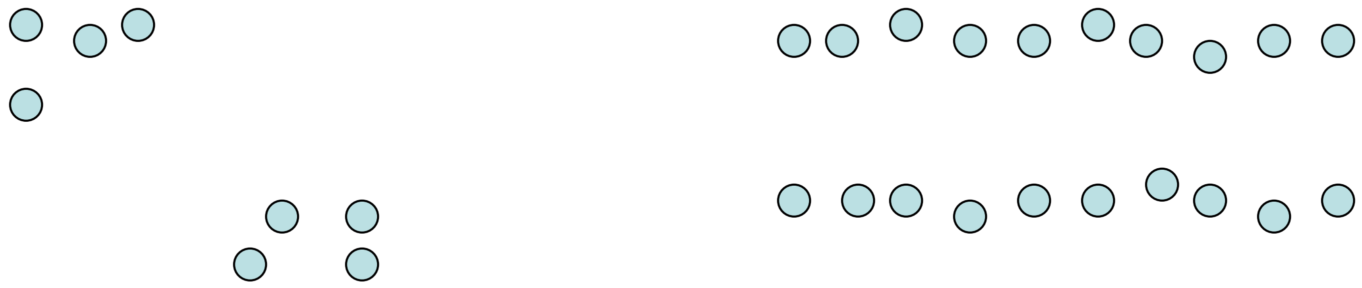
- Supervised learning
  - Training data includes desired outputs
- Unsupervised learning
  - Training data does not include desired outputs
- Semi-supervised learning
  - Training data includes a few desired outputs
- Reinforcement learning
  - Rewards from sequence of actions

Clustering



Clustering

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



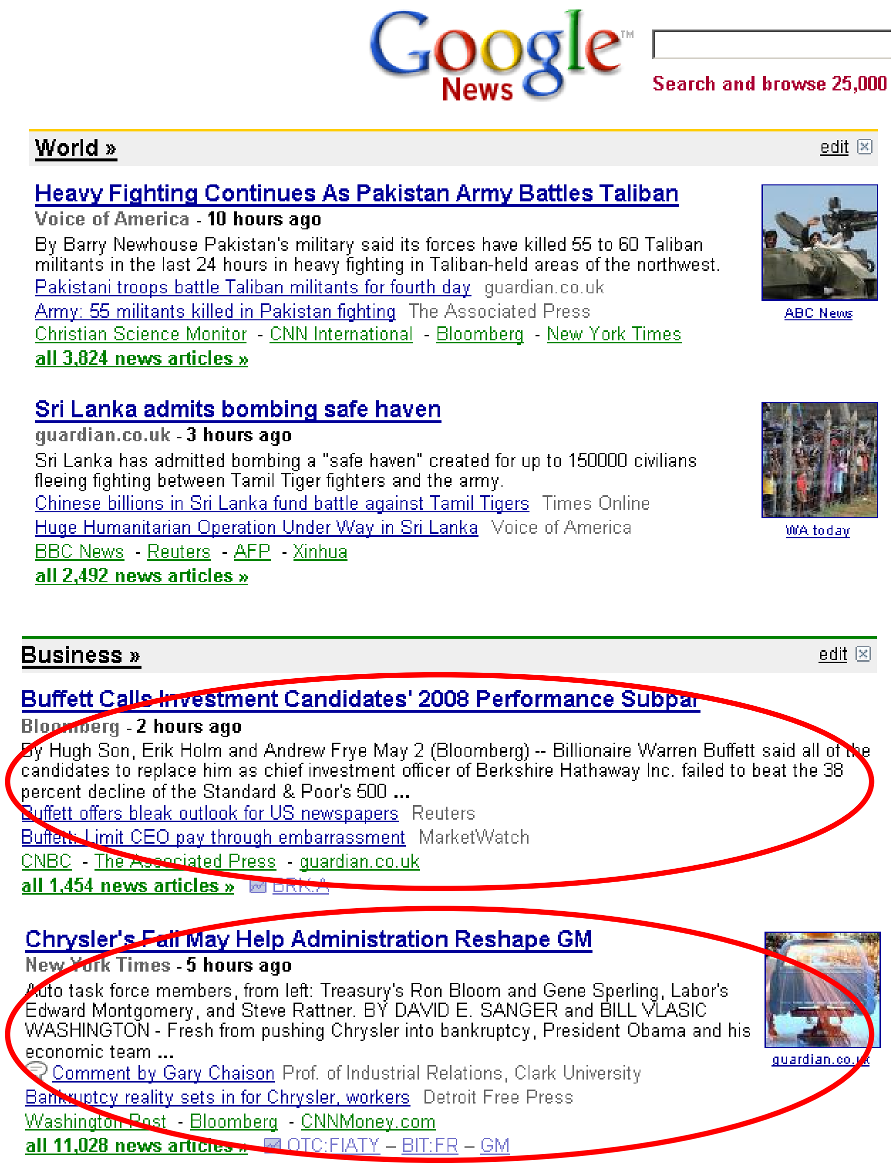
- What could "similar" mean?
  - One option: small (squared) Euclidean distance

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

  - Many other options, often domain specific

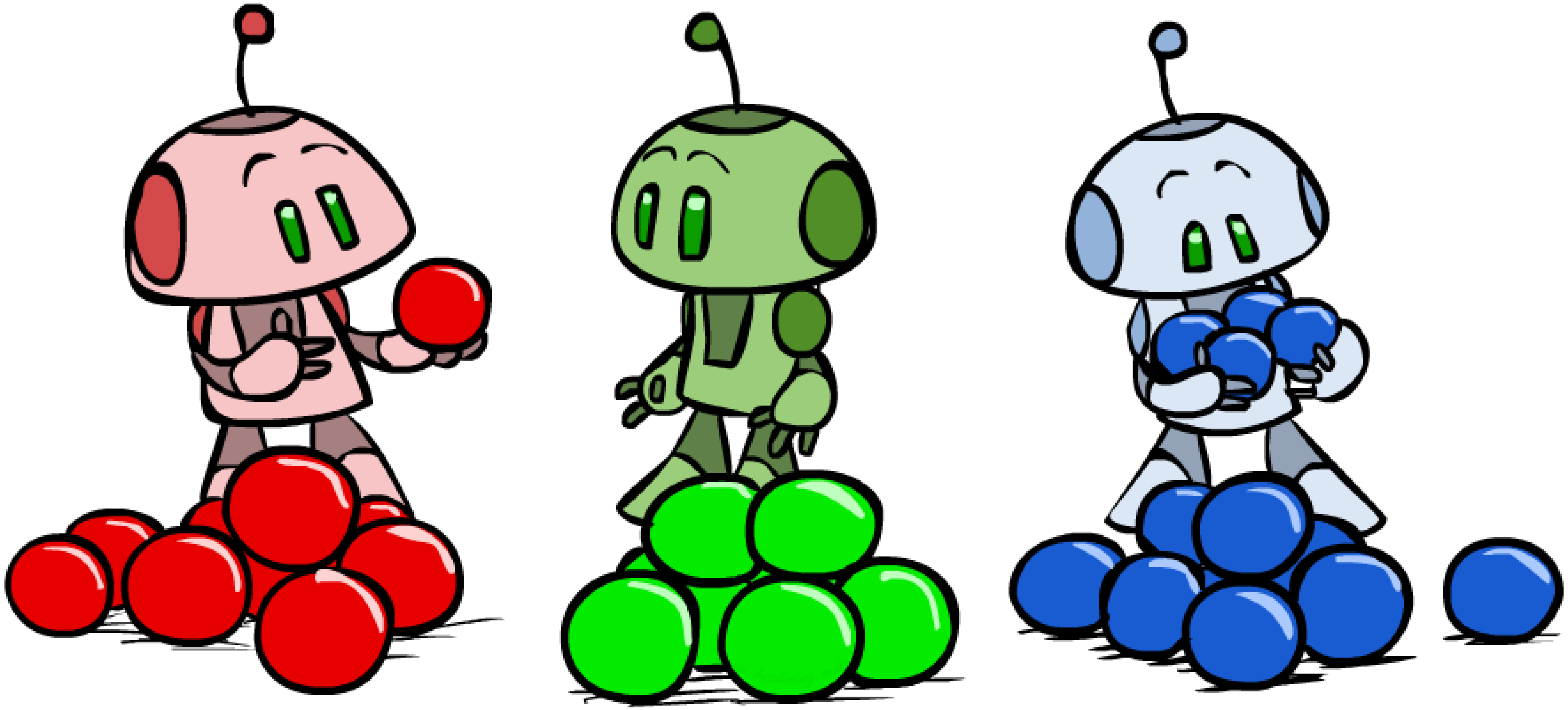
Clustering

- Applications
  - Group emails
  - Group search results
  - Find categories of customers
  - Detect anomalous program executions



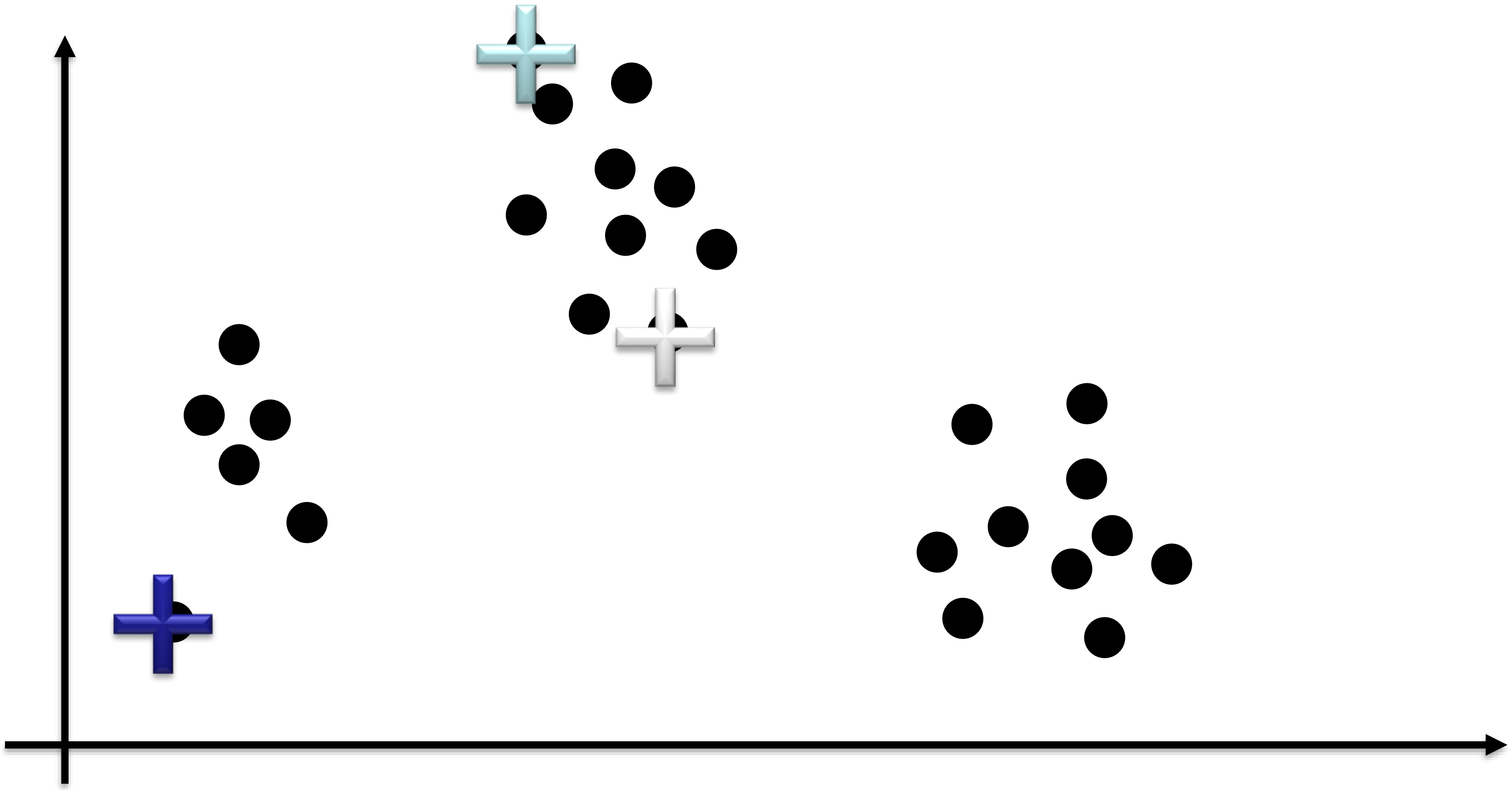
Story groupings: unsupervised clustering

K-Means



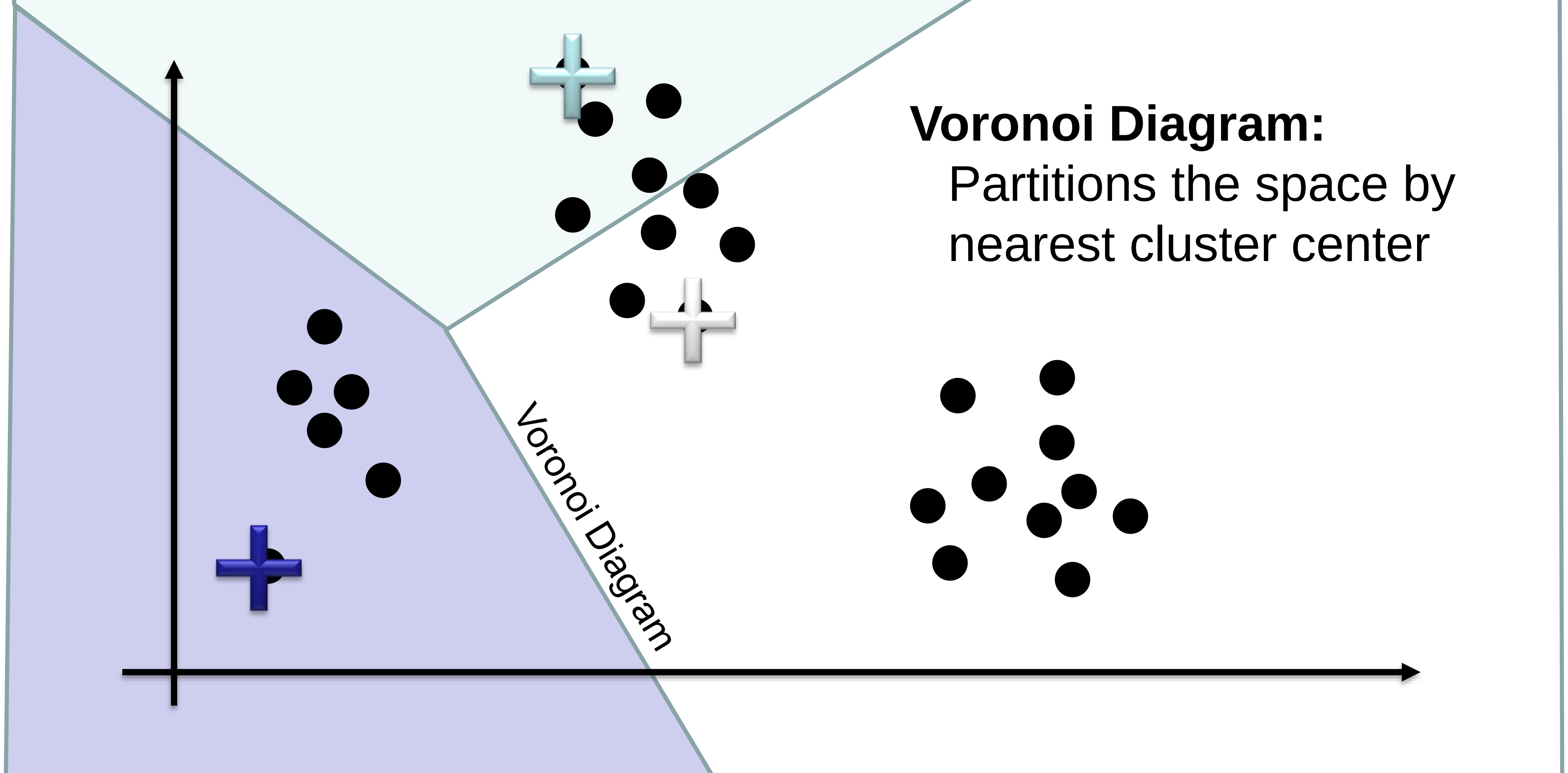
K-Means Clustering: *Intuition*

- Input K: The number of clusters to find
- Pick an initial set of points as cluster centers



K-Means Clustering: *Intuition*

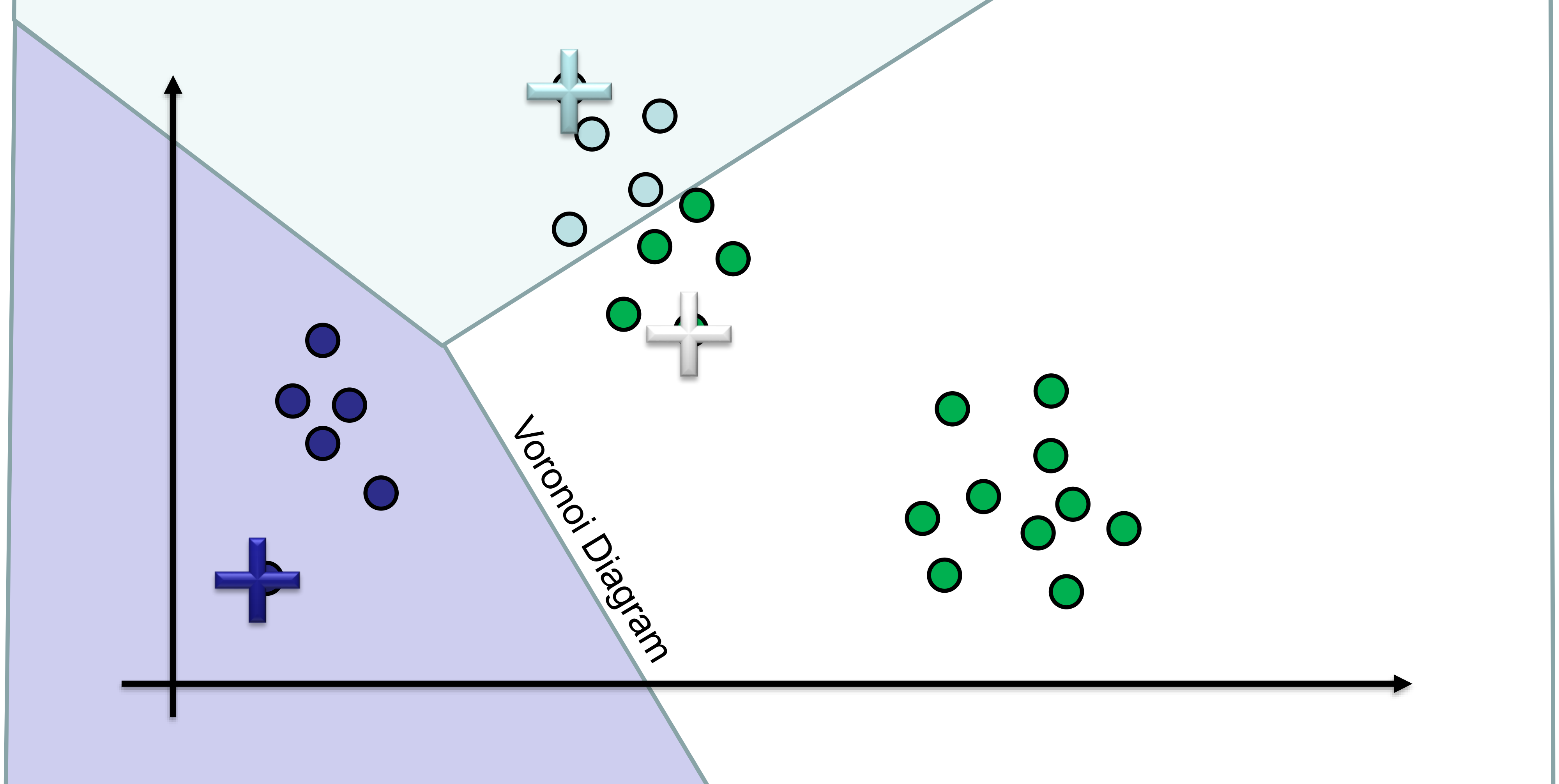
- For each data point find the cluster nearest center



Voronoi Diagram:
Partitions the space by nearest cluster center

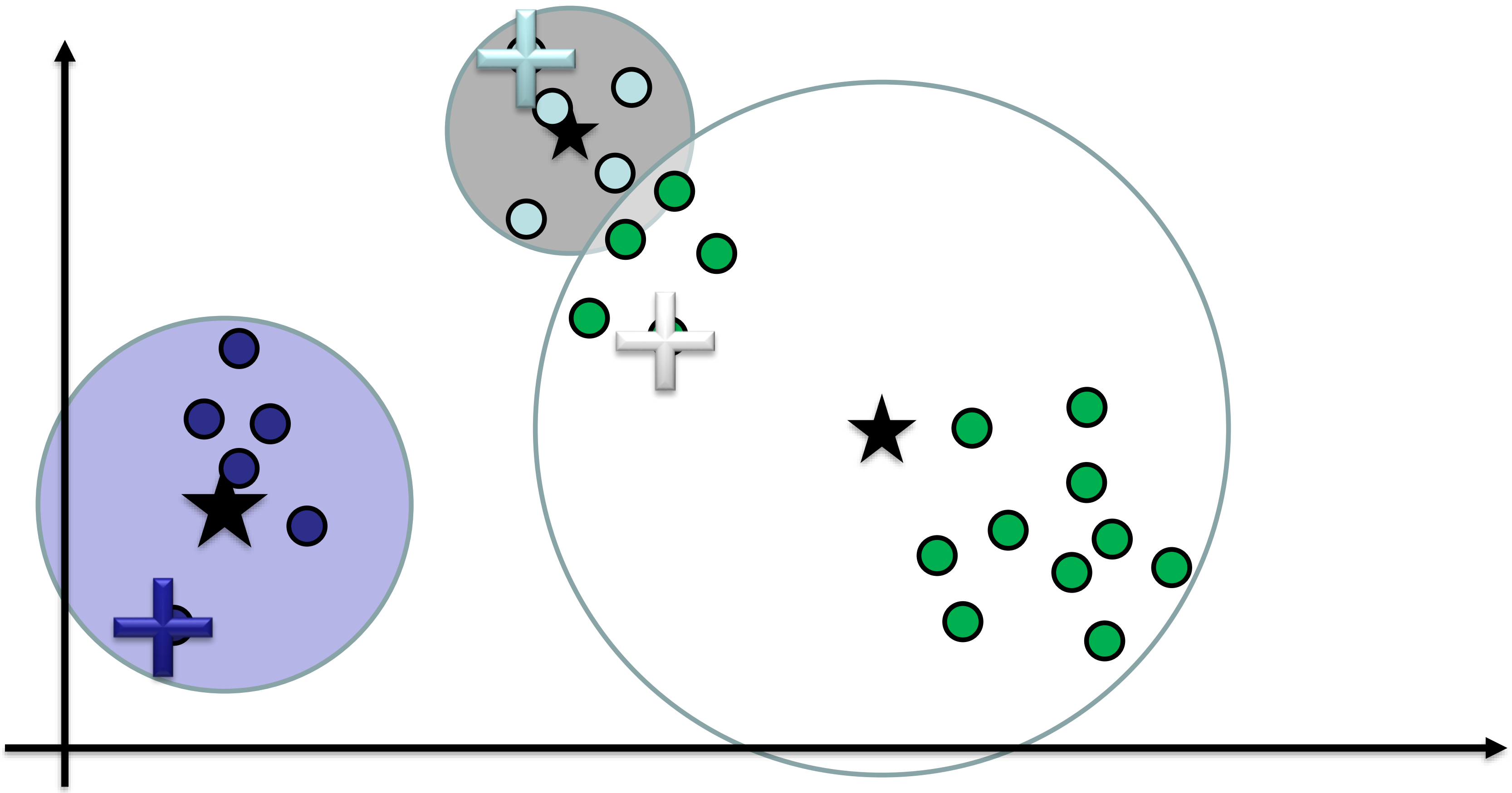
K-Means Clustering: *Intuition*

- For each data point find the cluster nearest center



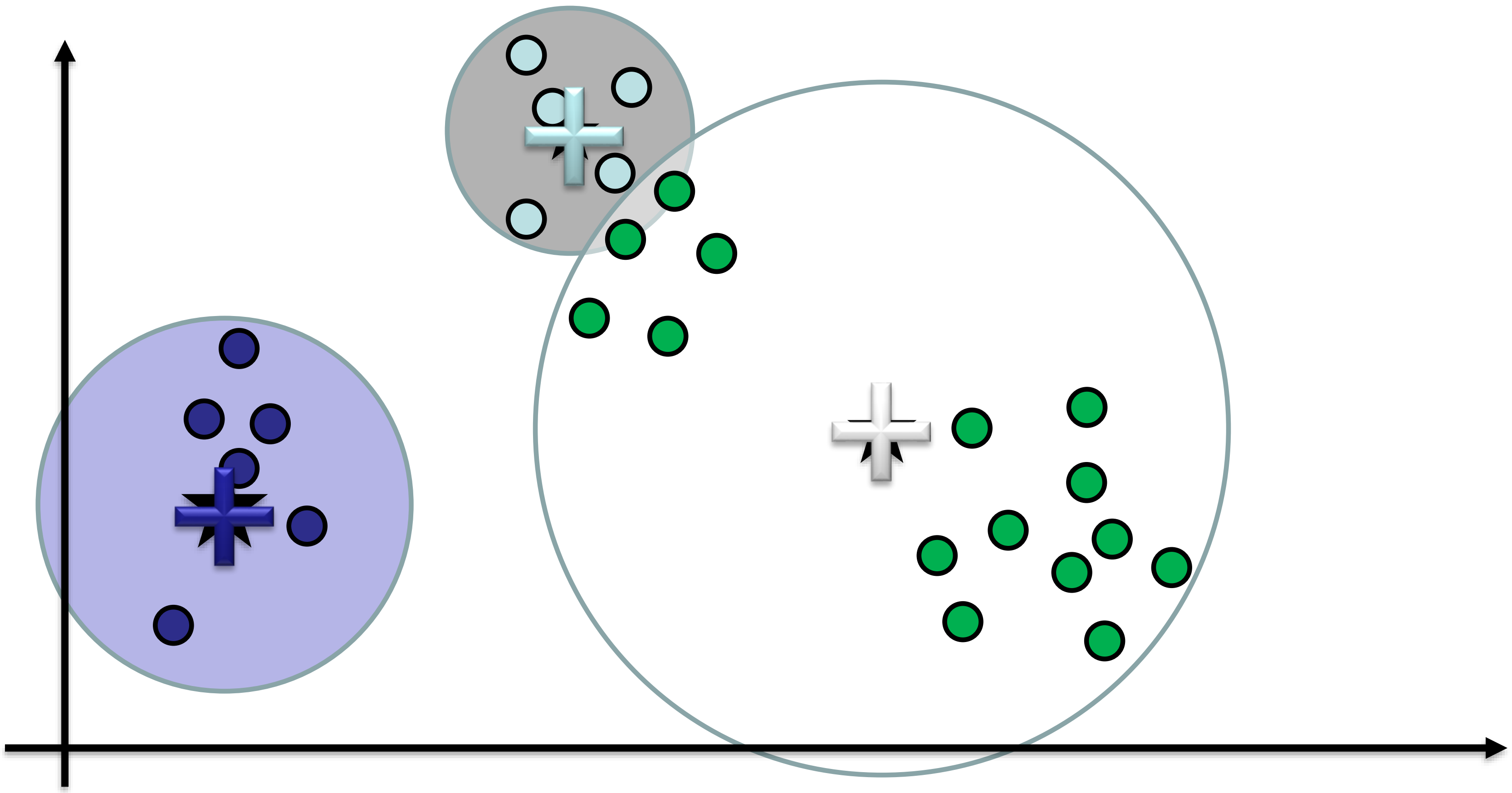
K-Means Clustering: *Intuition*

- Compute mean of points in each "cluster"



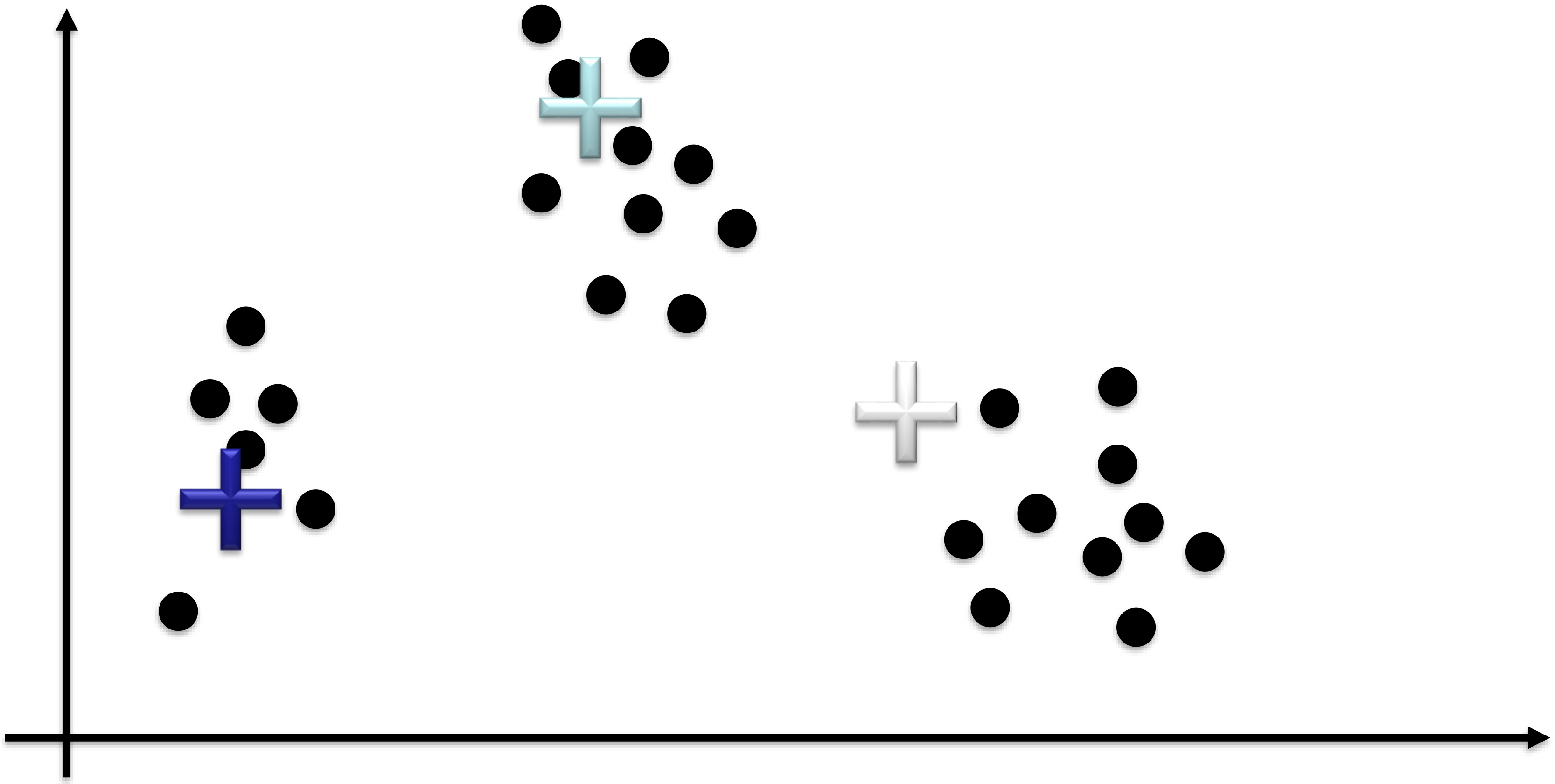
K-Means Clustering: *Intuition*

- Adjust cluster centers to be the mean of the cluster



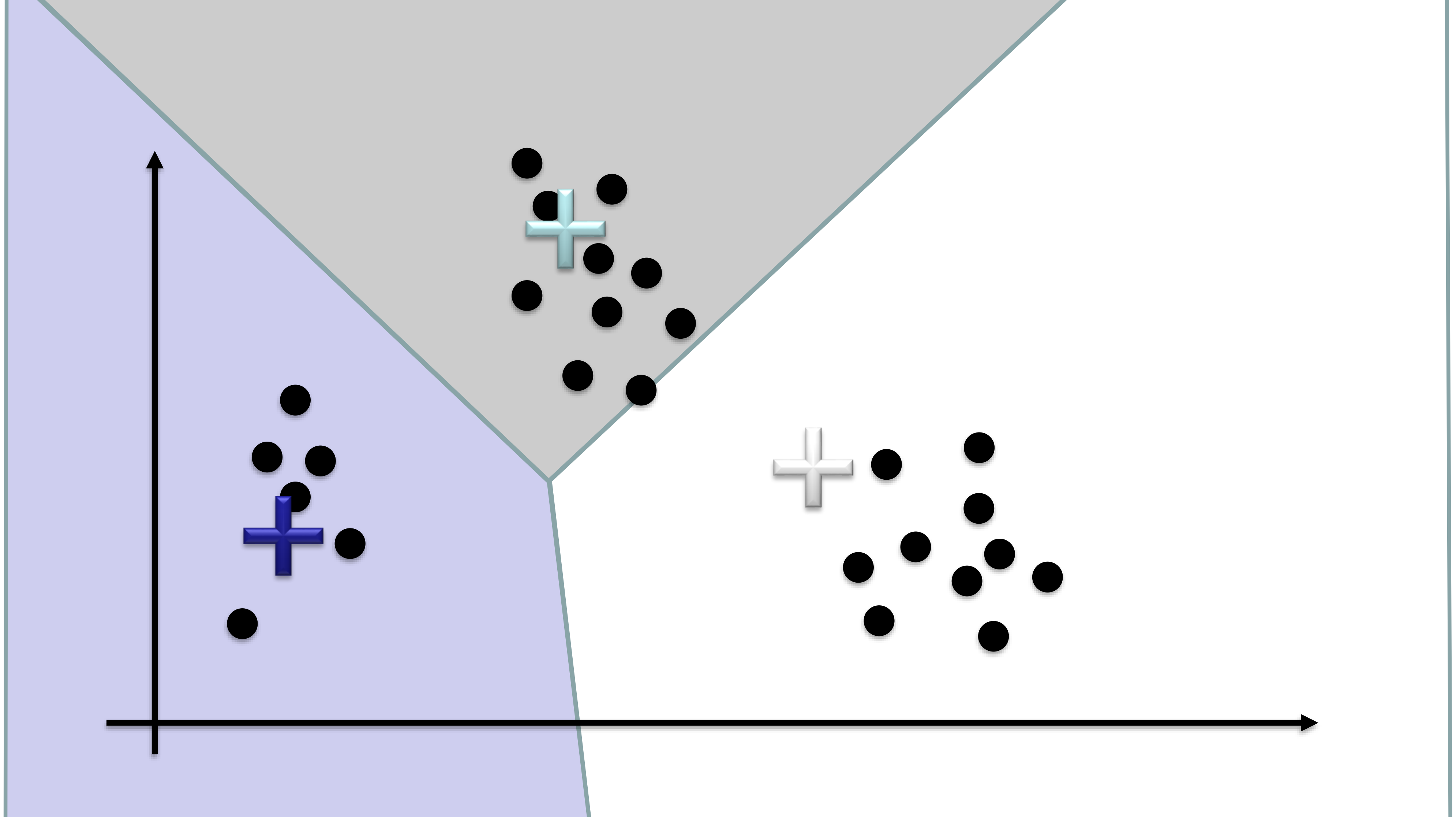
K-Means Clustering: *Intuition*

- Improved?
- Repeat



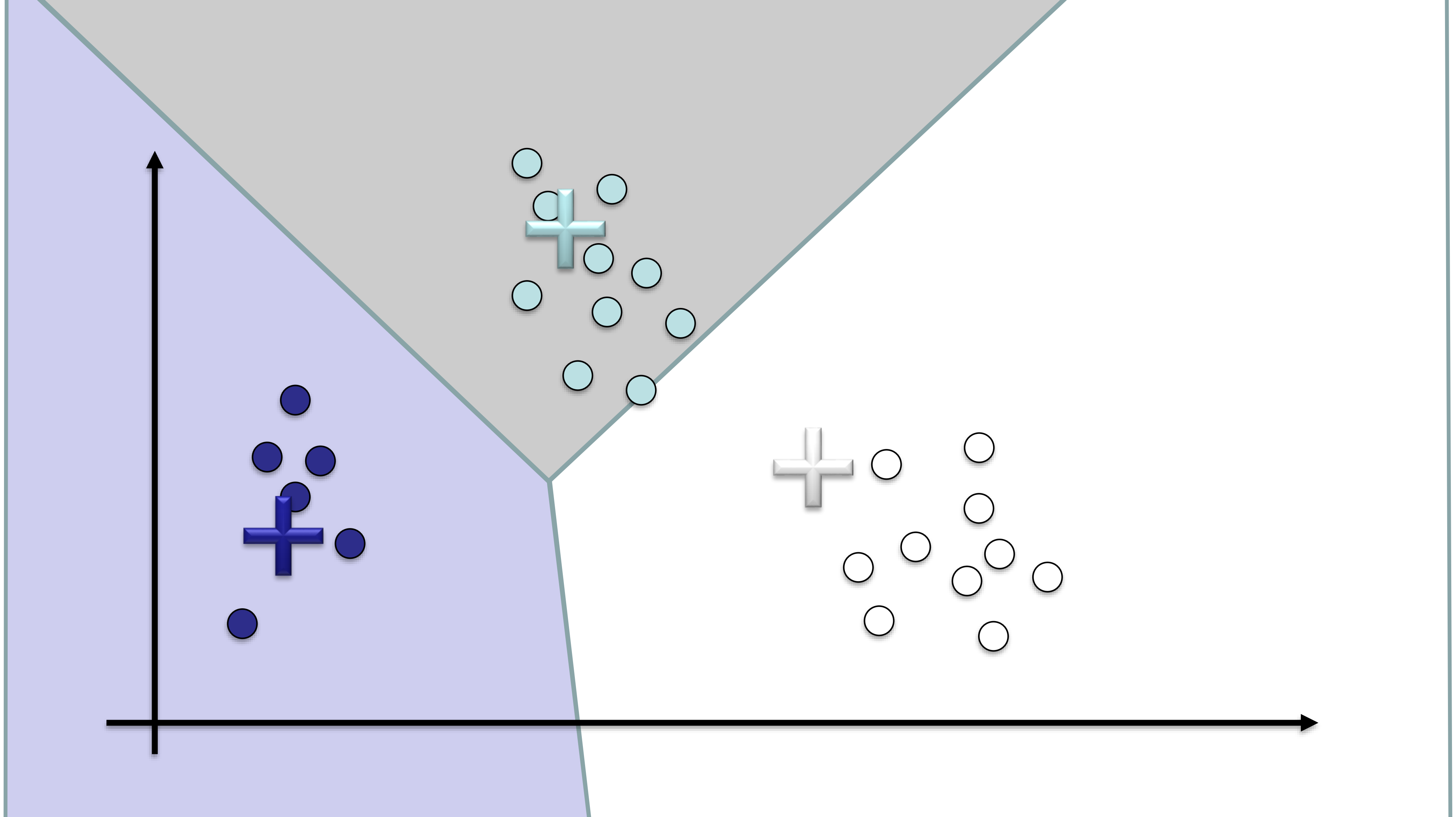
K-Means Clustering: *Intuition*

- Assign Points



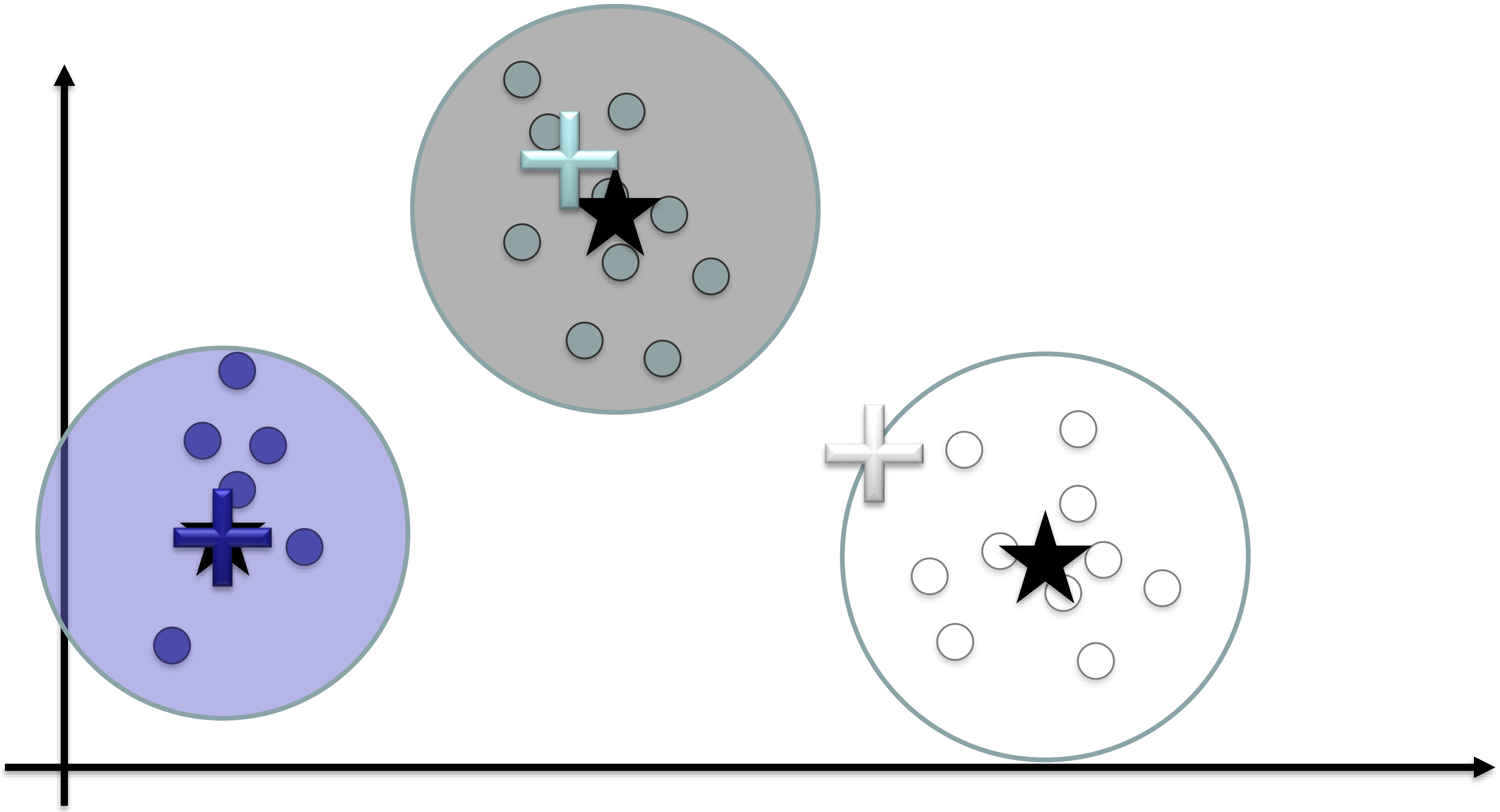
K-Means Clustering: *Intuition*

- Assign Points



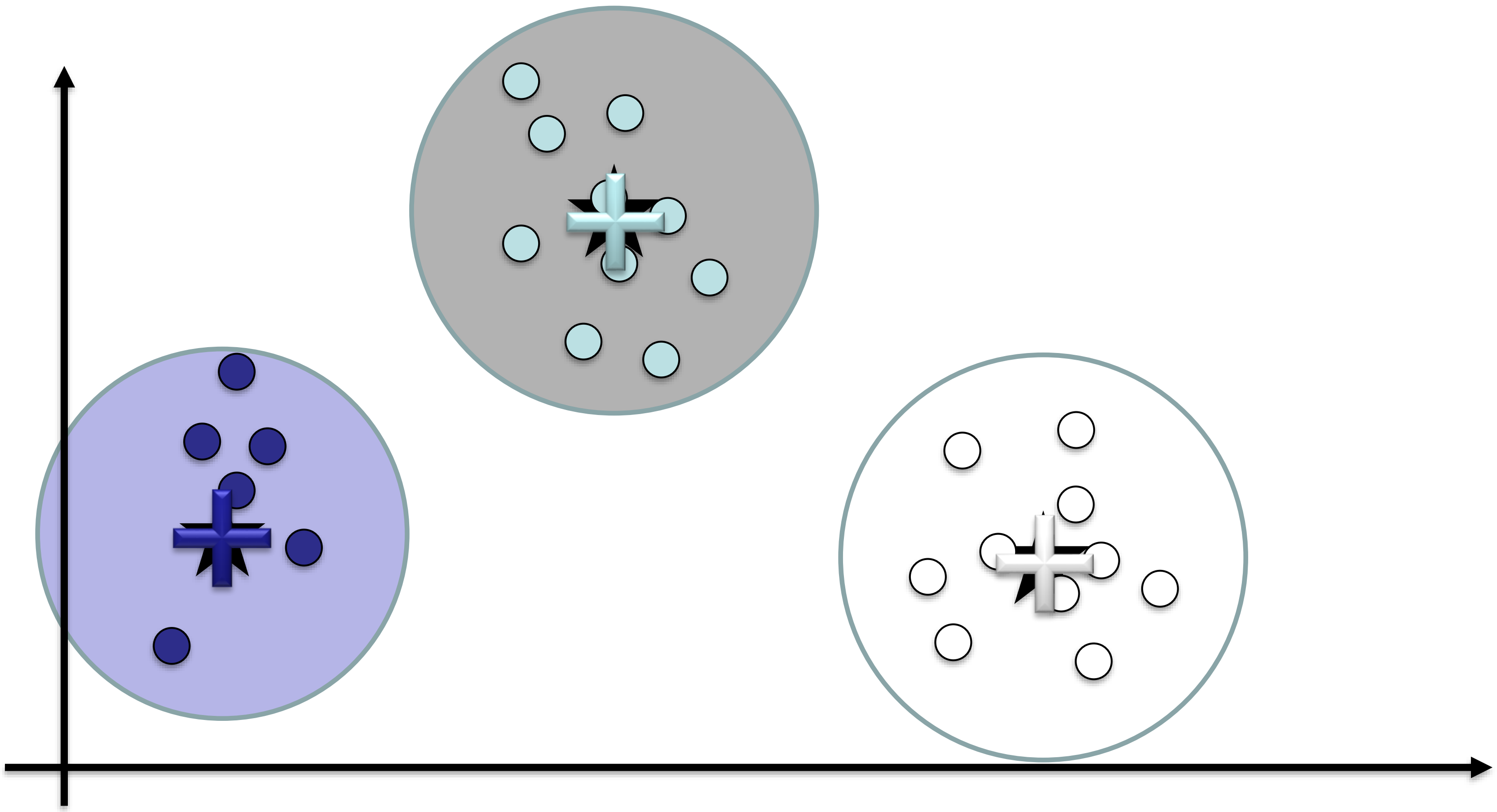
K-Means Clustering: *Intuition*

- Compute cluster means



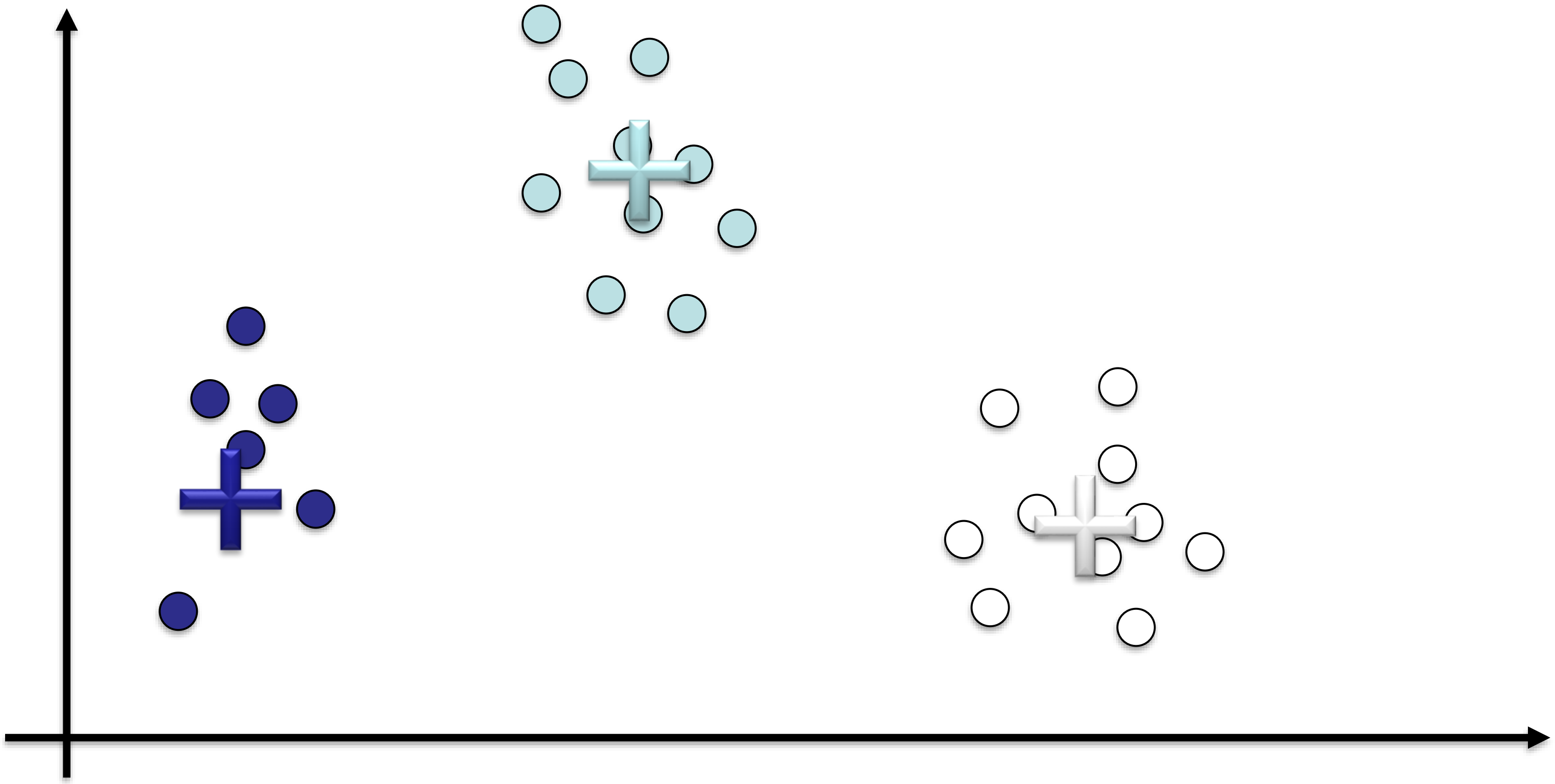
K-Means Clustering: *Intuition*

- Update cluster centers



K-Means Clustering: *Intuition*

- Repeat?
  - Yes to check that nothing changes → Converged!



K-Means as Optimization

- Consider the total distance to the means:

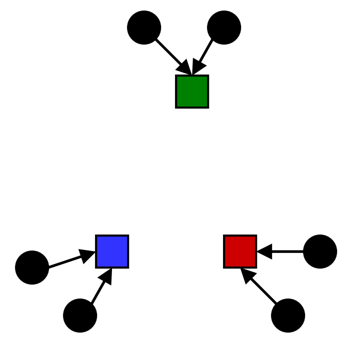
$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \mathsf{dist}(x_i, c_{a_i})$$

points

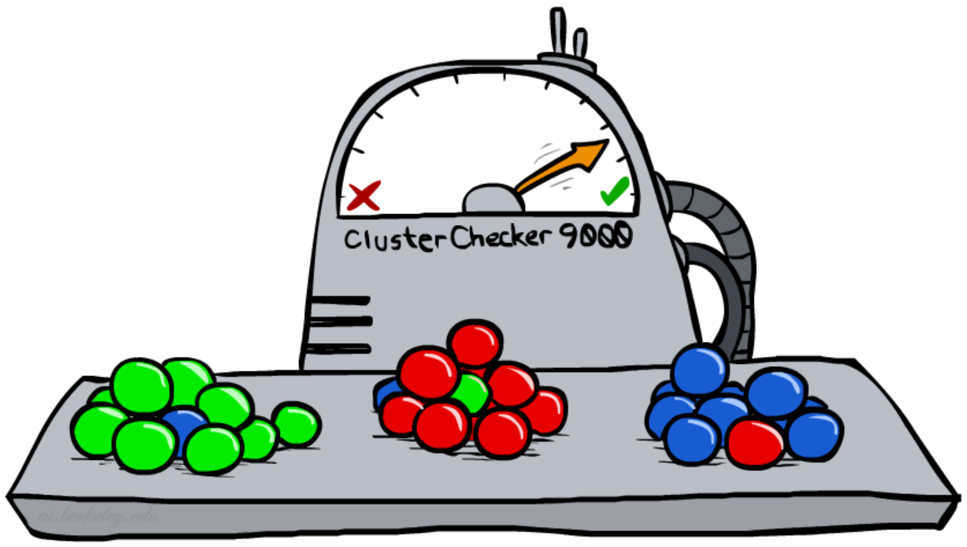
assignments

means

squared Euclidean distance



- Two stages each iteration:
  - Update assignments: fix means c, change assignments a
  - Update means: fix assignments a, change means c
- Each step cannot increase phi



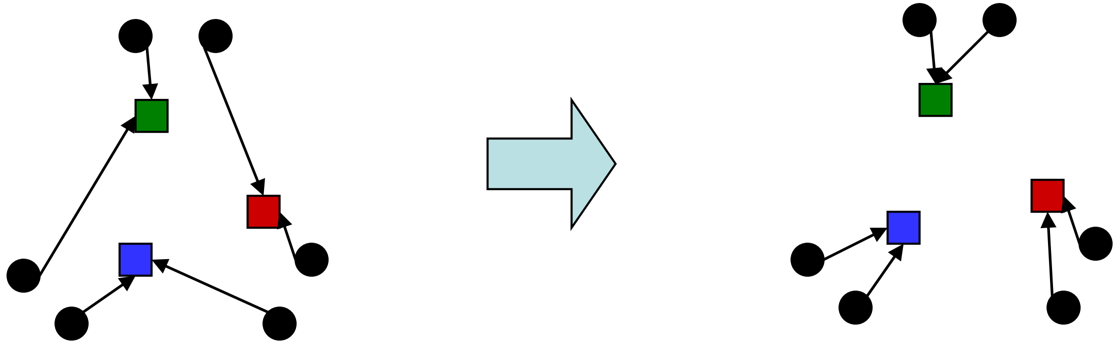
Phase I: Update Assignments

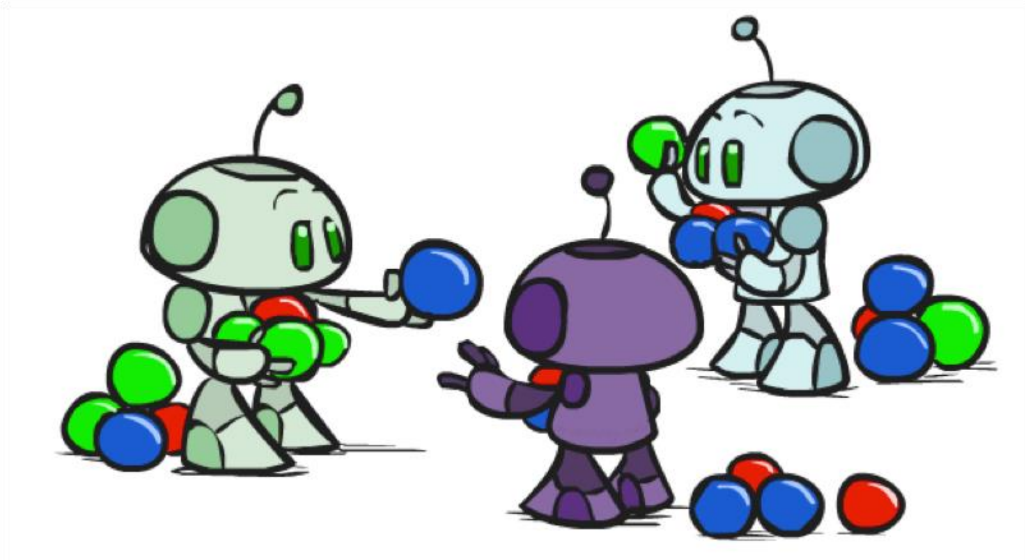
- For each point, re-assign to closest mean:

$$a_i = \underset{k}{\text{argmin}}\, \text{dist}(x_i, c_k)$$

- Cannot increase total distance phi!

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



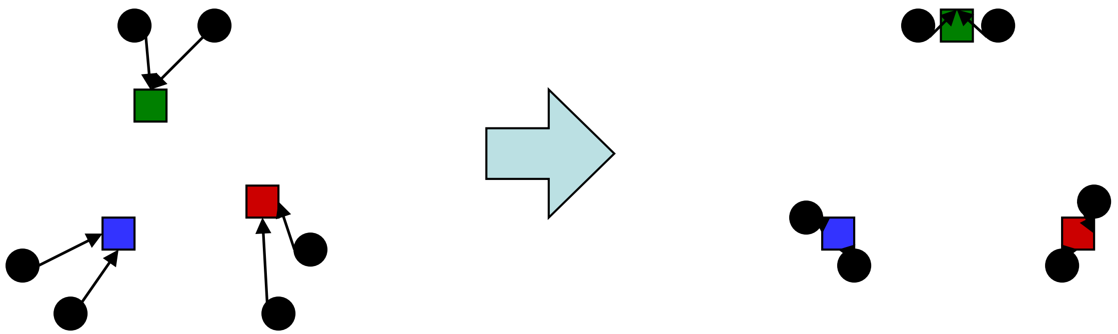


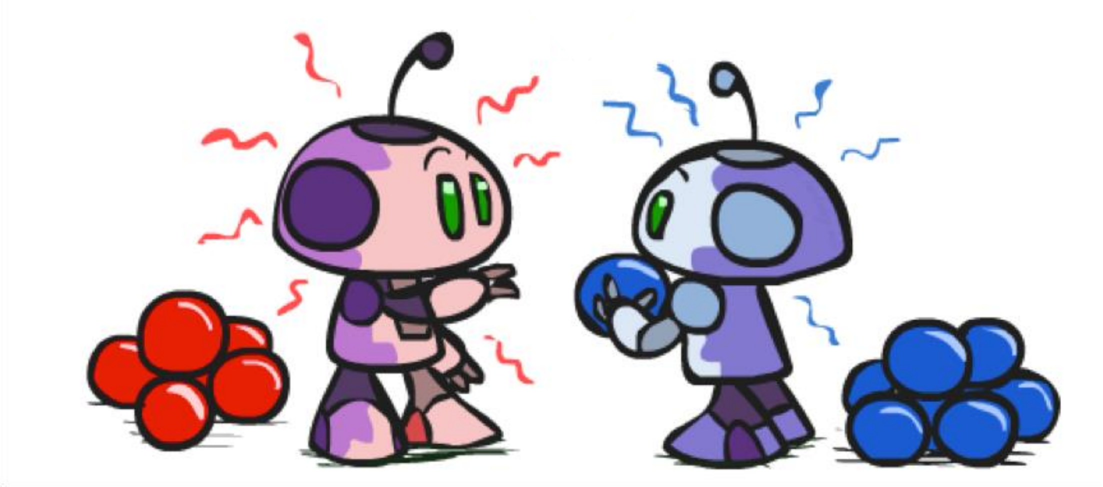
Phase II: Update Means

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

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

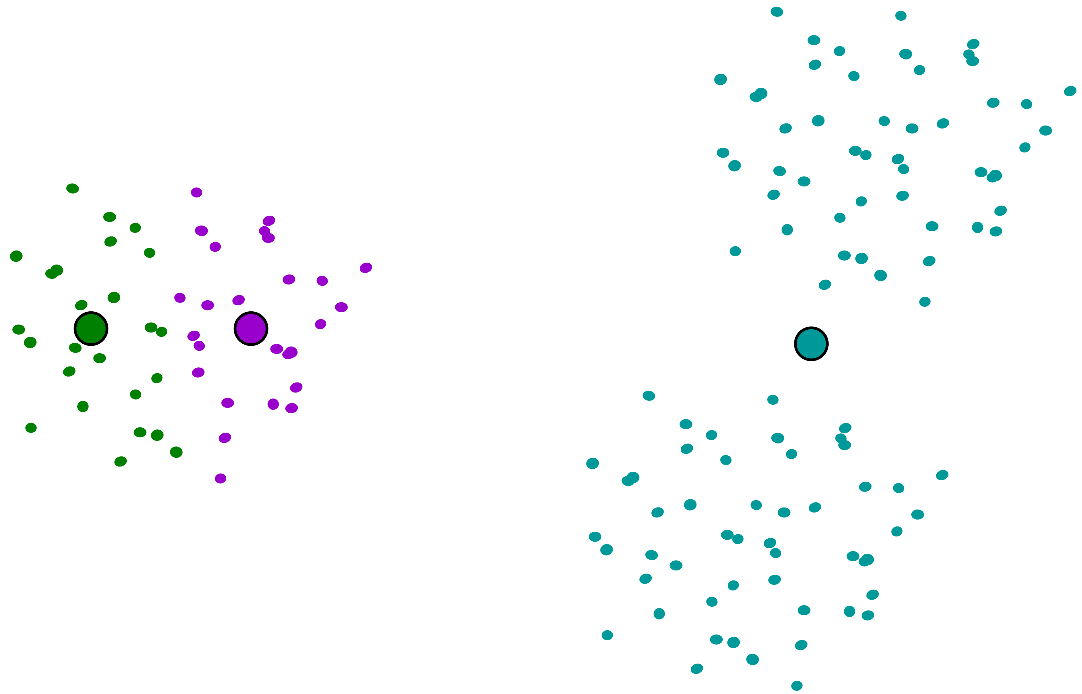
- Also cannot increase total distance
  - Fun fact: the point y with minimum squared Euclidean distance to a set of points {x} is their mean

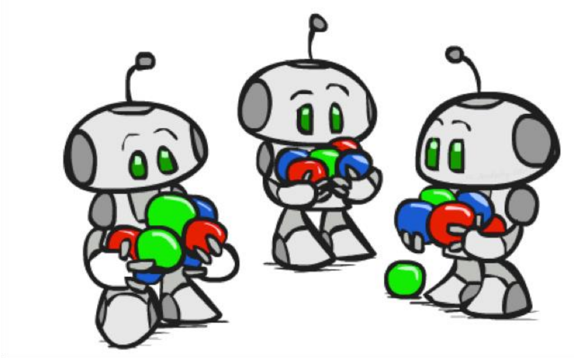


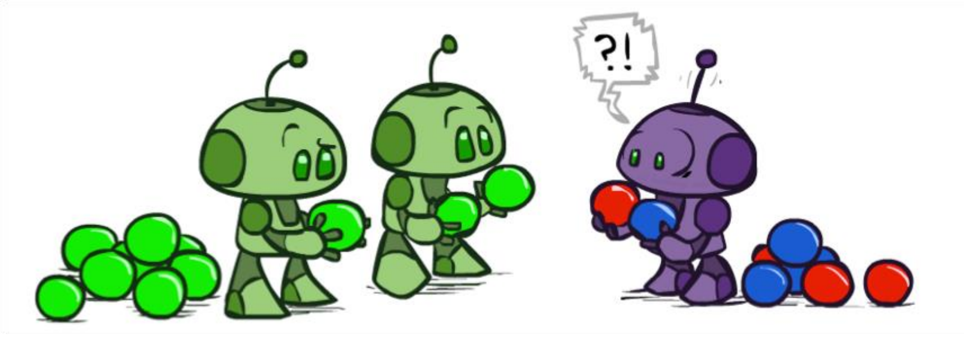


Initialization

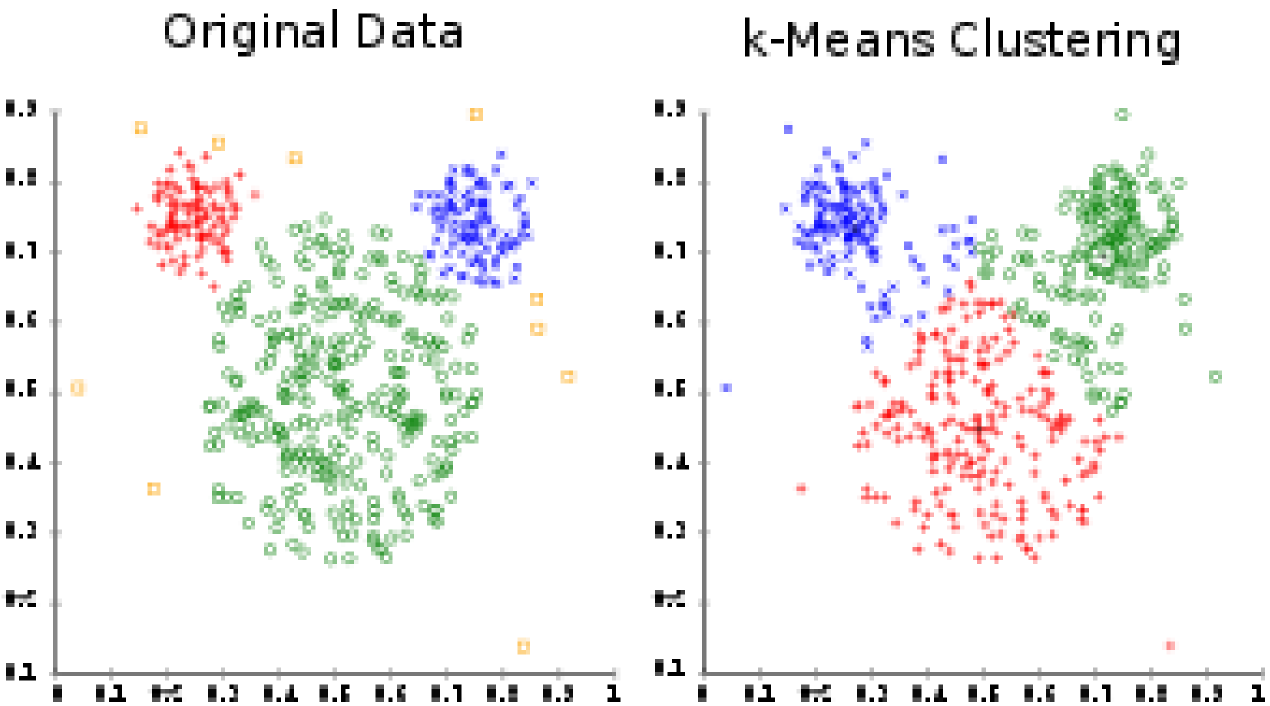
- K-means is non-deterministic
  - Requires initial means
  - It does matter what you pick!
  - What can go wrong?
    - Local optima



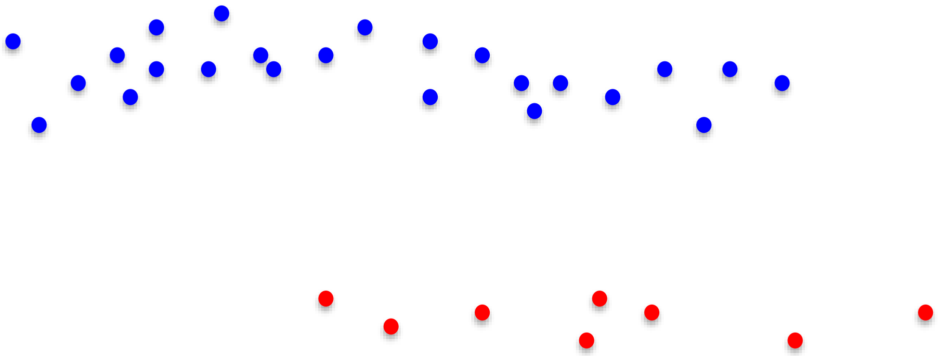




Inductive Bias



Equally Sized Clusters



Circular Clusters

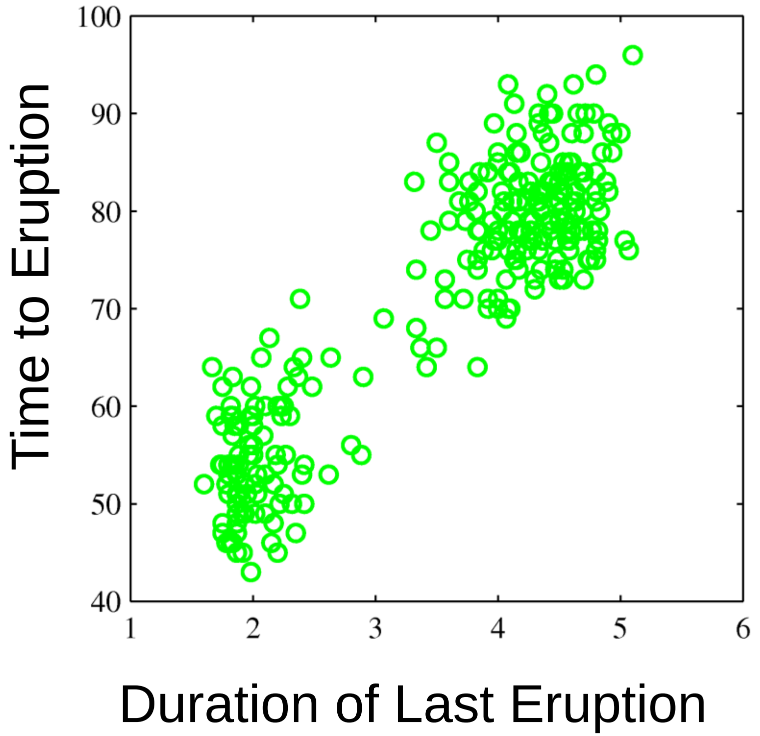
Probabilistic Clustering

- Try a probabilistic model!
  - allows overlaps, clusters of different sizes/shapes, etc.

- Gaussian mixture model (GMM)
  - also called Mixture of Gaussians

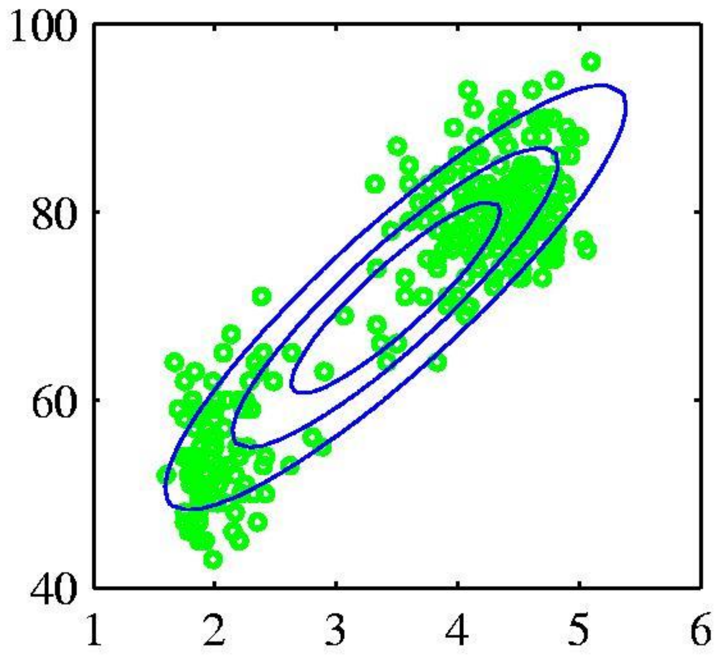
Mixtures of Gaussians

- Old Faithful Data Set

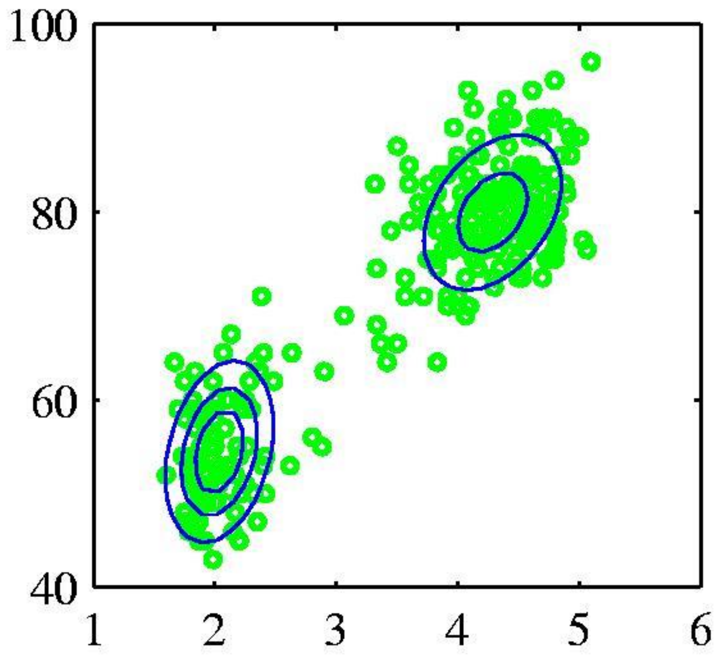


Mixtures of Gaussians

- Old Faithful Data Set



Single Gaussian



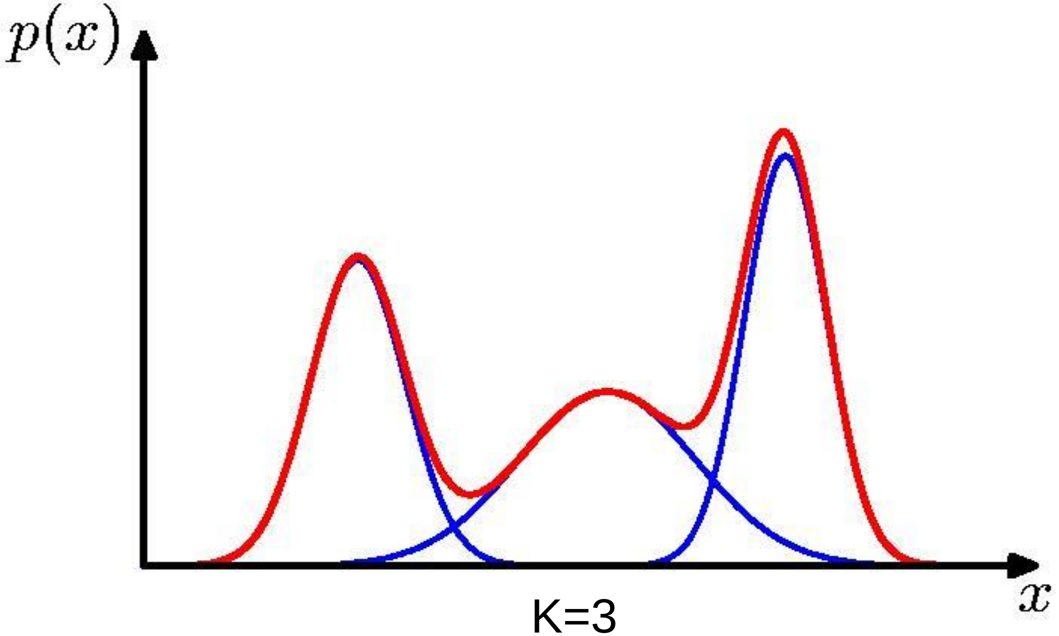
Mixture of two Gaussians

Mixtures of Gaussians

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \underbrace{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}_{\text{Component}}$$

Mixing coefficient

$$\forall k : \pi_k \geqslant 0 \qquad \sum_{k=1}^{K} \pi_k = 1$$



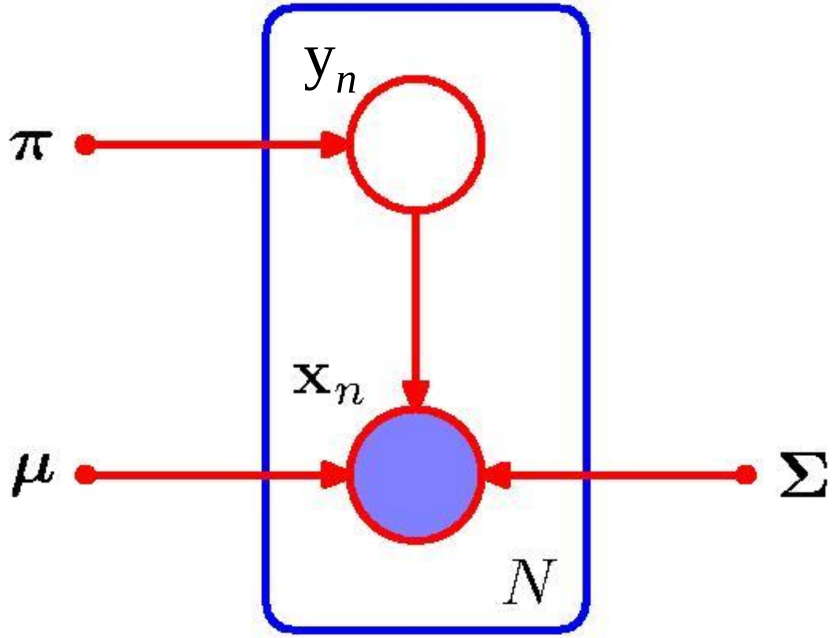
K=3

Gaussian mixture model

- P(Y): Distribution over *k* components (clusters)
- P(X|Y): Each component generates data from a **multivariate Gaussian** with mean μ_i and covariance matrix Σ_i

Each data point is sampled from a ***generative process***:

1. Choose component *i* with probability π_i
2. Generate data point from $N(\mathbf{x}|\mu_i, \Sigma_i)$

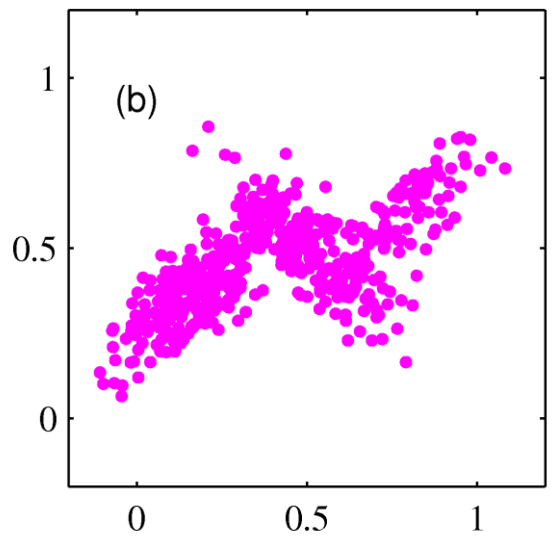


Unsupervised learning for GMM

- In clustering, we don't know the labels Y!
- Maximize marginal likelihood:

$$\prod_j P(\mathbf{x}_j) = \prod_j \sum_i P(y_j = i, \mathbf{x}_j) = \prod_j \sum_i \pi_i N(\mathbf{x}_j | \mu_i, \Sigma_i)$$

- How do we optimize it?
  - No closed form solution



Expectation Maximization (EM)

- Pick K random cluster models (Gaussians)
- Alternate:
  - Assign data instances proportionately to different models
  - Revise each cluster model based on its (proportionately) assigned points
- Stop when no changes

EM: Two Easy Steps

Objective: $\text{argmax}_\theta \prod_j \sum_{i=1}^{k} P(y_j=i,x_j|\theta) = \sum_j \log \sum_{i=1}^{k} P(y_j=i,x_j|\theta)$

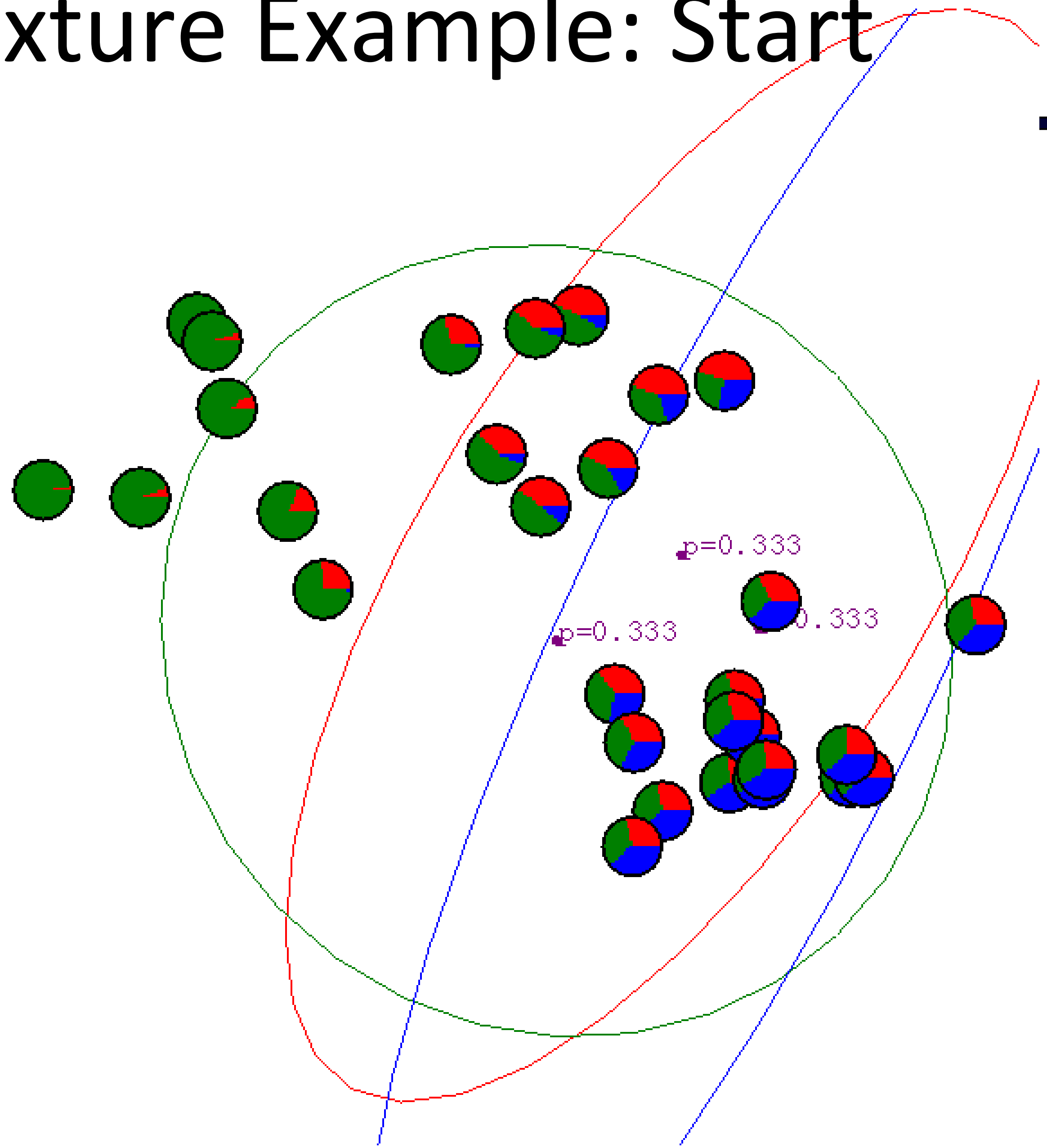
Data: $\{x_j \mid j=1 \,..\, n\}$

Notation a bit inconsistent
Parameters = $\theta=\lambda$

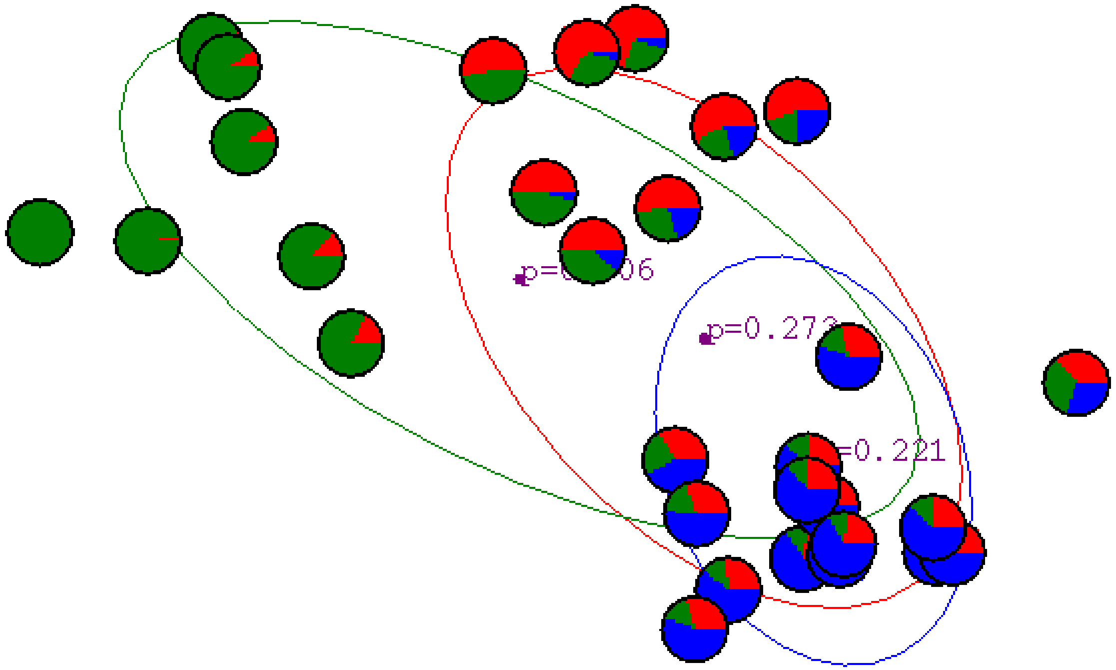
- **E-step**: Compute expectations to "fill in" missing y values according to current parameters, θ
  - For all examples j and values i for y, compute: $P(y_j=i \mid x_j, \theta)$
- **M-step**: Re-estimate the parameters with "weighted" MLE estimates
  - Set $\theta = \text{argmax}_\theta \sum_j \sum_{i=1}^{k} P(y_j=i \mid x_j, \theta) \log P(y_j=i,x_j|\theta)$

Especially useful when the E and M steps have closed form solutions!!!

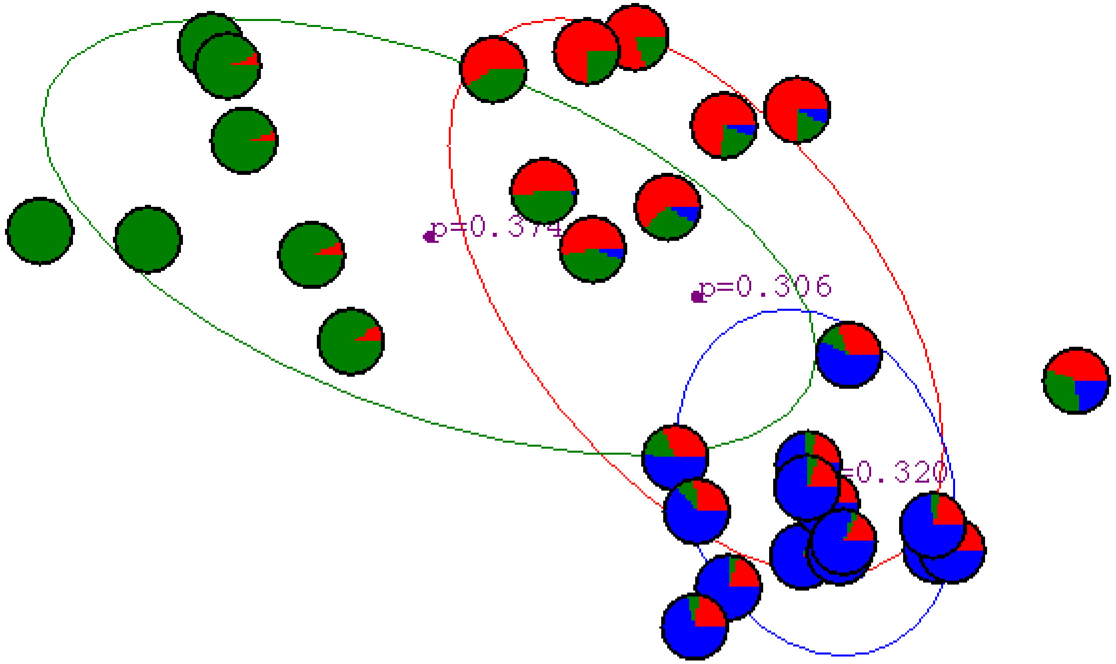
Gaussian Mixture Example: Start



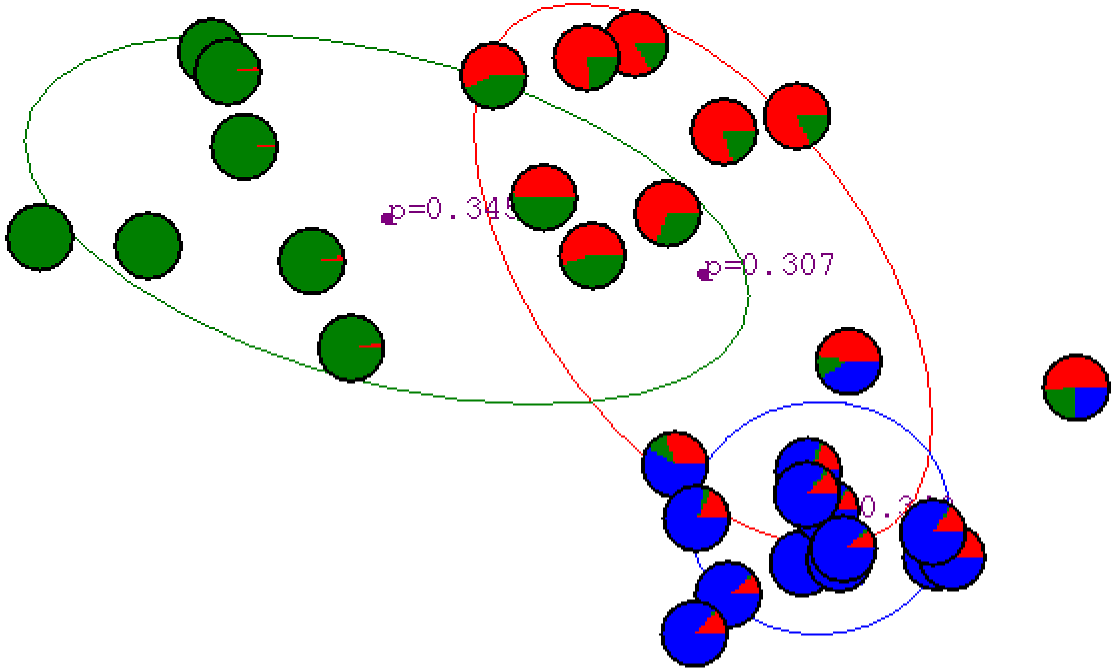
After first iteration



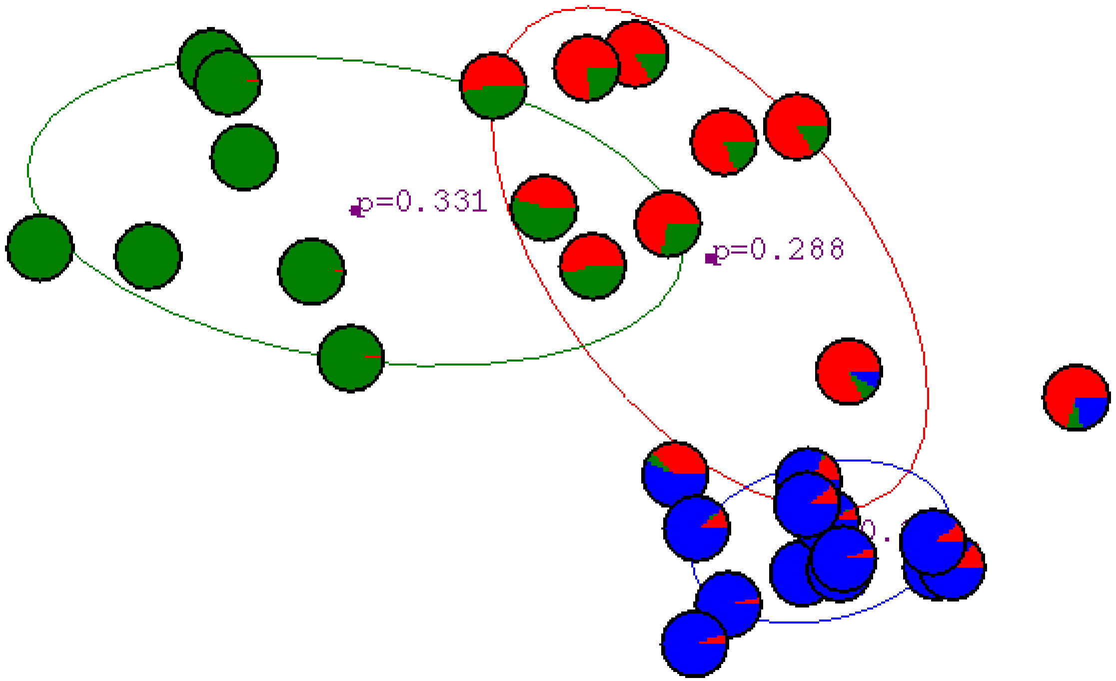
After 2nd iteration



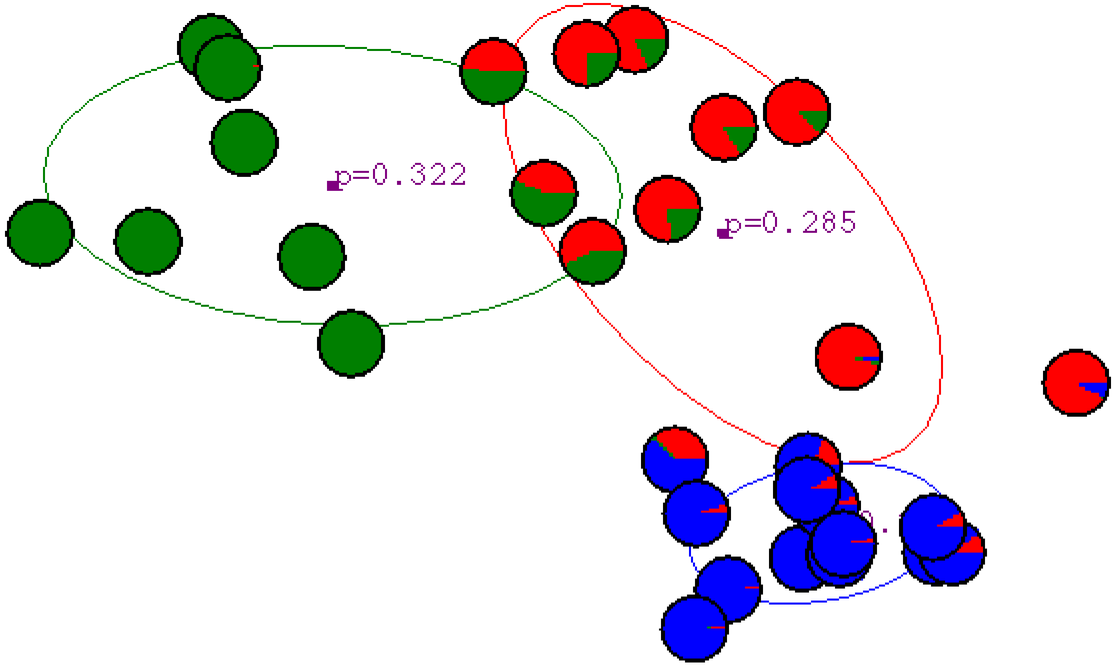
After 3rd iteration



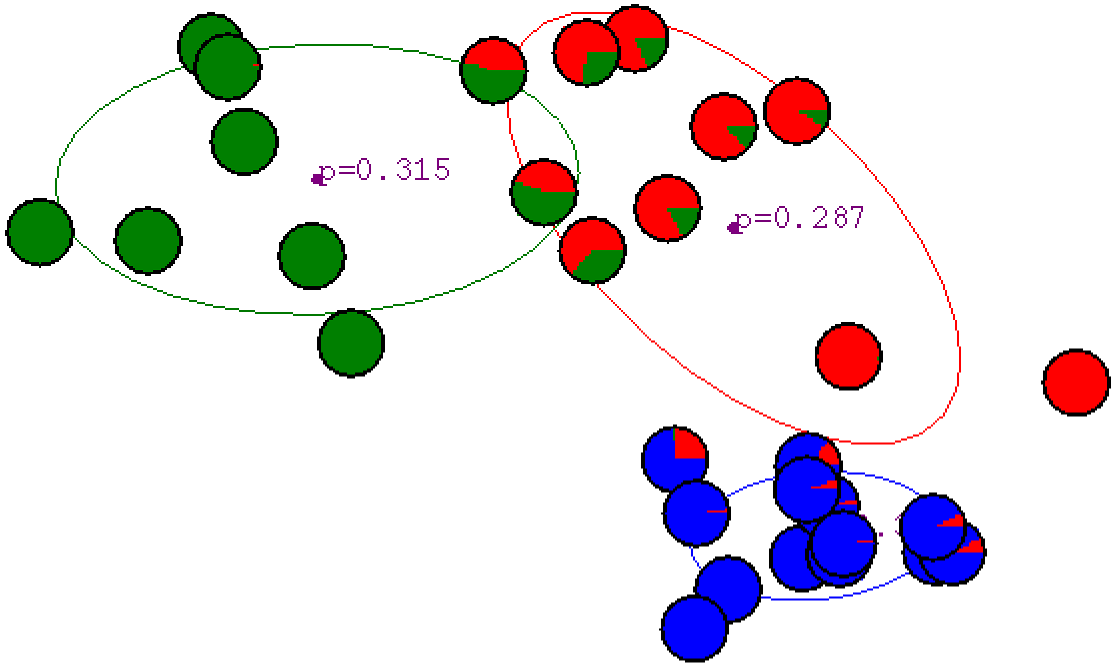
After 4th iteration



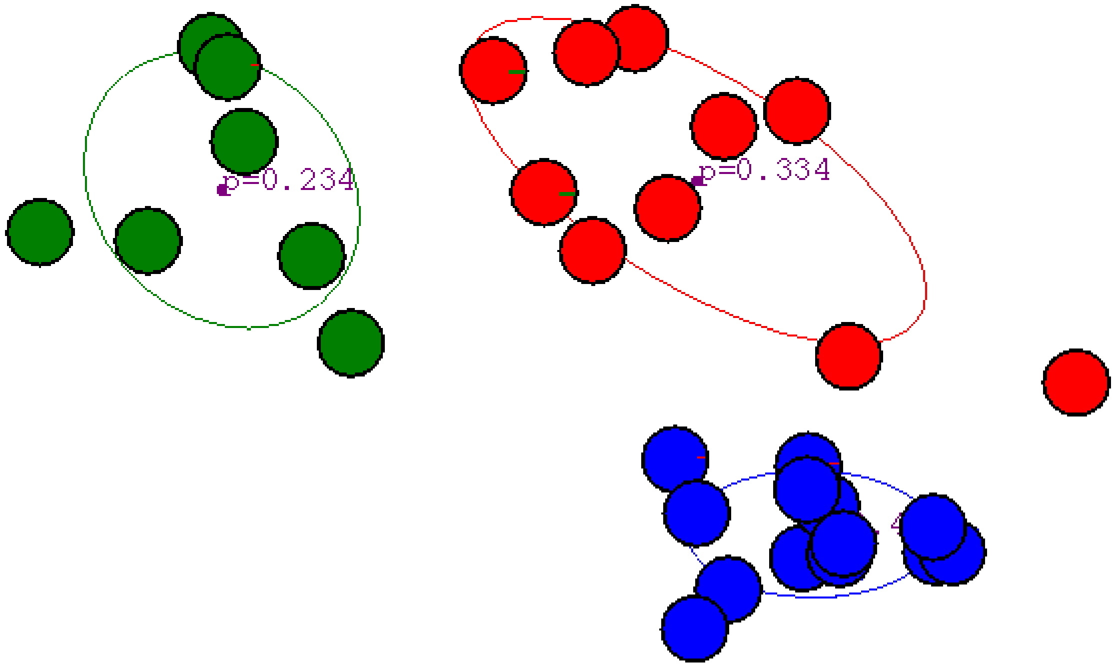
After 5th iteration



After 6th iteration



After 20th iteration



EM and K-means

- EM degrades to k-means if we assume
  - All the Gaussians are spherical and have identical weights and covariances
    - i.e., the only parameters are the means
  - The label distributions computed at E-step are point-estimations
    - i.e., hard-assignments of data points to Gaussians
    - Alternatively, assume the variances are close to zero

EM in General

- Can be used to learn any model with hidden variables (missing data)
- Alternate:
  - Compute distributions over hidden variables based on current parameter values
  - Compute new parameter values to maximize expected log likelihood based on distributions over hidden variables
- Stop when no changes

Summary

- Clustering
  - Group together similar instances
- K-means
  - Assign data instances to closest mean
  - Assign each mean to the average of its assigned points
- EM
  - Assign data instances proportionately to different Gaussian models
  - Revise each model based on its (proportionately) assigned points