Machine Learning and Data Mining

Lecture 10.2: Final Exam Review

CS 434

Final Exam

Points	Section
_____ / 2	Bonus Questions On This Page
_____ / 40	Concept Questions
_____ / 17	kMeans and HAC Clustering
_____ / 17	Overfitting in Decision Trees
_____ / 16	Dimensionality Reduction with PCA
_____ / 10	Simple Neural Networks
_____ / 100	Total

Final Exam

12-13-2023 from 1400 to 1550 in COVL 216 - Covell Hall 216 (GP)

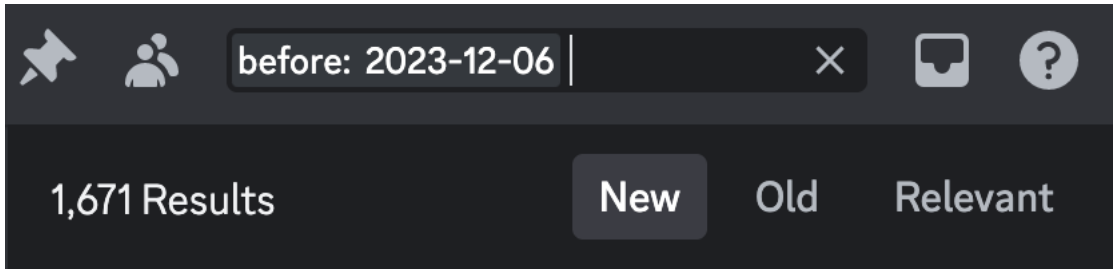
RECAP

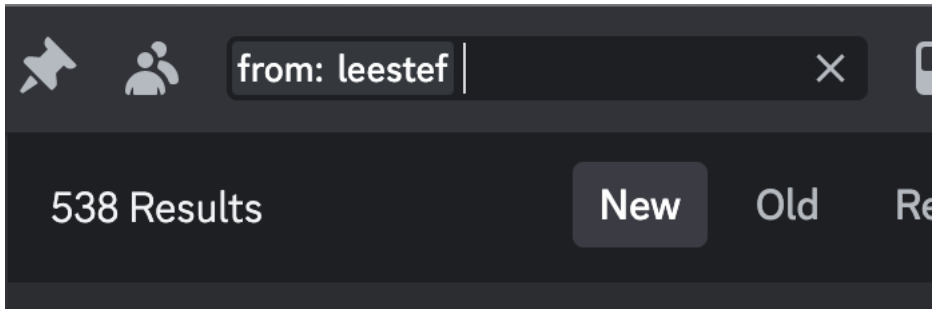
From Last Lecture

CS434 in Review

- Thanks for your feedback in discord / conversation / debriefings. They help me make the course better.

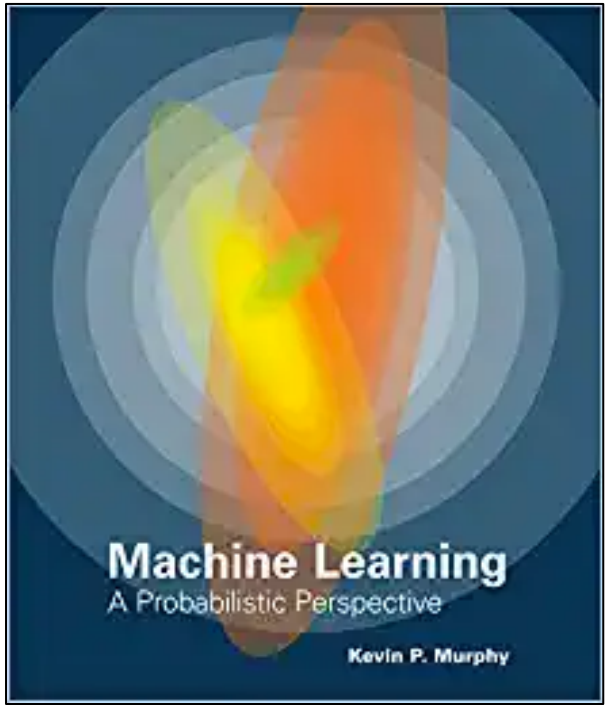
- Thanks for building a community around this course!
  1,671 Discord messages (only public channels counted)

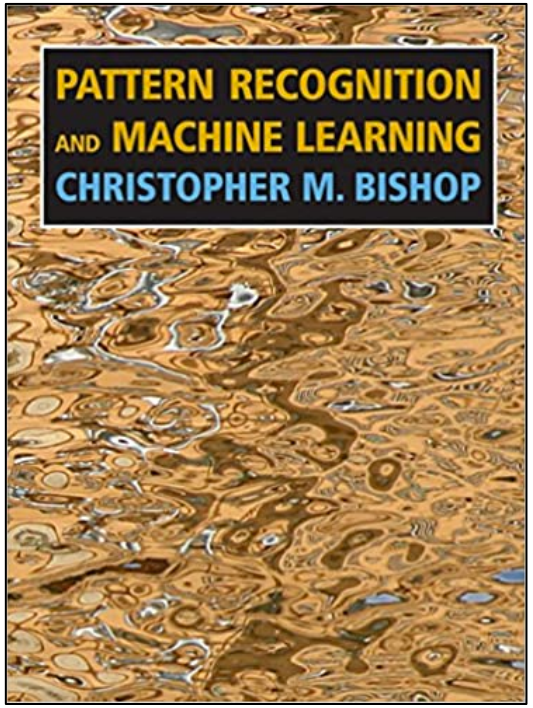


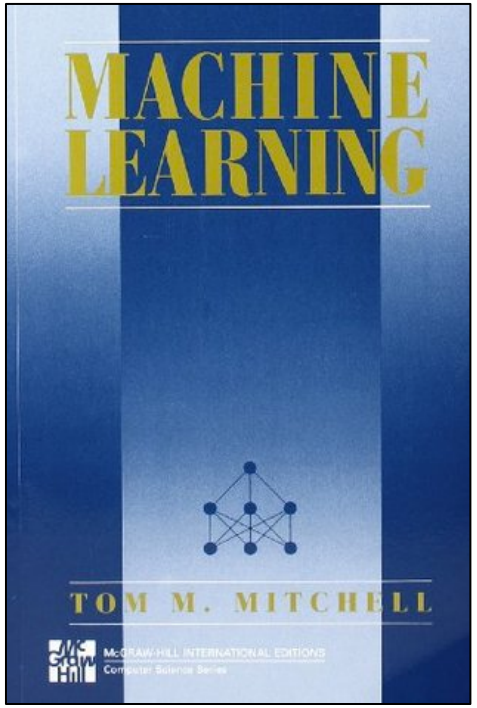


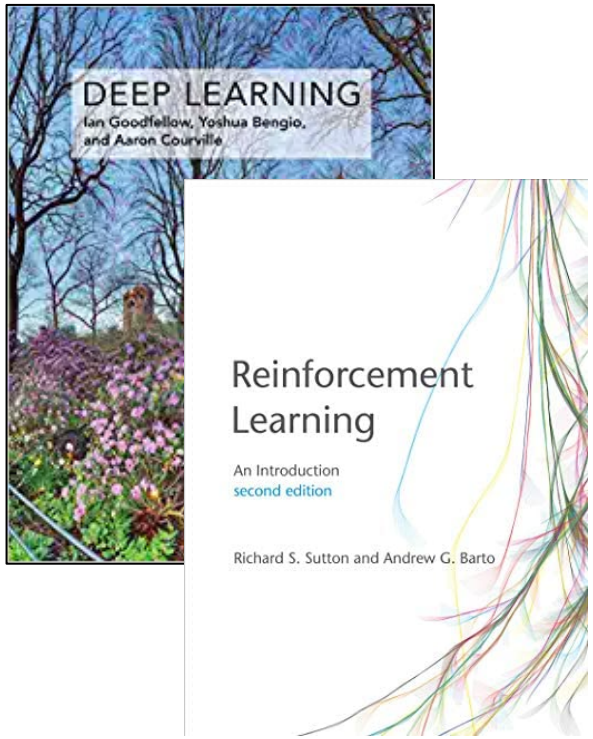
- Thanks for all the effort you've put in. My goal is to make the "things learned vs. time spent" ratio high. I hope you found the effort to be rewarding.

If you want to know more from here, these are good books:









If I've done my job as well as I hope, you should be able to self-study these now.

Topic Area:

Neural Networks

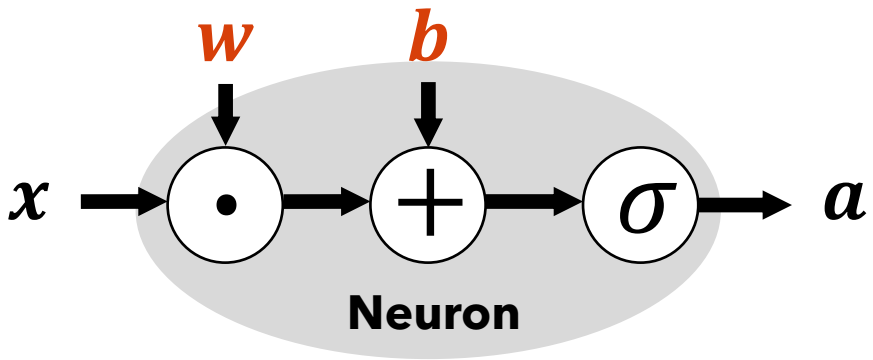
The Humble Neuron Vectorized

$$a = \sigma\left(b + \sum_{i=1}^{d} w_i x_i\right) = \sigma(b + \boldsymbol{w}^T \boldsymbol{x})$$

$$\underset{1\times1}{\boldsymbol{a}} \quad \begin{bmatrix} a \end{bmatrix} = \sigma\left(\begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}^{\boldsymbol{T}} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} + \underset{1\times1}{\boldsymbol{b}} \begin{bmatrix} b \end{bmatrix} \right)$$

$\boldsymbol{w}$ $d\times1$ $\quad$ $\boldsymbol{x}$ $d\times1$

Neuron

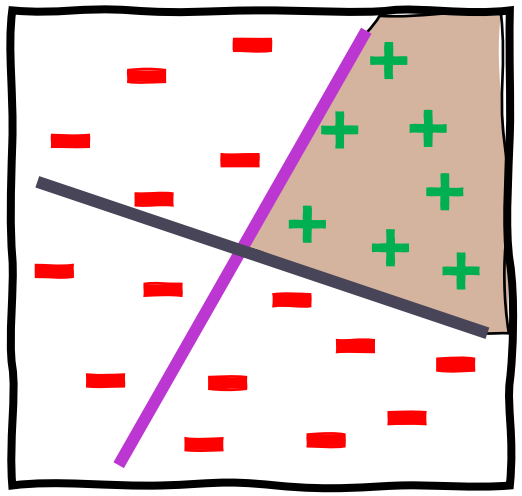


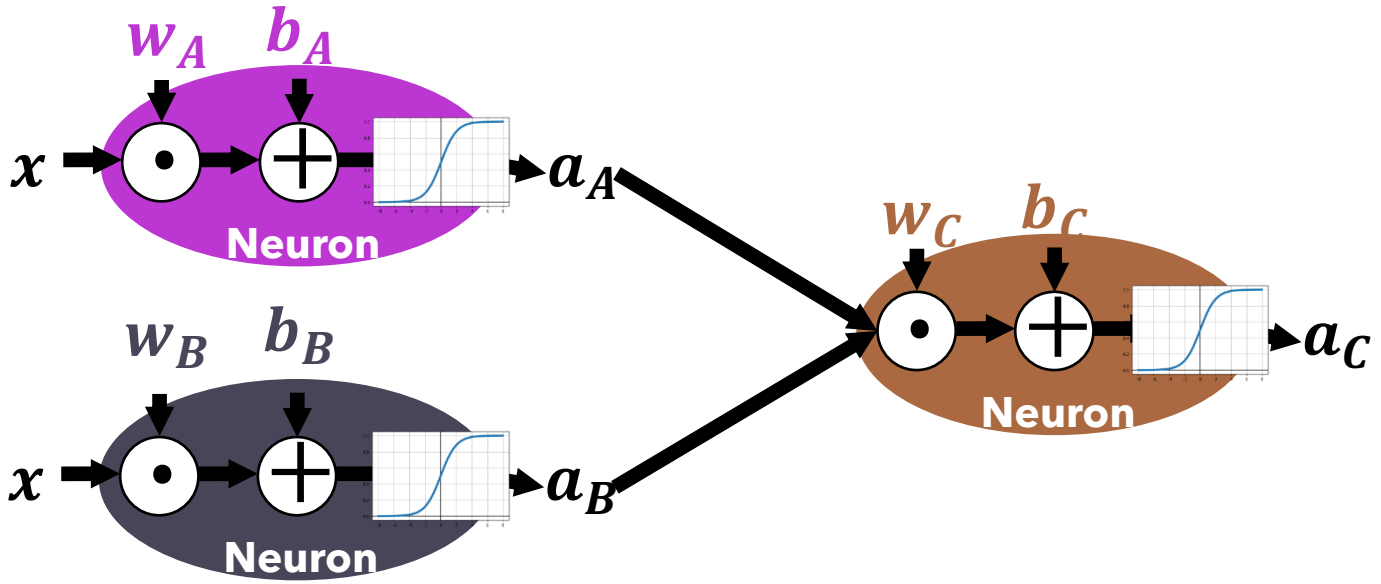
Neuron is a linear function of its input followed by a (typically) non-linear activation function to produce output.

Hyperparameters : activation function (σ)

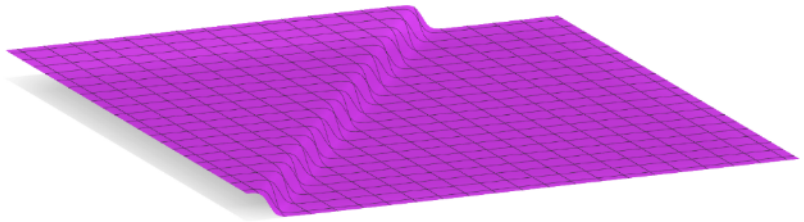
Learnable Parameters: $\mathbf{w} \in \mathbb{R}^d$ and $\mathbf{b} \in \mathbb{R}$

Idea: Stack Them Together to form Neural Nets



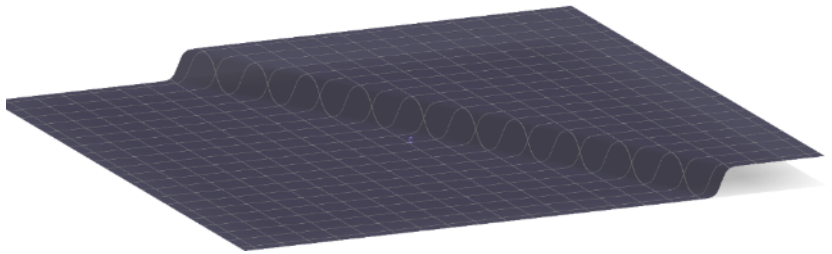


\- and +



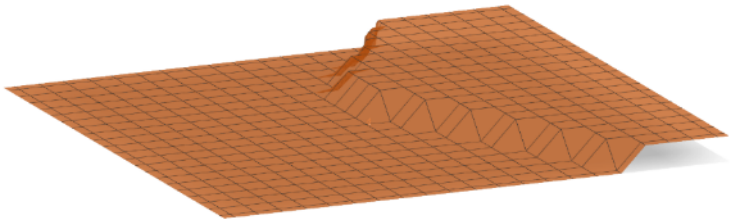
$w = [-7, 5]^T$

$b = -15$



$w = [10, 5]^T$

$b = -15$

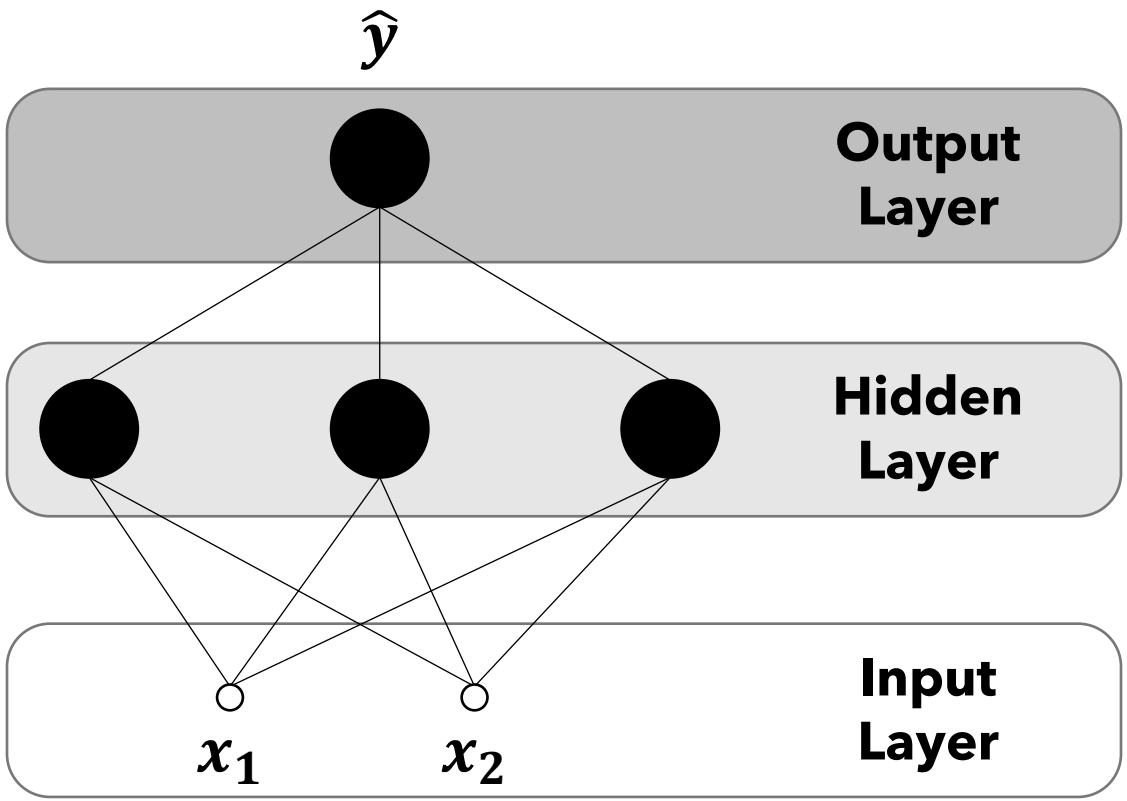


$w = [-100, 100]^T$

$b = -10$

Basic Multilayer Neural Network

A Neural Network is a set of connected neurons. A very typical arrangement is a feed-forward multilayer neural network like the one shown below.



Each layer receives its input from the previous layer and forwards its output to the next – thus the **feed-forward** description.

The layers of neurons between the input and output are referred to as **hidden layers**.

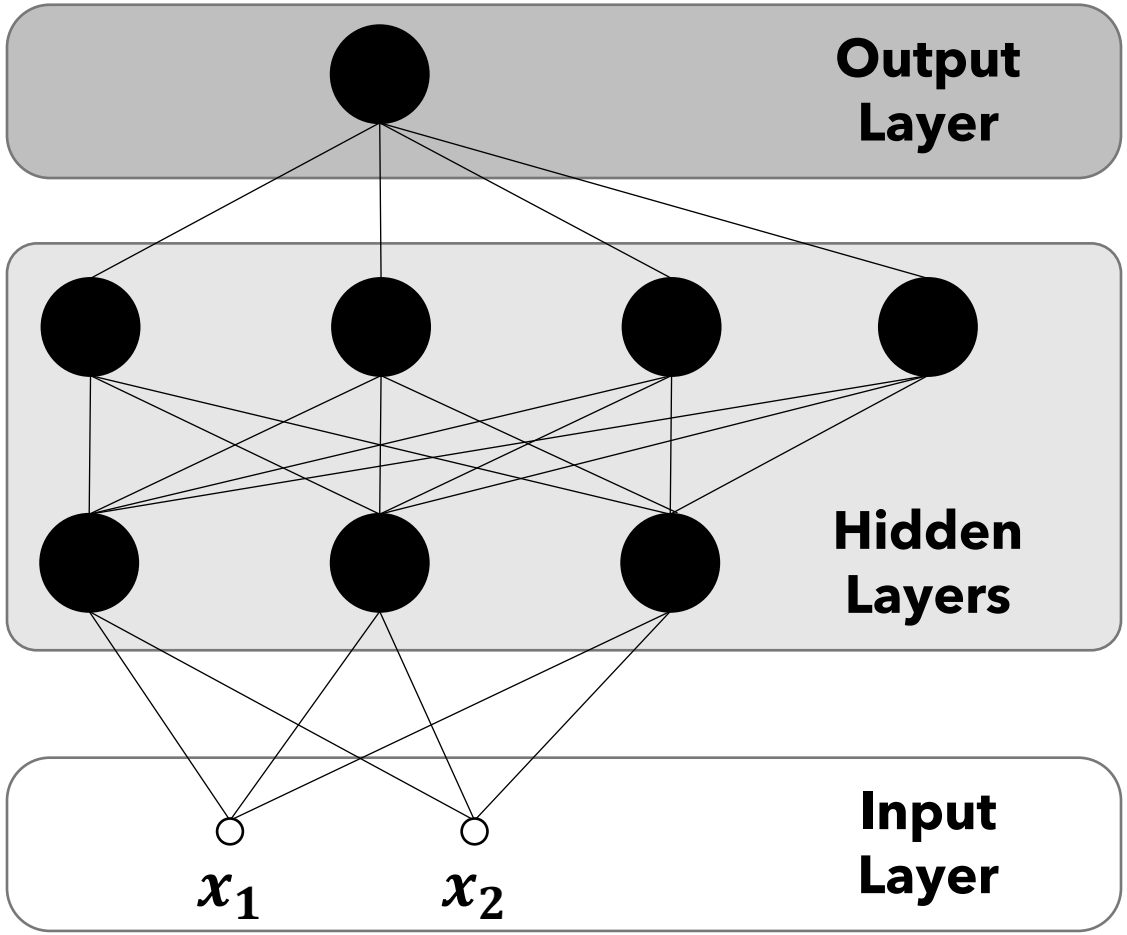
- This network for instance is **a 2-layer neural** network (1 hidden and 1 output)
- Number of neurons in a layer is referred to as its width.

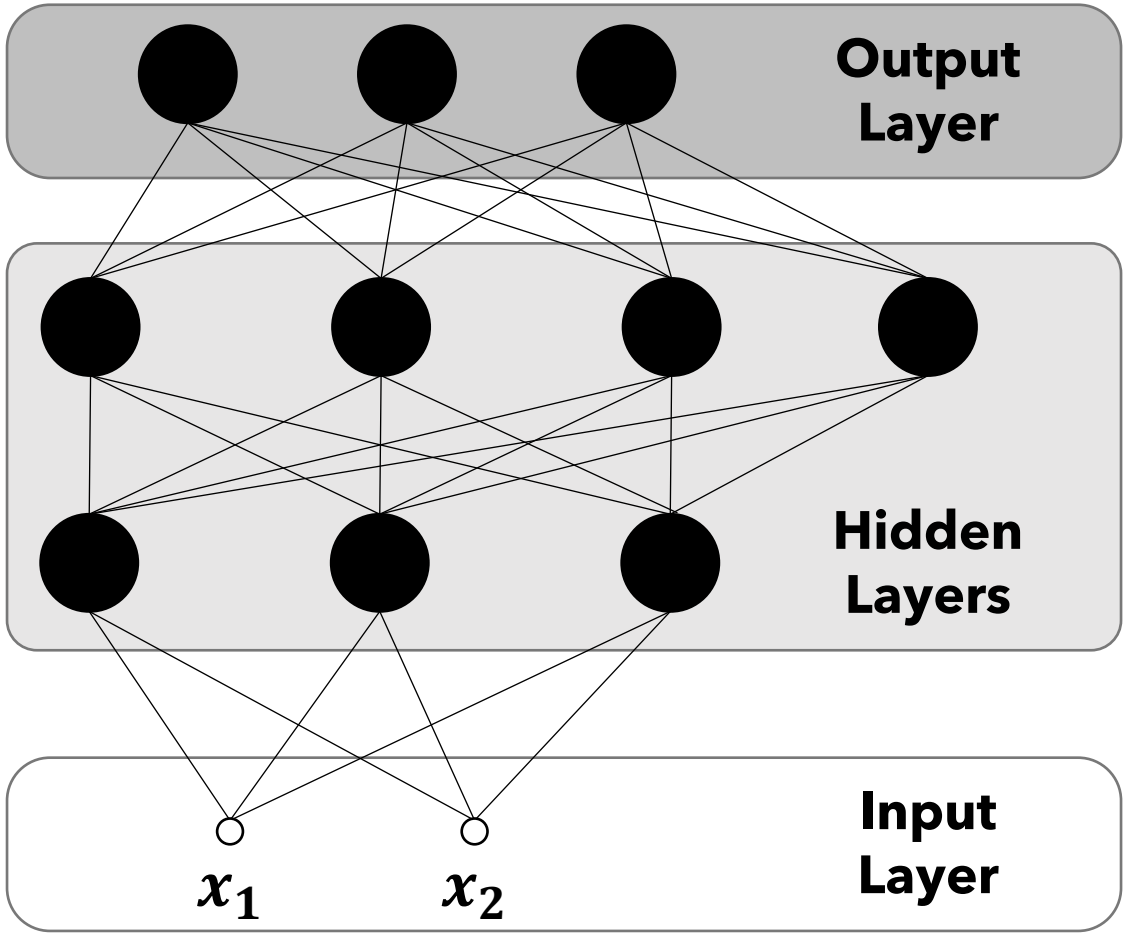
Activation functions in different layers can be heterogenous.

- Output layer's activation is task dependent
  - Linear for regression
  - Sigmoid or softmax for classification

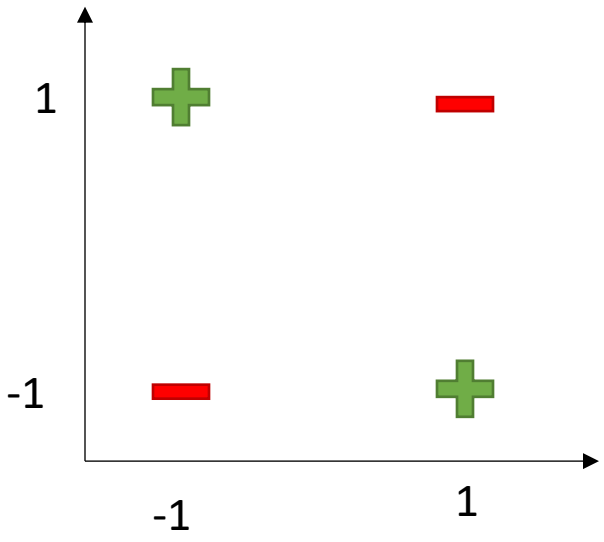
Basic Multilayer Neural Network

Can make arbitrary configurations of # of hidden layers, layer widths, and number of inputs / outputs. Very flexible models!



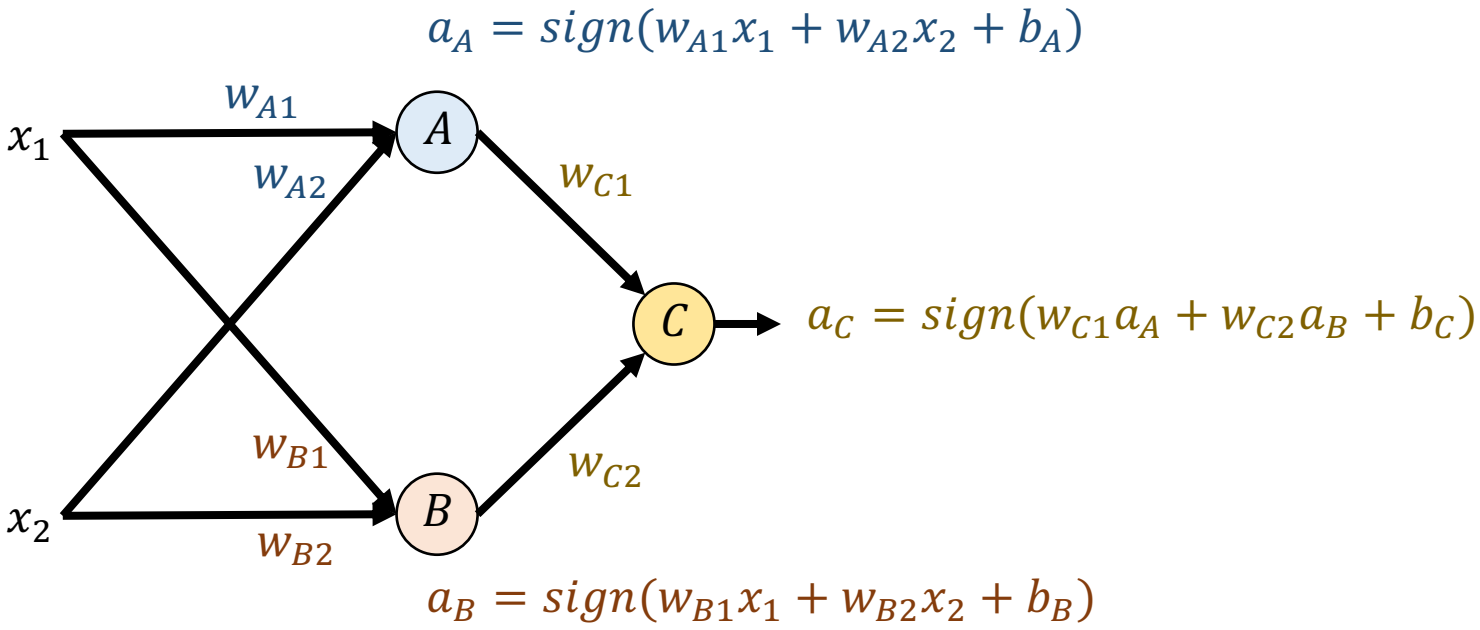


An Example



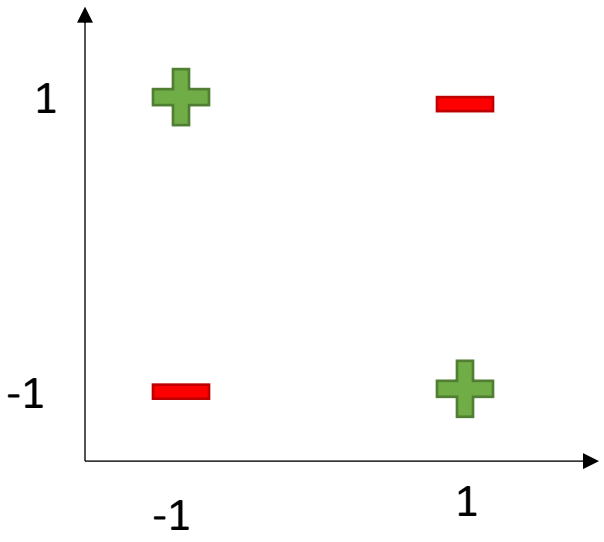
X1	X2	Y
0	0	-1
0	1	1
1	0	1
1	1	-1

Let's consider neurons with a sign activation:



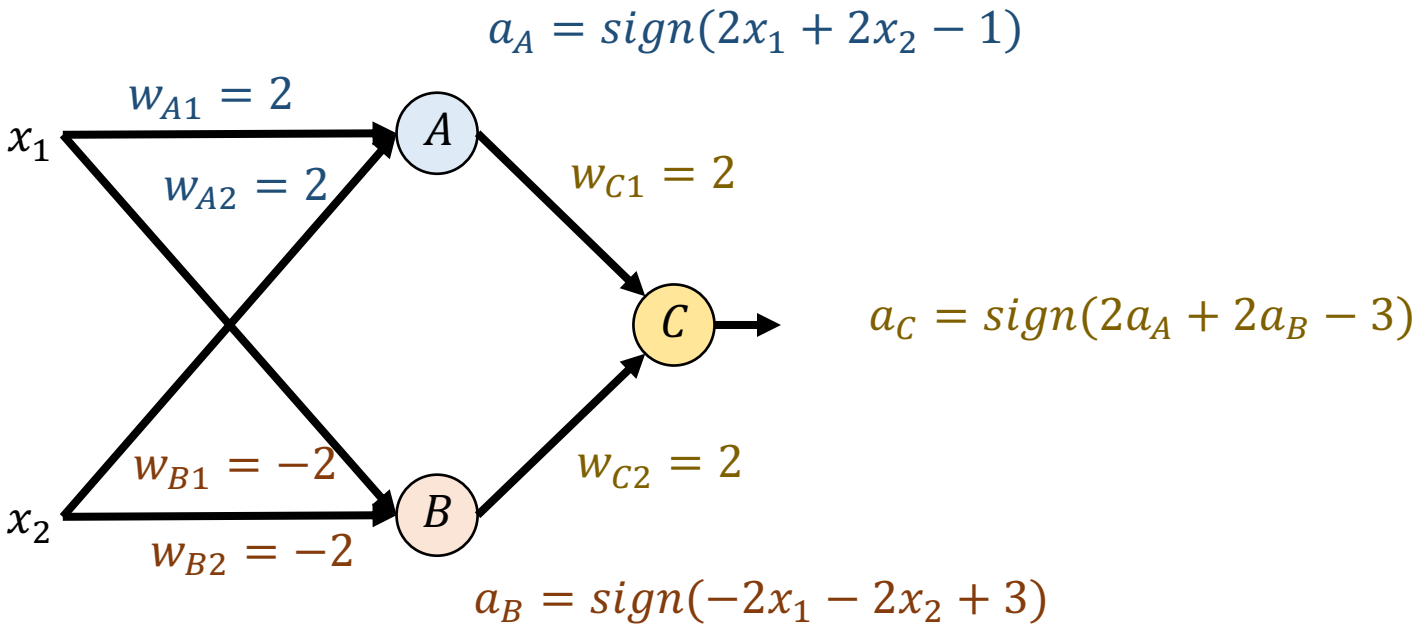
What values can we set the parameters to let the network solve this XOR dataset?

An Example



X1	X2	Y
0	0	-1
0	1	1
1	0	1
1	1	-1

Let's consider neurons with a sign activation:



A operates like an OR.
B operates like a NAND.
C operates like an AND.

Questions Break!

Define a neural network that can achieve zero training error on this dataset. You can choose how many neurons, but you must provide a value for all parameters.
The final output should equal Y for all examples.

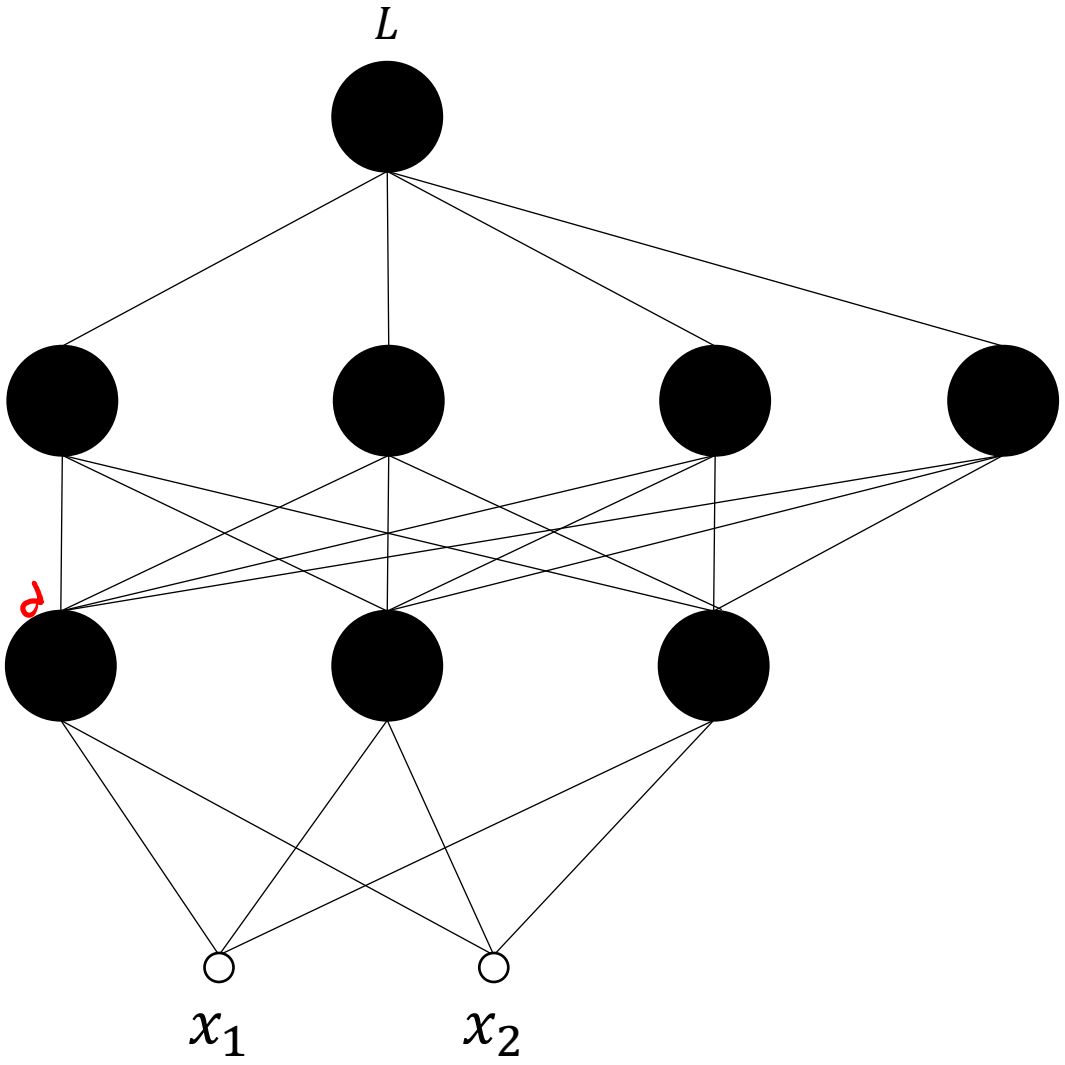
X1	X2	Y
0	0	-1
0	1	1
1	0	-1
1	1	1

Questions Break!

Define a neural network that can achieve zero training error on this dataset. You can choose how many neurons, but you must provide a value for all parameters.
The final output should equal Y for all examples.

(X1 and X3) or (X2 and X3)

X1	X2	X3	Y
0	0	0	-1
0	0	1	-1
0	1	0	-1
0	1	1	1
1	0	0	-1
1	0	1	1
1	1	0	-1
1	1	1	1



Neural Network Training In Words

Forward Pass

1) For each training example:
   - Compute and store all activations
   - Compute loss

Backward Pass

2) Compute gradient of the loss with respect to all network parameters
   - Will do this efficiently with an algorithm called **backpropagation**

Update

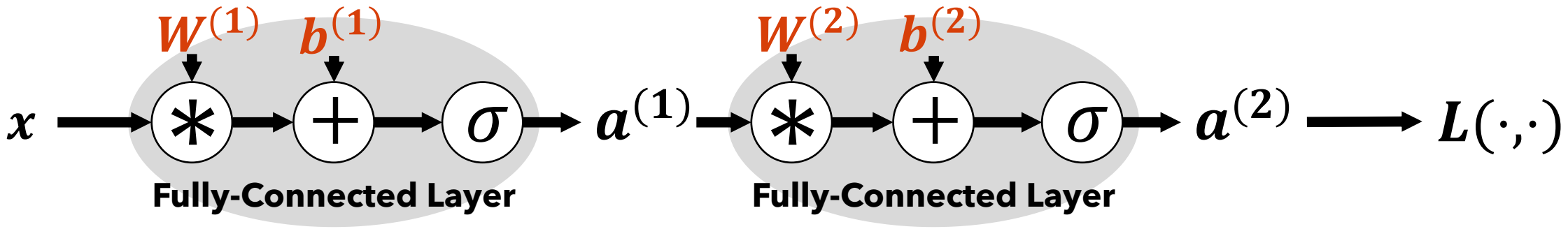
Take a step of gradient descent to minimize the loss

What are Neural Networks good for?

Non-linear Function Learning: As we saw in the demo yesterday, neural networks can learn non-linear decision boundaries. But unlike the kernel methods / basis functions we used before, we don't need to specify the exact from of this non-linear function.

Theoretically Universal Approximators: For any continuous function F over a bounded subspace of D-dimensional space, there exists a two-layer neural network $\hat{F}$ with a finite number of hidden units that approximates F arbitrarily well. That is to say, for all x in the domain of F, $\left|F(x) - \hat{F}(x)\right| < \epsilon$

Incredibly Flexible: Can be used for classification or regression. Or just any problem with a differentiable loss function.



Loss Function *L*– A function measuring "how bad" a network's output is, usually relative to some gold-standard for what the output should be.

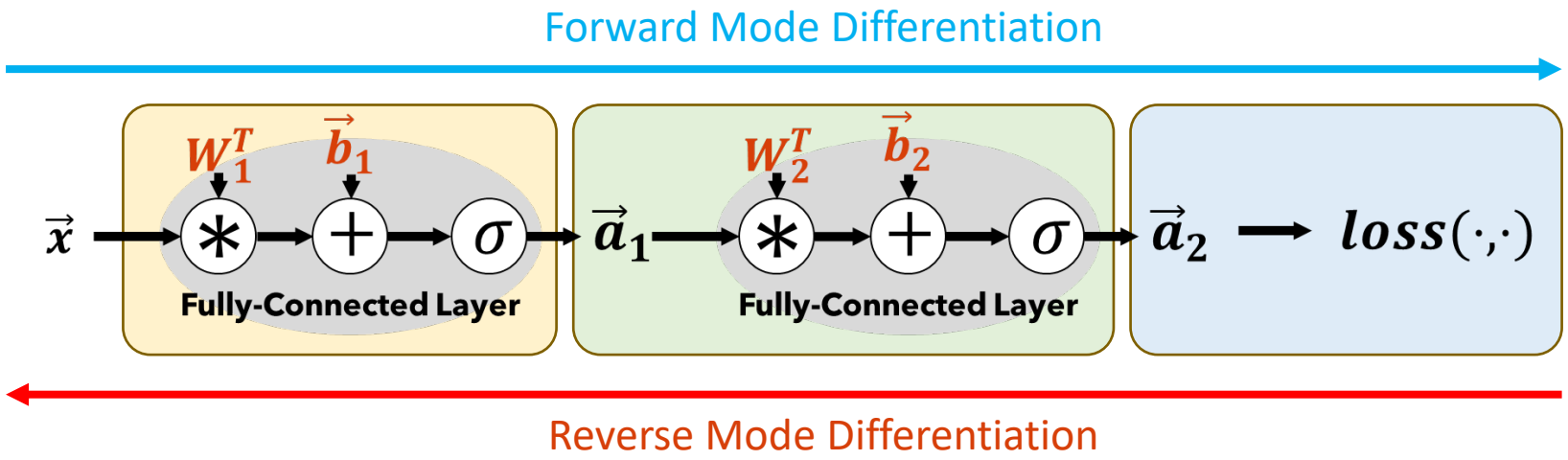
Backpropagation – A reverse-mode automatic differentiation algorithm commonly used to efficiently compute parameter gradients when training neural networks via gradient descent.

Builds off two simple observations / ideas:

1) Neural networks tend to have lower dimensional outputs than inputs.

2) We shouldn't recompute something we already know.

Seed 1) Forward vs. Reverse Mode Differentiation

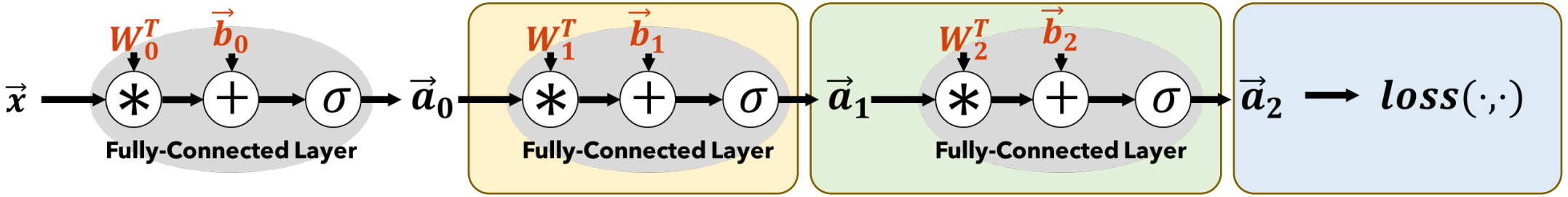


Forward Mode Differentiation
(input to output, dim input << dim output)

$$\frac{\delta loss}{\delta \boldsymbol{W}_1} = \frac{\delta loss}{\delta\, \vec{\boldsymbol{a}}_2} * \frac{\delta \vec{\boldsymbol{a}}_2}{\delta \vec{\boldsymbol{a}}_1} * \frac{\delta \vec{\boldsymbol{a}}_1}{\delta\, \boldsymbol{W}_1}$$

Reverse Mode Differentiation
(output to input, dim output << dim input)

Gradients of Our Parameters / The Seeds of Backprop

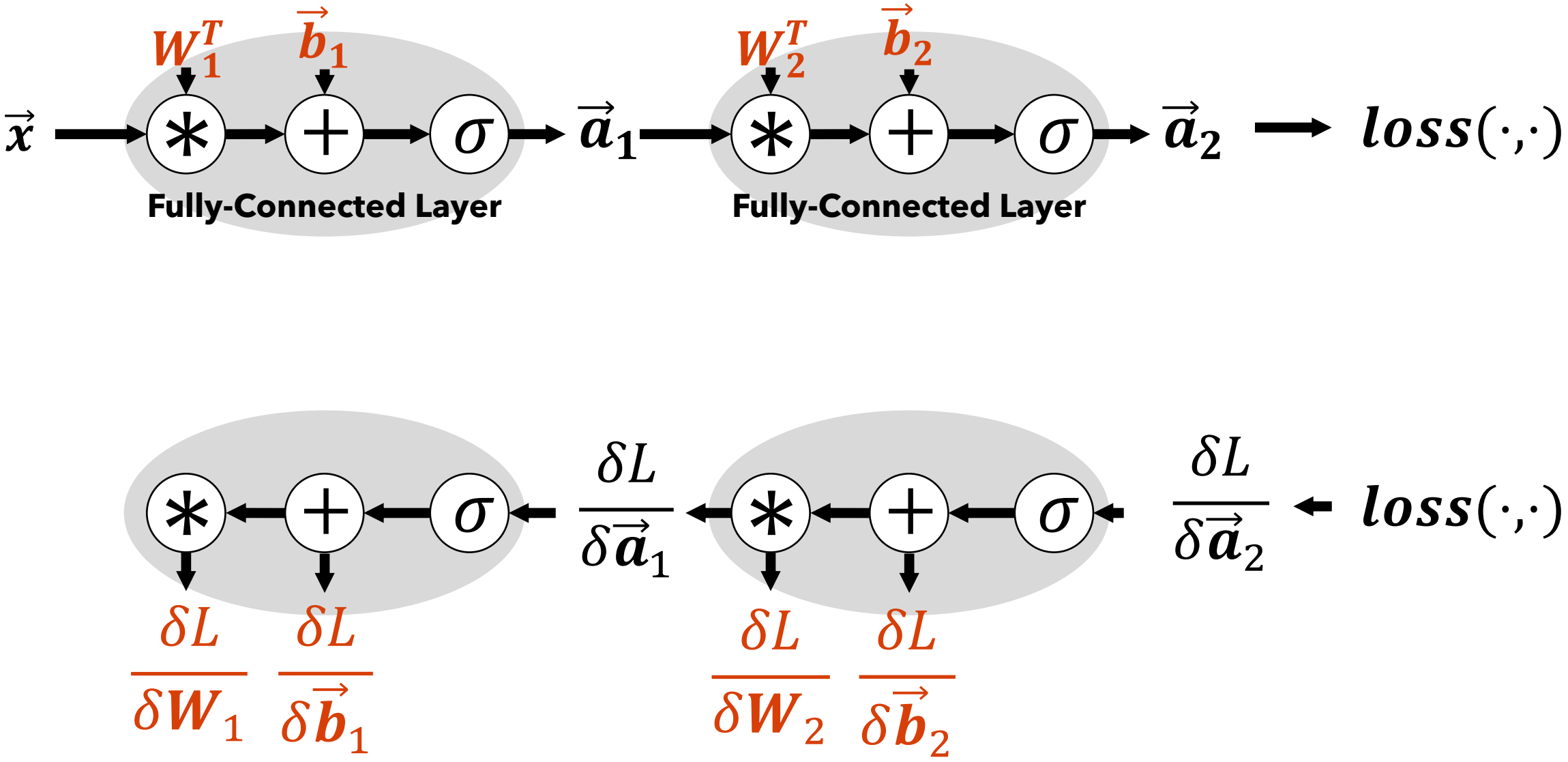


$$\frac{\delta L}{\delta \boldsymbol{W}_2} = \frac{\delta L}{\delta \vec{\boldsymbol{a}}_2} * \frac{\delta \vec{\boldsymbol{a}}_2}{\delta \boldsymbol{W}_2} \qquad \frac{\delta L}{\delta \vec{\boldsymbol{b}}_2} = \frac{\delta L}{\delta \vec{\boldsymbol{a}}_2} * \frac{\delta \vec{\boldsymbol{a}}_2}{\delta \vec{\boldsymbol{b}}_2}$$

$$\frac{\delta L}{\delta \boldsymbol{W}_1} = \frac{\delta L}{\delta \vec{\boldsymbol{a}}_2} * \frac{\delta \vec{\boldsymbol{a}}_2}{\delta \vec{\boldsymbol{a}}_1} * \frac{\delta \vec{\boldsymbol{a}}_1}{\delta \boldsymbol{W}_1} \qquad \frac{\delta L}{\delta \vec{\boldsymbol{b}}_1} = \frac{\delta L}{\delta \vec{\boldsymbol{a}}_2} * \frac{\delta \vec{\boldsymbol{a}}_2}{\delta \vec{\boldsymbol{a}}_1} * \frac{\delta \vec{\boldsymbol{a}}_1}{\delta \vec{\boldsymbol{b}}_1}$$

$$\frac{\delta L}{\delta \boldsymbol{W}_0} = \frac{\delta L}{\delta \vec{\boldsymbol{a}}_2} * \frac{\delta \vec{\boldsymbol{a}}_2}{\delta \vec{\boldsymbol{a}}_1} * \frac{\delta \vec{\boldsymbol{a}}_1}{\delta \vec{\boldsymbol{a}}_0} * \frac{\delta \vec{\boldsymbol{a}}_1}{\delta \boldsymbol{W}_0} \qquad \frac{\delta L}{\delta \vec{\boldsymbol{b}}_0} = \frac{\delta L}{\delta \vec{\boldsymbol{a}}_2} * \frac{\delta \vec{\boldsymbol{a}}_2}{\delta \vec{\boldsymbol{a}}_1} * \frac{\delta \vec{\boldsymbol{a}}_1}{\delta \vec{\boldsymbol{a}}_0} * \frac{\delta \vec{\boldsymbol{a}}_0}{\delta \vec{\boldsymbol{b}}_0}$$

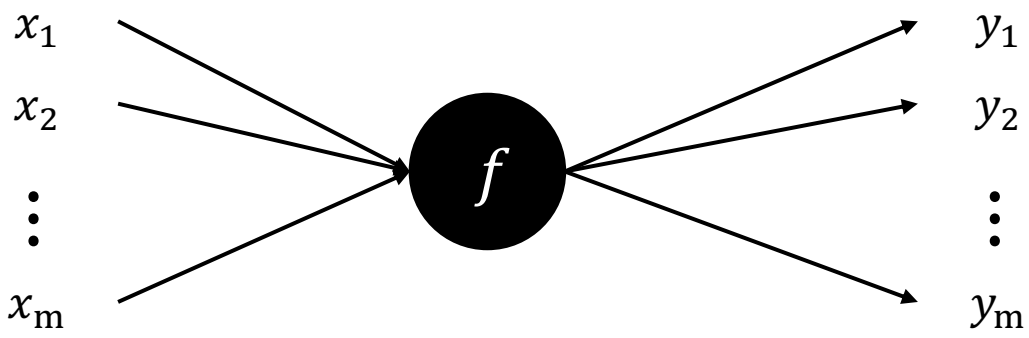
Backpropagation Algorithm



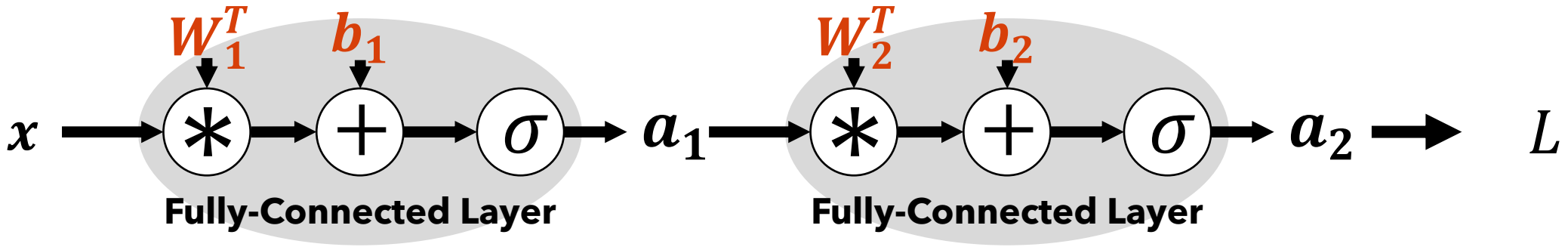
Computational Graph – A directed acyclic graph (DAG) with vertices corresponding to computation and edges to intermediate results of the computation.

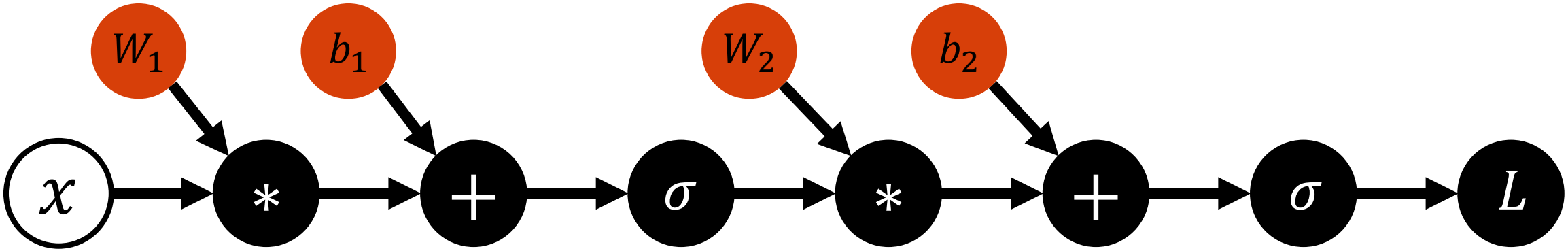
For backprop to work, each node needs to define:

- Its **forward** computation $y_1, \ldots, y_k = f(x_1, \ldots, x_m)$
- Its **backward** computation: $\frac{\delta y_i}{\delta x_j} \; \forall i, j$

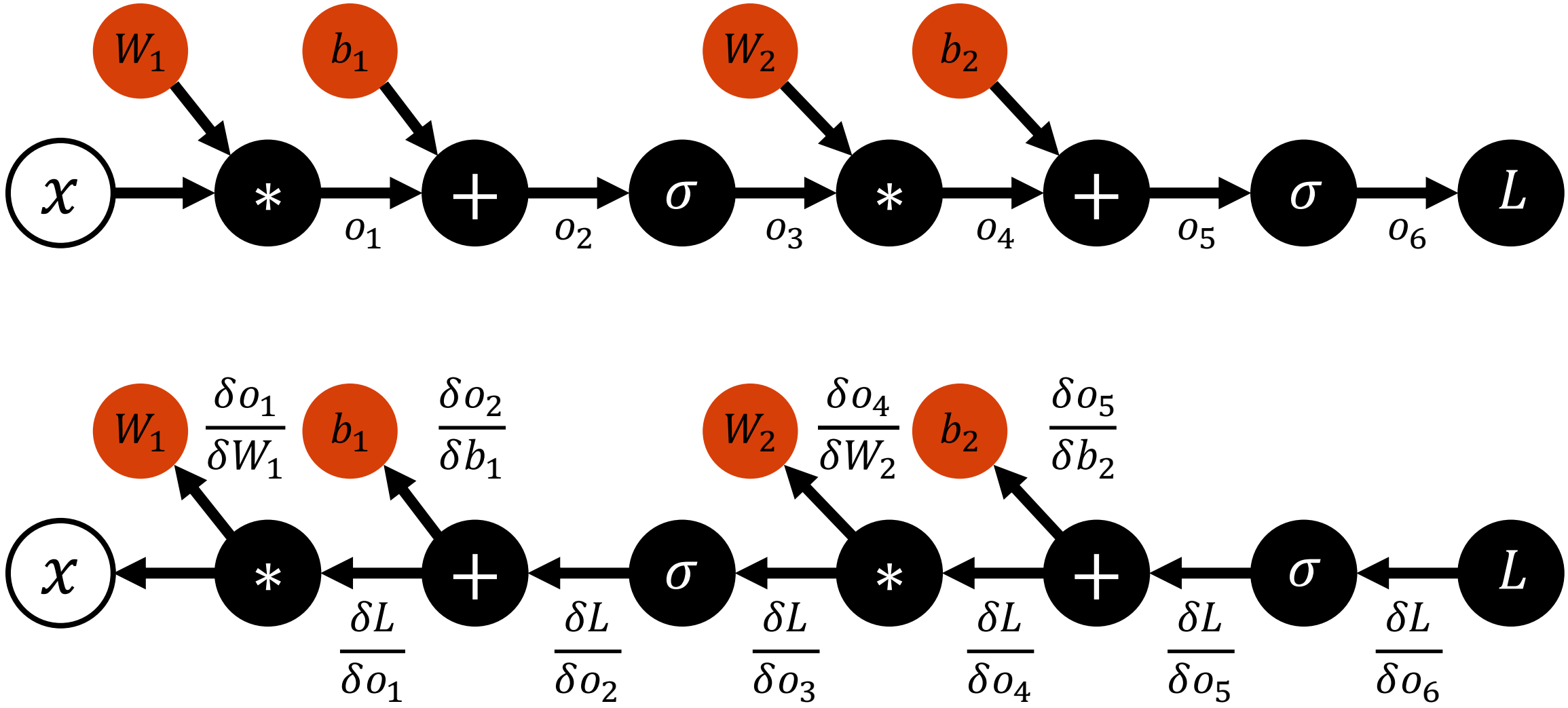


Neural Networks as Computational Graphs



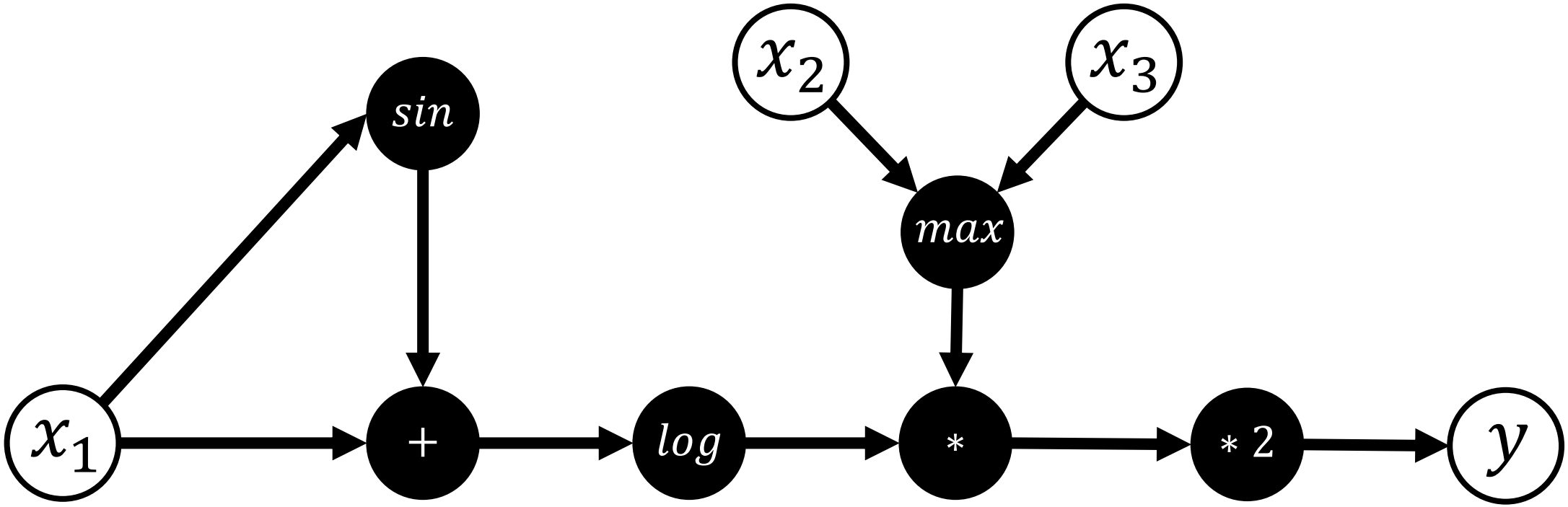


Neural Networks as Computational Graphs



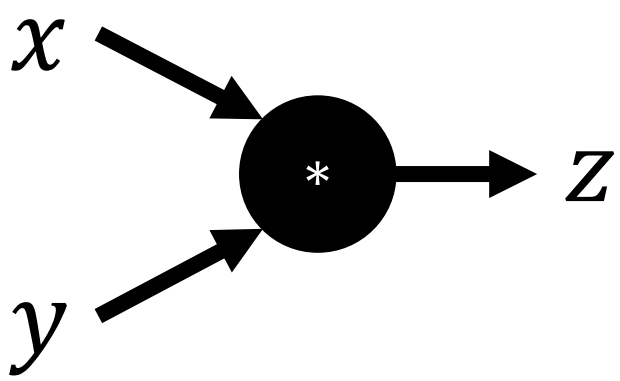
Backpropagation on Computational Graphs

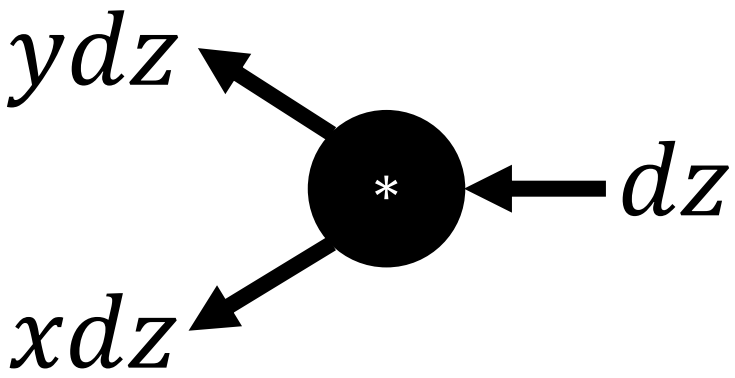
$$y = 2\log(x_1 + \sin(x_1))\max(x_2, x_3)$$



Adapted from Fei-Fei Li, Justin Johnson, Serena Yeung CS 231n

Modularized Implementation: Forward / Backward API






```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Modern Autograd Frameworks Make This Easy

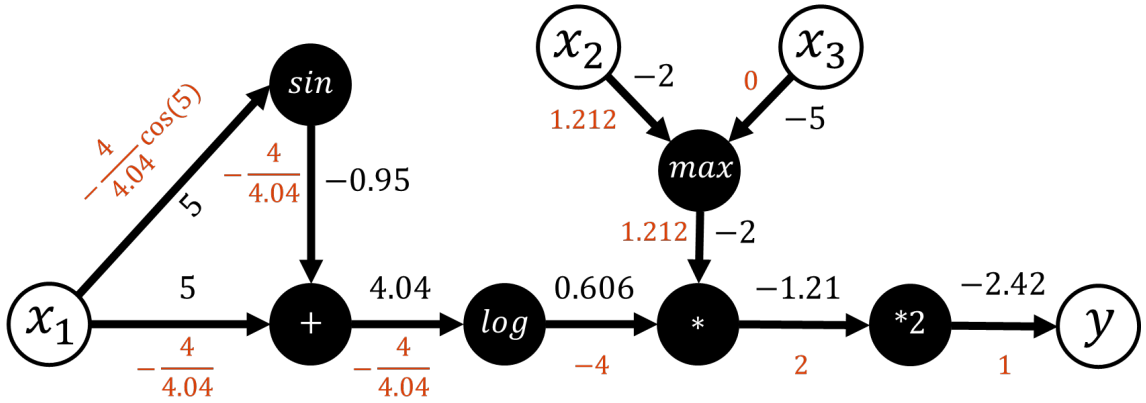
Implement special variable types (e.g., Tensor in PyTorch) that:

- Override most operations with corresponding forward/backward APIs
- Dynamically build a computation graph as the variables are manipulated

Implement engine that can run backpropagation over the computation graph

Modern Autograd Frameworks -- PyTorch

$$y = 2\log(x_1 + \sin(x_1))\max(x_2, x_3)$$



$$\frac{\delta_y}{\delta x_1} = -\frac{4}{4.04}\cos(5) - \frac{4}{4.04} \qquad \frac{\delta_y}{\delta x_2} = 1.213 \qquad \frac{\delta_y}{\delta x_3} = 0$$

```
import torch
import numpy as np

x = torch.tensor([5.0, -2.0, -5.0], requires_grad=True)
y = 2*torch.log( x[0] + torch.sin(x[0]))/np.log(10)*torch.max( x[1:])

y.backward()
print(x.grad)
```

```
tensor([-0.5518,  1.2130,  0.0000])
```

Questions Someone Might Ask You About Neural Networks

Neurons: What is a neuron? What can a single neuron learn? What are activation functions? What are popular activation functions? Are some activations better than others? How does a neuron with a sigmoid activation relate to logistic regression? How does a neuron with the sign function as activation relate to a perceptron?

Neural Networks: What is a multi-layer neural network? What can it learn? What hyperparameters are there?

Loss Functions: What is a loss function? What loss function should I use for regression? What loss function should I use for classification?

Training Neural Networks: What is backpropagation? Why is it an efficient way to compute gradients? What is stochastic gradient descent? How does training behavior change with different settings of learning rate? What is a computational graph?

Advanced Neural Networks: What assumptions are made in convolutional neural networks? What type of computation do recurrent neural networks model?

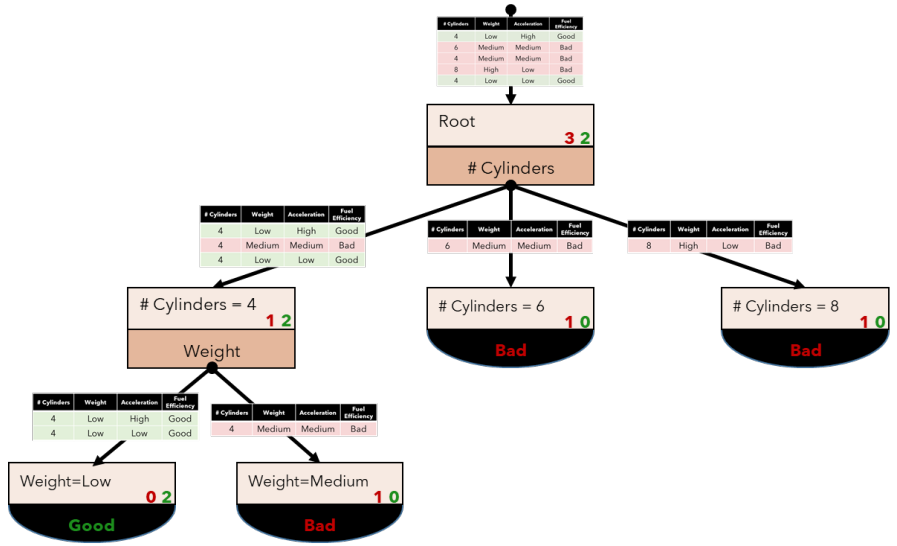
Topic Area:

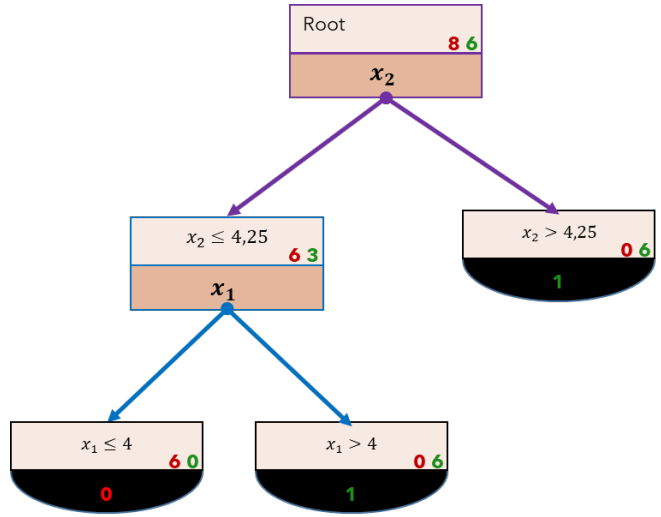
Decision Trees

Decision Trees

A decision tree is a tree structured model where:

- **Internal nodes** perform tests against an individual input feature value
- Each branch from an internal node represents a potential value or range of values of the tested attribute.
- Each **leaf node** predicts an output
  - Either class or continuous value





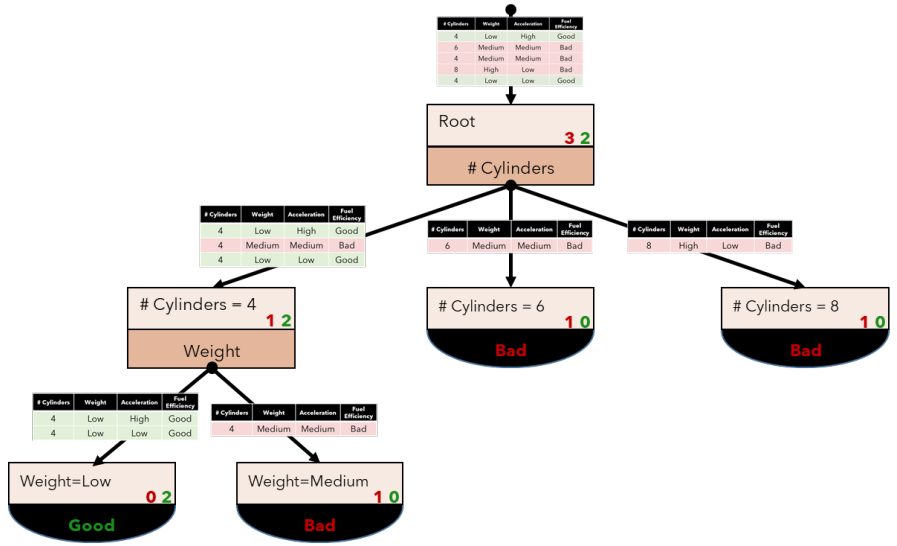
Decision Trees

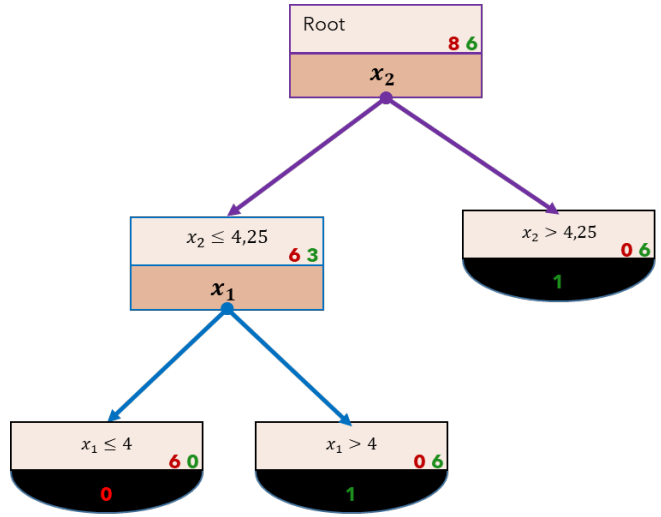
Decision trees have many appealing properties and are work-horses in many practical ML applications.

- Resulting models are human-interpretable (more or less)

- Deals with mixed discrete / continuous inputs without needing careful input encoding
  - Recall our binary encoding for categorical inputs in kNN for HW1

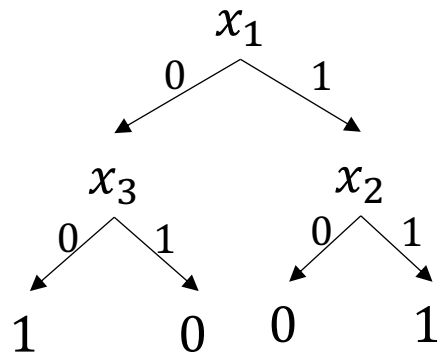
- Can represent arbitrarily complex functions as depth increases

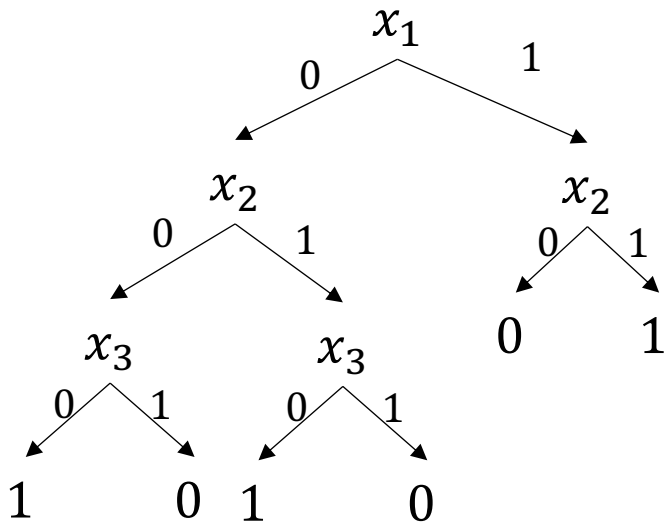




A Few Notes

- Not all features/attributes need to appear in the tree
- A feature/attribute may appear in multiple branches
- On any given path, repeating discrete features is not useful
- Many trees represent the same concept / logical equation but not all trees will have the same size.





Example from Carlos Geustrin

How do we learn decision trees from data?

Easy Mode: find a decision tree that achieves minimum error on training data. Trivially achievable with large enough tree

- Imagine a tree deep enough that each leaf contains only one example (or all examples sharing identical feature values)

NP-Hard Mode: find the smallest decision tree that achieves the minimum training error

- Why might we care about "smallest"?
  Small trees represent *simpler* functions → less likely to overfit.

Most popular methods build the tree greedily – can't guarantee finding smallest.

Greedy Decision Tree Algorithm

Back to our recursive learning algorithm:

BasicGreedyAlgorithm(dataset S):

If S all have same label or all same features:

return Leaf(majorityLabel(S))

Based on S, choose "best" test t to split the data:

Evaluate the information gain of possible splits and pick the largest

Split S into subset $S_1, \ldots, S_k$ for each unique outcome of t applied to S

For i in {1,…,k}:

Create child node c_i = BasicGreedyAlgorithm(S_i)

Entropy of a Random Variable Y: More uncertainty, more entropy!

Discrete Distributions:

$$H(Y) = -\sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

Continuous Distributions:

$$H(Y) = -\int_{y \in Y} P(Y = y) \log_2 P(Y = y)\, dy$$

Conditional Entropy of a Random Variable Y Given X:

Entropy of Y when only considering records were $X = x_j$

$$H(Y|X) = -\sum_{j}^{v} P(X = x_j) \sum_{i=1}^{k} P\left(Y = y_i | X = x_j\right) \log_2 P(Y = y_i | X = x_j)$$

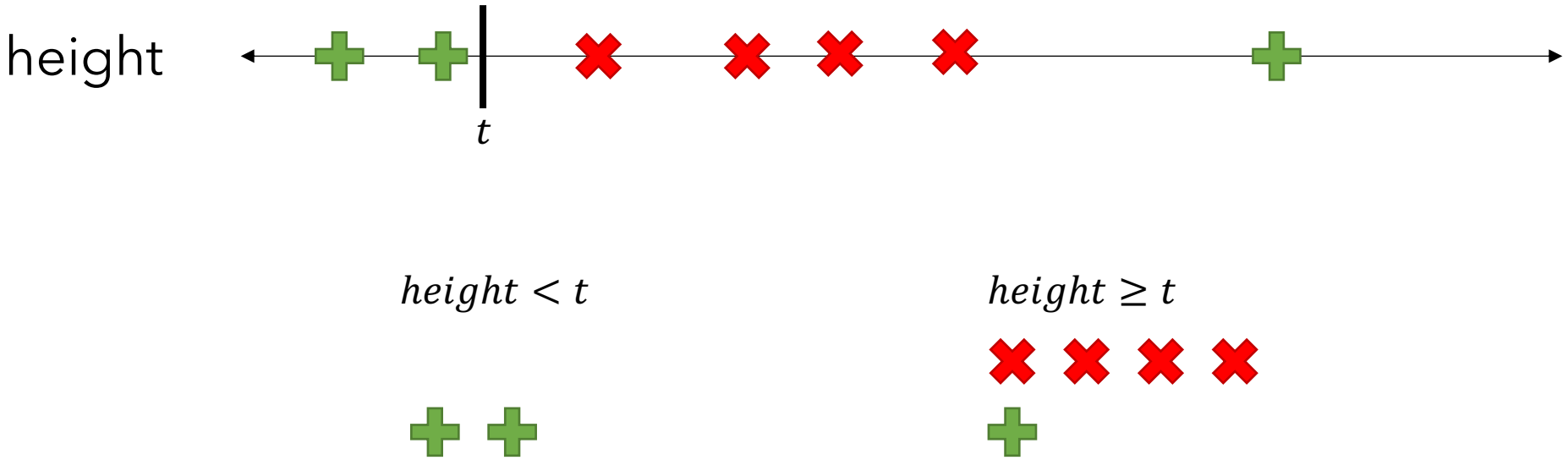
Weighted by probability of $X = x_j$

Continuous distribution version replaces sums with integrals as is typical.

Dealing with Continuous Features

Want to make splits of the form $X_i \geq t$ for some threshold t:

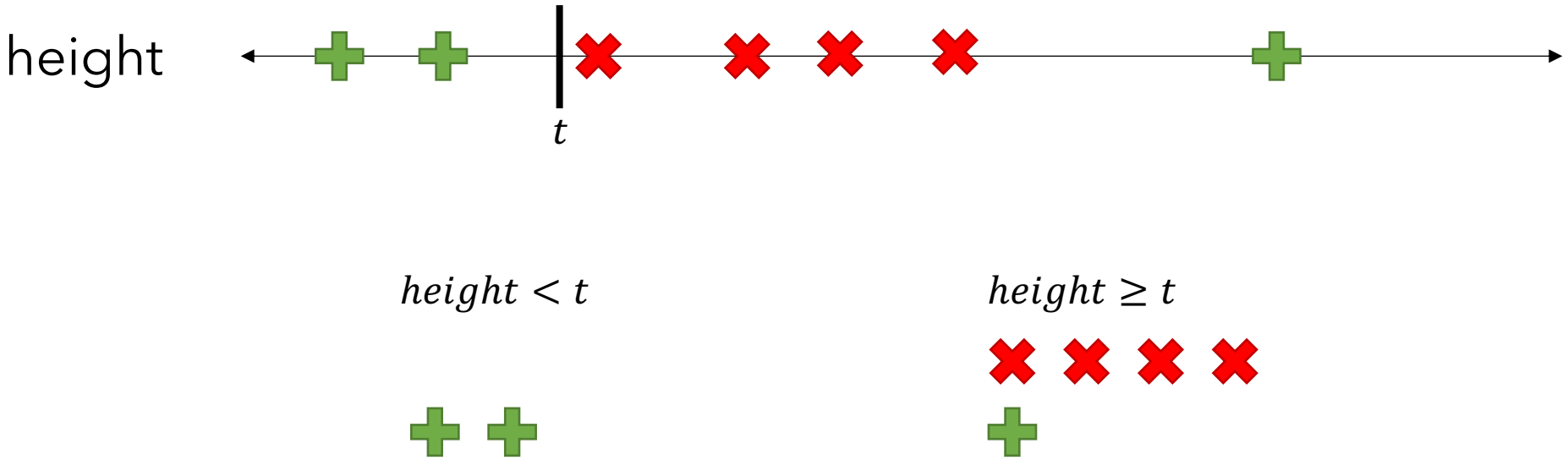
- There are infinitely many possible such splits, but the information gain only changes when a threshold crosses at datapoint



Dealing with Continuous Features

Want to make splits of the form $X_i \geq t$ for some threshold t:

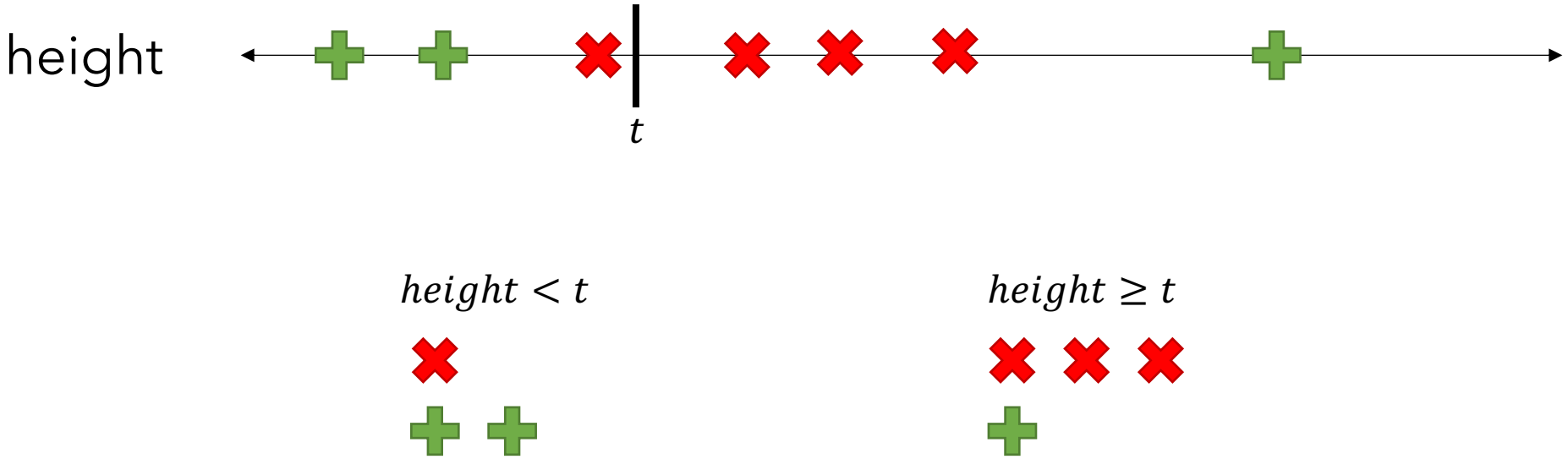
- There are infinitely many possible such splits, but the information gain only changes when a threshold crosses at datapoint



Dealing with Continuous Features

Want to make splits of the form $X_i \geq t$ for some threshold t:

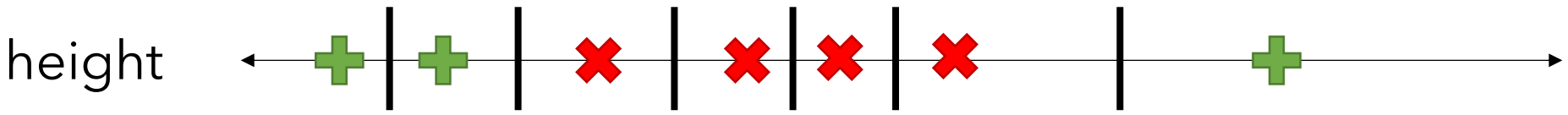
- There are infinitely many possible such splits, but the information gain only changes when a threshold crosses at datapoint



Dealing with Continuous Features

Want to make splits of the form $X_i \geq t$ for some threshold t:

- Sort the values of X_i in the dataset, consider thresholds that occur in between consecutive datapoints.
- Compute information gain for each and choose the max.



Problem of Overfitting

Really large trees can achieve minimal error on **training sets**, but may not generalize to **test data** – overfitting.

Avoiding Overfitting

Option 1: Add more hyperpameters to control tree size:

- Limit depth / limit number of nodes / only split if at least K datapoints present

Option 2: Early stopping based on validation performance

- Monitor the validation accuracy and stop splitting a branch when performance saturates.
- Can be tricky for the same reasons that our proposed Base Case 3 can be.

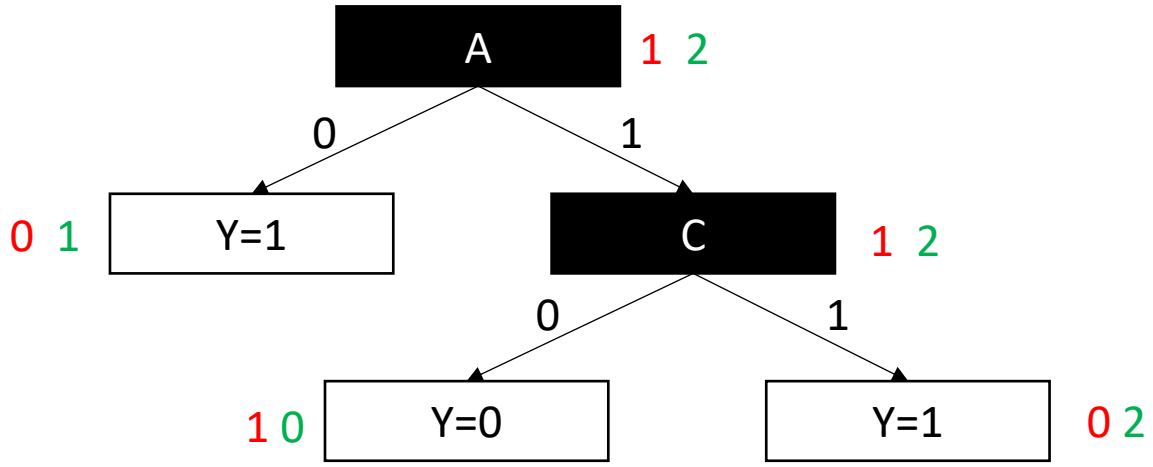
Option 3: Post Pruning

- **Grow full tree on the training set, then consider the impact of removing each node on validation performance.**
- **Greedily prune the node that most improves validation set performance.**

Post Pruning

Training Set

A	B	C	Y
1	0	1	1
0	1	1	1
1	1	0	0
0	0	1	1



What is the training accuracy here?

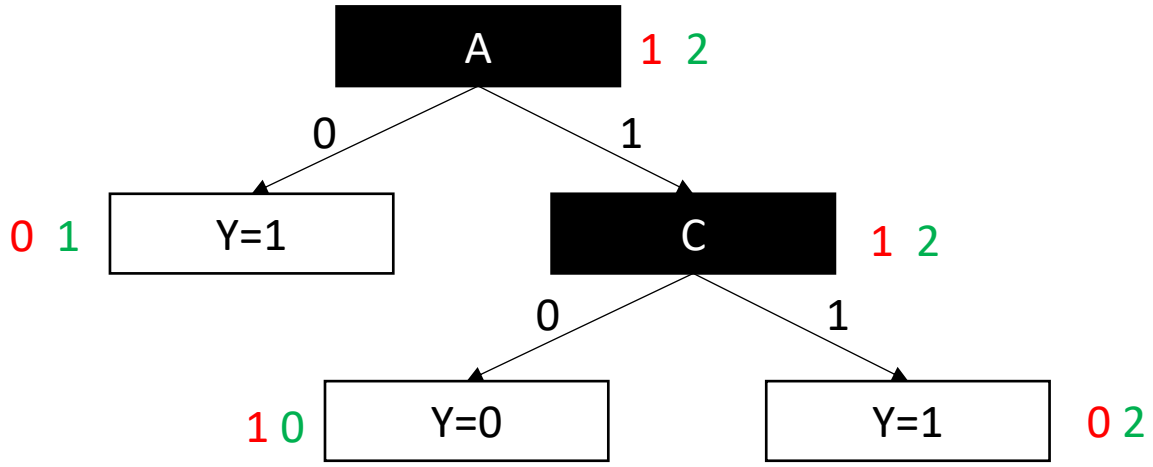
Post Pruning

Training Set

A	B	C	Y
1	0	1	1
0	1	1	1
1	1	0	0
0	0	1	1

Validation Set

A	B	C	Y
1	1	0	1
0	0	1	1
1	0	0	1
1	1	1	1



What is the validation accuracy here?

What if we removed node C and replaced it with a leaf where Y=1?

Summary of Decision Trees

Decision trees are very flexible classifiers

- Can model arbitrarily complex decision boundaries
- Complexity of model can be controlled by tree depth
- Handles both continuous and discrete features
- Can be used for both classification and regression

Learning decision trees

- Greedy top-down selection of splits
- Not guaranteed to find an optimal tree (smallest with minimal error)

Decision trees can easily overfit

- Can avoid this with early stopping or post pruning

Questions Someone Might Ask You About Decision Trees

Decision Trees: What is a decision tree? What can decision trees learn? How are they used to make predictions for classification and regression? Why are decision boundaries for decision trees composed of axis-aligned segments? What is the maximum depth a decision tree could reach? When would a leaf of a fully-expanded decision tree have more than one datapoint in the leaf?

Training Decision Trees: How are decision trees trained? Does the training algorithm you describe guaranteed to find the optimal tree? What is Entropy? What is Conditional Entropy? What is Information Gain? Why do we rank splits by Information Gain? How can decision trees handle continuous variables?

Overfitting: How can you avoid overfitting in decision trees? What is post-pruning? Can you perform post-pruning?

Topic Area:

Bias and Variance

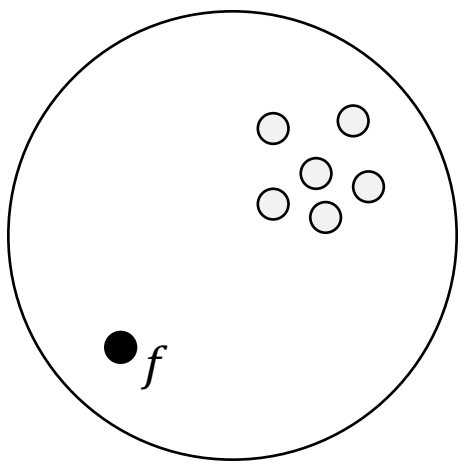
Bias-Variance Tradeoff

Bias: Error due to assumptions in the model not matching the problem (aka modelling error)

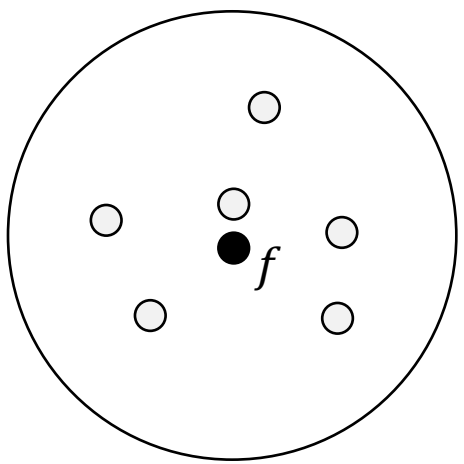
- More formally, average error between model and the true function over all possible datasets.

Variance: Error due to sensitivity to changes in the dataset (aka estimation + optimization)

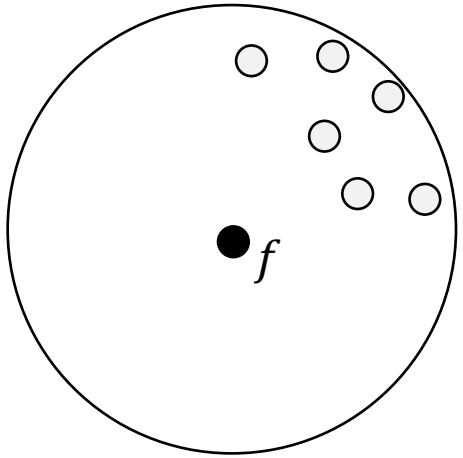
- More formally, variance of error between model and the true function over all possible datasets.



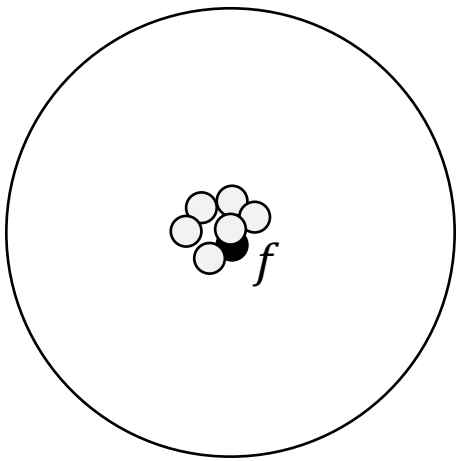
High Bias
Low Variance



Low Bias
High Variance

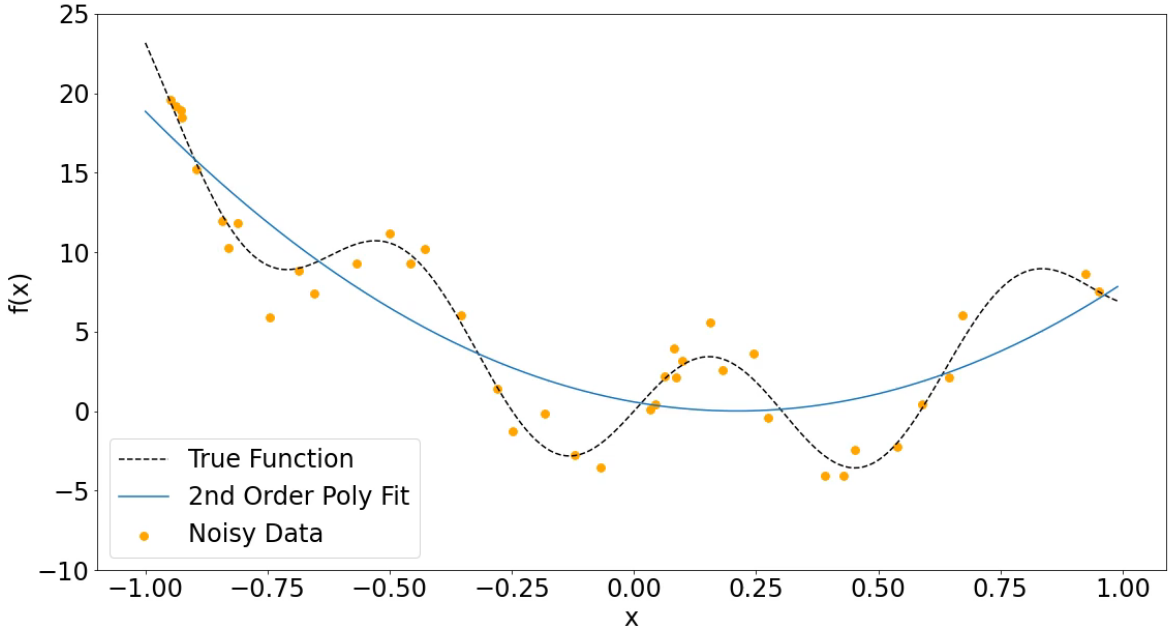


High Bias
High Variance



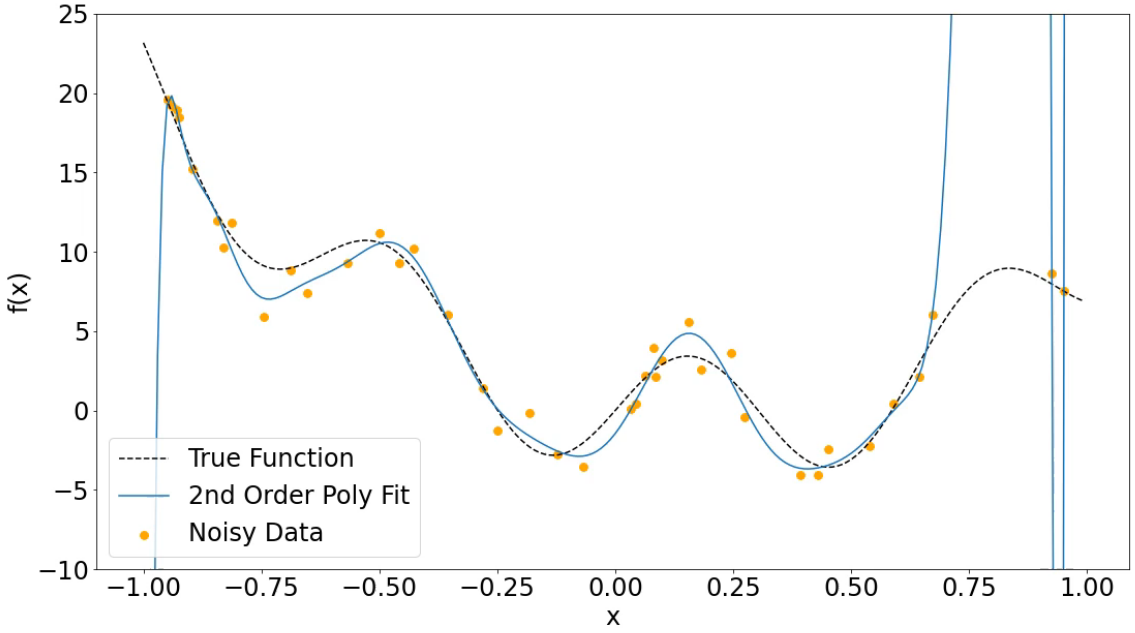
Low Bias
Low Variance

Bias-Variance Tradeoff



2nd Order Polynomial Fit:
Low variance / high bias

20th Order Polynomial Fit:
High variance / low bias



What We've Learned in Context of the Bias-Variance Tradeoff

Low Variance / High Bias Learners: (aka "weak" learners)

- Naïve Bayes, Logistic Regression, Decision Stumps (or shallow decision trees)
  - Good: Low variance – don't usually overfit
  - Bad: High bias – can't represent complex functions

High Variance / Low Bias Learners: (aka "strong" learners)

- Kernel methods, Neural Networks, Large Decision Trees, kNN
  - Good: Low bias – have the potential to learn complex functions
  - Bad: High variance – easy to overfit to training dataset

Can we get the best of both?

- In general, no… no free lunch.
- But often yes!

Questions Someone Might Ask You About Bias and Variance

What is bias? What is variance? Why do “weak” models tend to have high bias but low variance? Why do “strong” models tend to have low bias but high variance? How do bias and variance relate to sources of error we discussed earlier in the course (optimization / estimation / modelling / Bayes)?

Topic Area:

Ensemble Methods

Core Intuition of Ensemble Methods: A combination of multiple classifiers may perform better than a single classifier.

Instead of learning a single model, learn many models. Final output of the "ensemble" of models is a combination of each model's output.

Ensemble methods work well when each individual member is:

- Accurate (Better than chance)
- Diverse (Uncorrelated errors on new examples) ← **Why?**

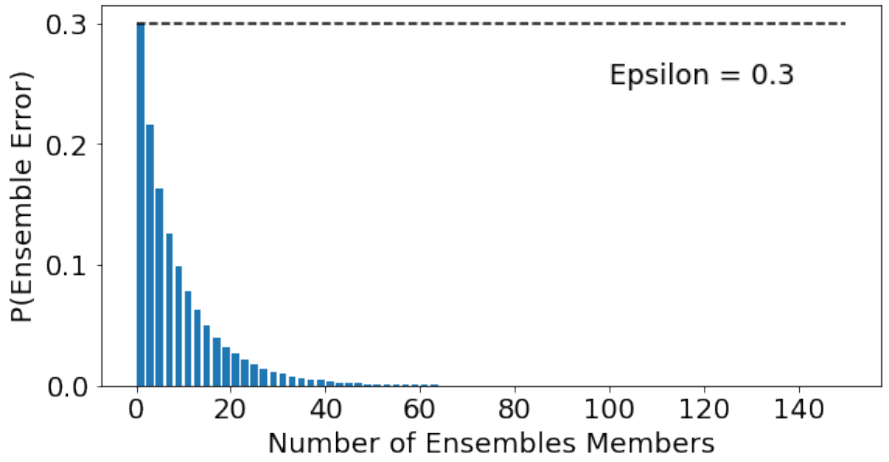
Consider an ensemble of M models that takes the majority vote of its members as the final output. Assume each as an uncorrelated error rate ϵ

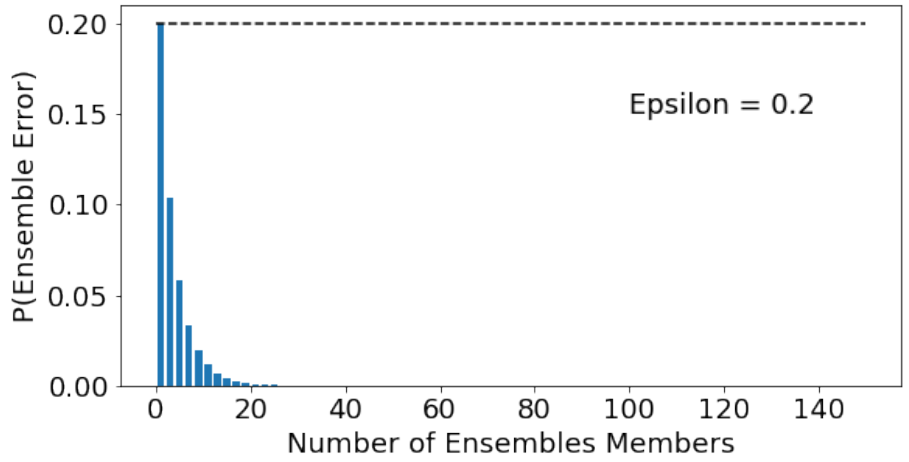
Probability that a majority (> M/2) models simultaneously make an error:

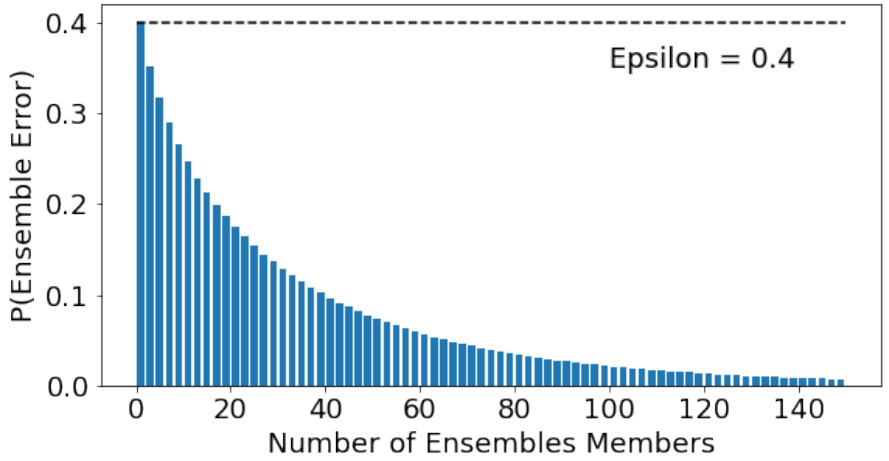
$$P\left(\#errors \geq \frac{M}{2}\right) = \sum_{h=M/2}^{M} \binom{M}{h} \epsilon^h (1-\epsilon)^{M-h}$$

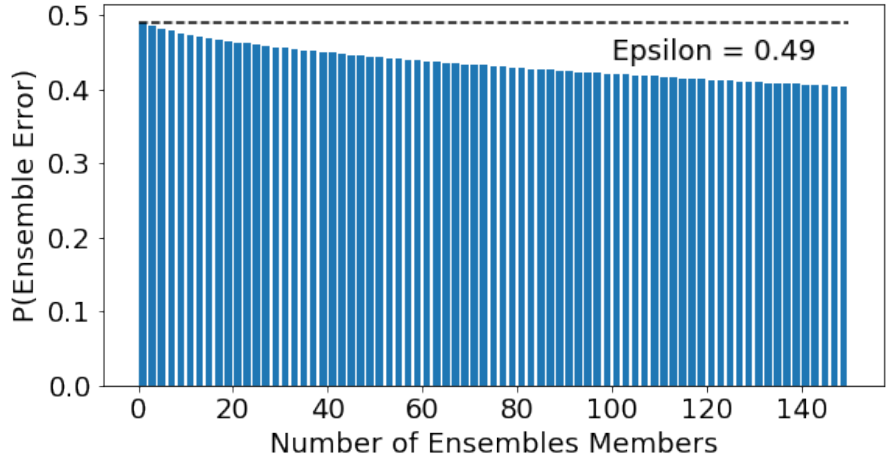
Assuming uncorrelated error rates of ϵ,

$$P\left(\#errors \geq \frac{M}{2}\right) = \sum_{h=M/2}^{M} \binom{M}{h} \epsilon^h (1-\epsilon)^{M-h}$$









Ensemble Methods

Idea: Instead of learning a single model, learn many models. Final output of the "ensemble" of models is a weighted combination of each model's output.

Bagging: Try to reduce variance in strong learners

- Uncorrelated errors → expected error goes down
- On average, do better than single classifier!

Boosting: Try to reduce bias in weak learners

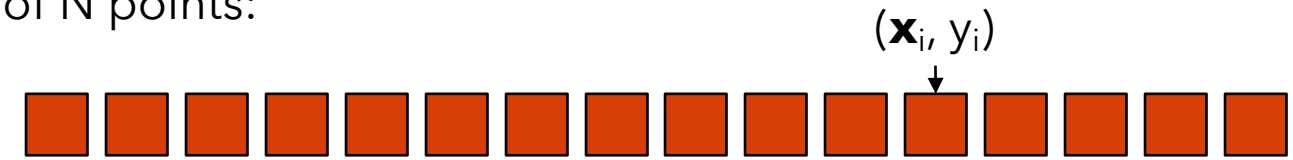
- Each model good at different parts of the input space
- On average, do better than single classifier!

Adapted from Dhruv Batra. The cats are from me.

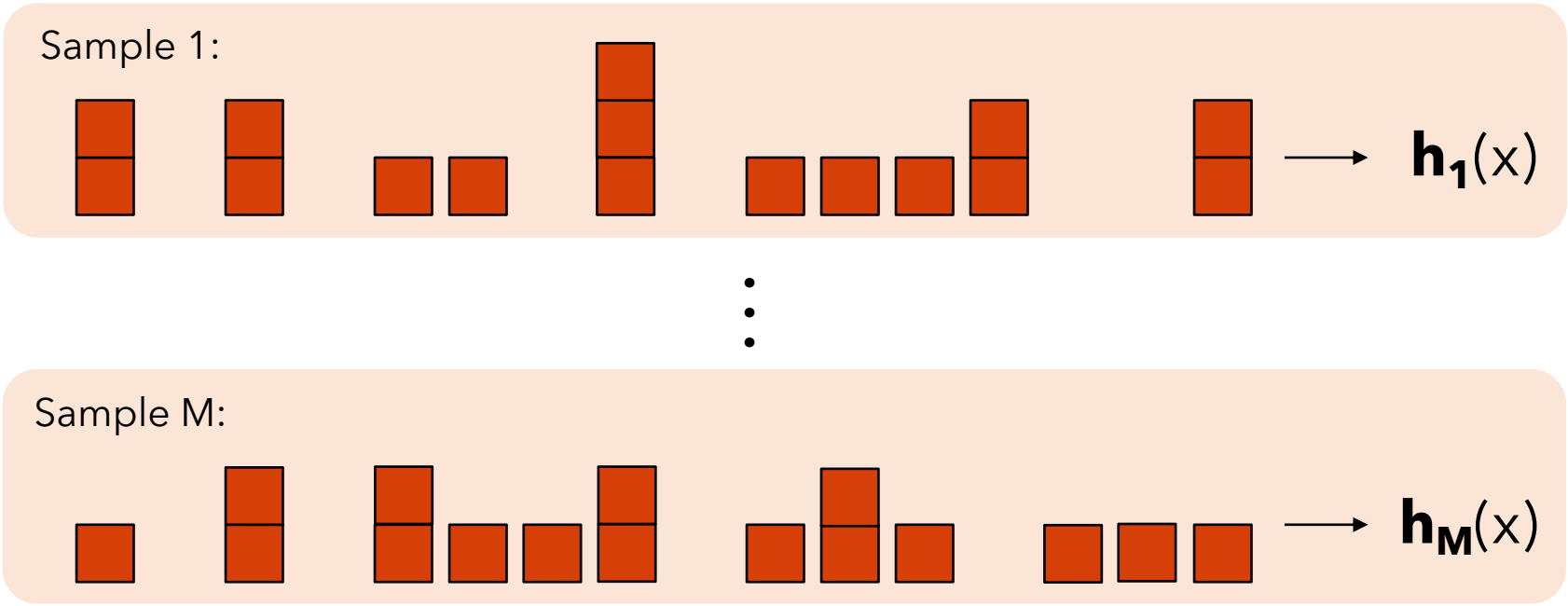
Bagging (**B**ootstrap **Agg**regat**ing**)

Bagging: Average multiple models trained from resamples of the data.

Given a dataset of N points:



Sample N training points **with replacement** and train a model, repeat M times:



Bagging (**B**ootstrap **Agg**regat**ing**)

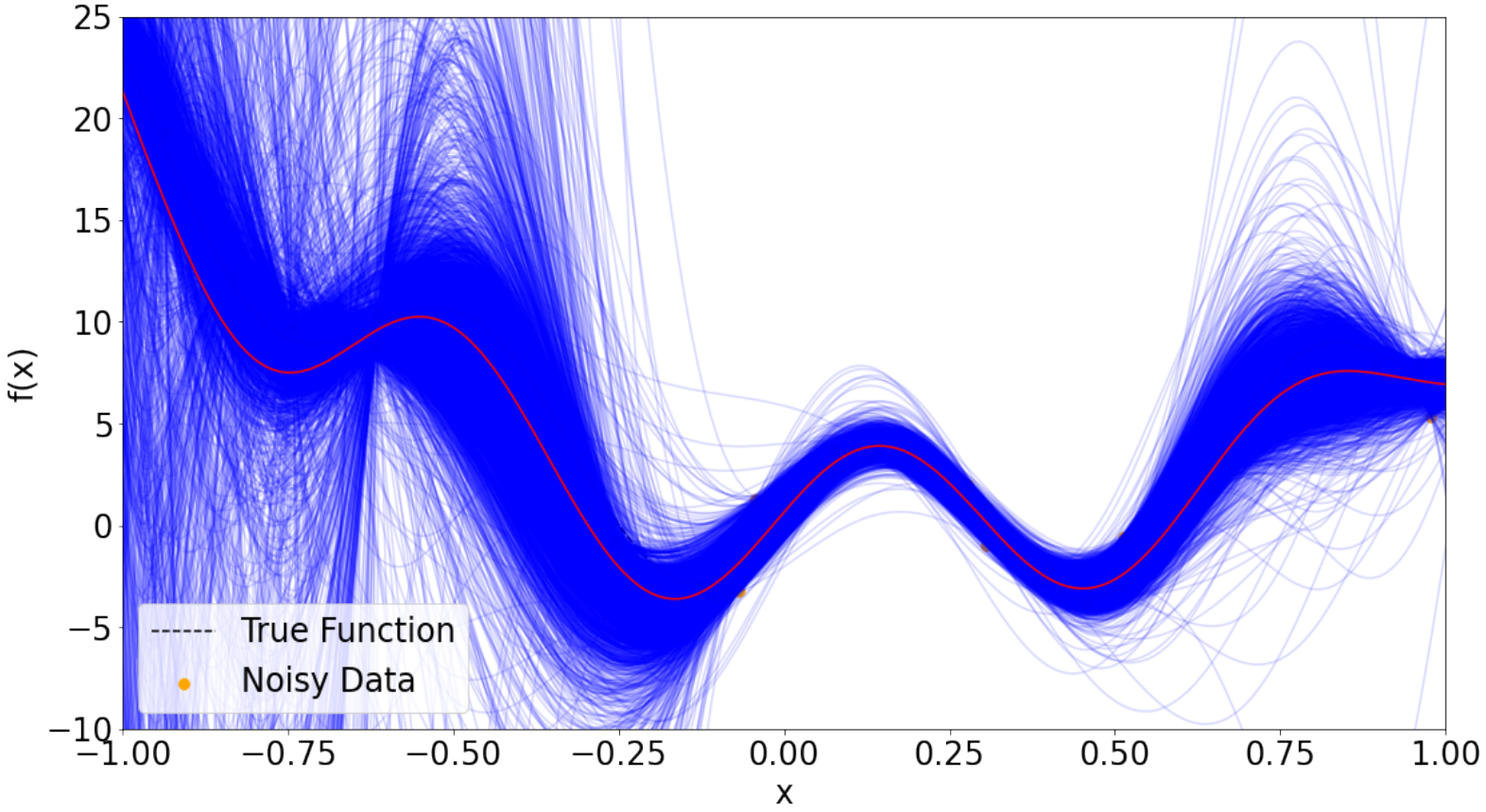
Bagging: Average multiple models trained from resamples of the data.

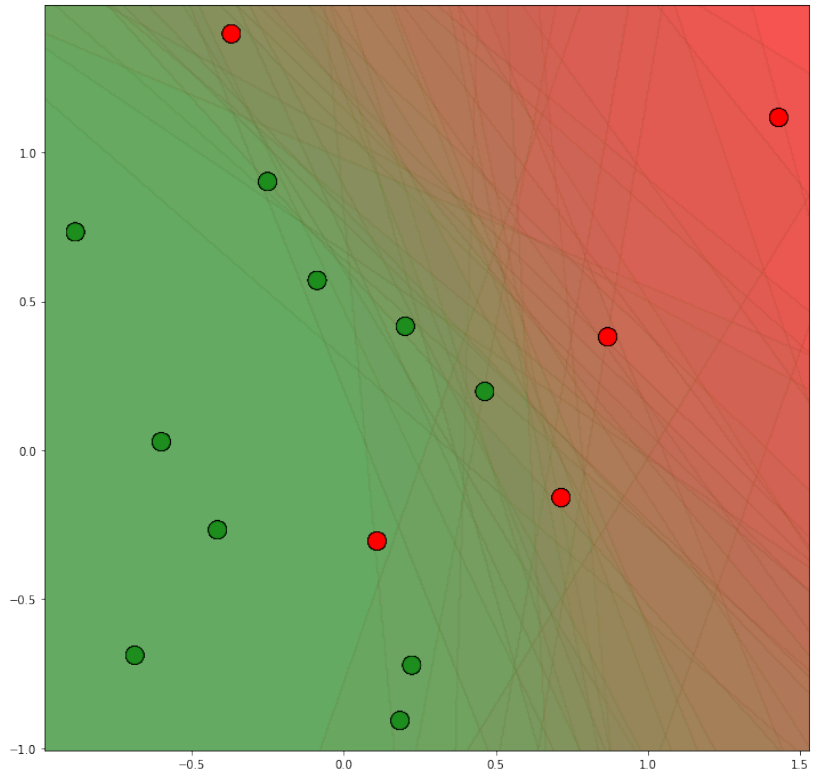
Regression: At test time, simply run each model and take the average of their outputs:

$$y_{pred} = \frac{1}{M} \sum_{m=1}^{M} h_m(x)$$

Classification: Can take the majority vote instead (if averaging classifier output isn't meaningful)

Bagging (**B**ootstrap **Agg**regat**ing**)





Bagging is most useful for strong learners like neural networks and decision trees that have high variance but low bias.

- **Works best when errors in each ensemble member are not correlated.**
- Each ensemble member can be trained **in parallel**.

Use bagging when…

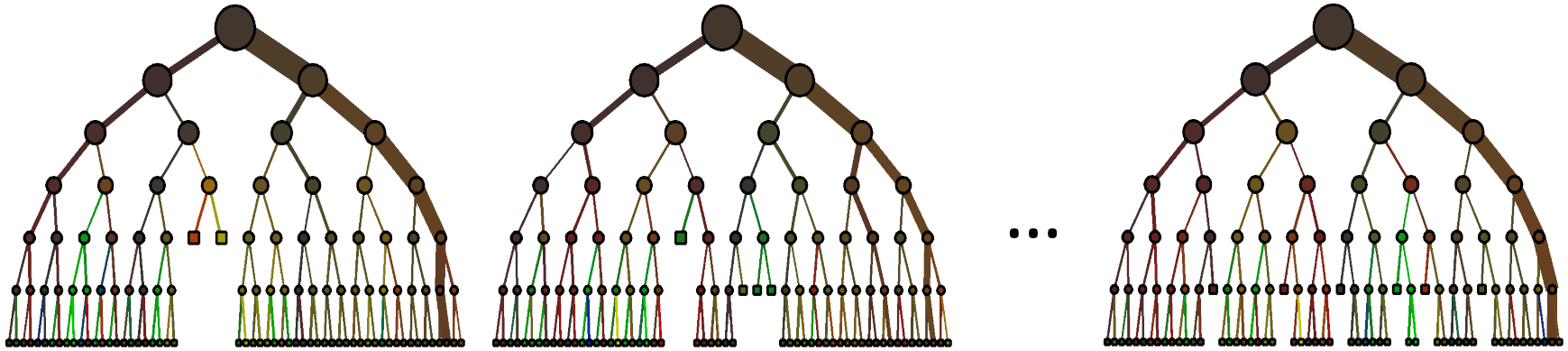
… you have overfit strong learners

… you have a somewhat reasonably sized dataset

… you want an extra bit of performance from your models

Let's consider applying bagging to decision trees.

Train a M trees on different resamplings of the data and call it a **forest**.



Make prediction by majority vote over the trees.

Issue: Our greedy decision tree learning algorithm will likely use the same attributes early on, despite a resampling. **Why is this an issue?**

Random Forests:

- Train an ensemble of decision trees with bagging.

- During tree learning, at each split consider only a random subset of attributes / thresholds when choosing what to split on.
  - Often $\sqrt{d}$ features are considered at each split for d-dimensional inputs

- Output is majority vote from the forest.

Random forests are incredibly widely used in practical applications involving medium-sized dataset. An excellent "first attempt" for supervised problems with relatively low dimensionality.

Ensemble Methods

Idea: Instead of learning a single model, learn many models. Final output of the "ensemble" of models is a weighted combination of each model's output.

Bagging: Try to reduce variance in strong learners

- Uncorrelated errors → expected error goes down
- On average, do better than single classifier!

Boosting: Try to reduce bias in weak learners

- Each model good at different parts of the input space
- On average, do better than single classifier!

Adapted from Dhruv Batra. The cats are from me.

Boosting

Idea: Train classifiers sequentially. After each classifier, focus more on points the previous models have large errors on.

- First train the base classifier on all the training data with equal importance weights on each datapoint.
- Then re-weight the training data to emphasize the hard datapoints and train a second model.
  - How do we re-weight the data?
- Repeat this and keep training new models on re-weighted data
- Finally, use a weighted ensemble of all the models for the test data.
  - How do we weight the models in the committee?

Example: L2 Boosting (Regression)

More generally, let the error after the m^{th} model be:

$$e_i^{(m)} = y_i - \sum_{j=1}^{m} h_j(x_i)$$

Train the next classifier against this error:

$$h_{m+1} = \underset{h}{argmin} \sum_i \left(e_i^{(m)} - h(x_i)\right)^2$$

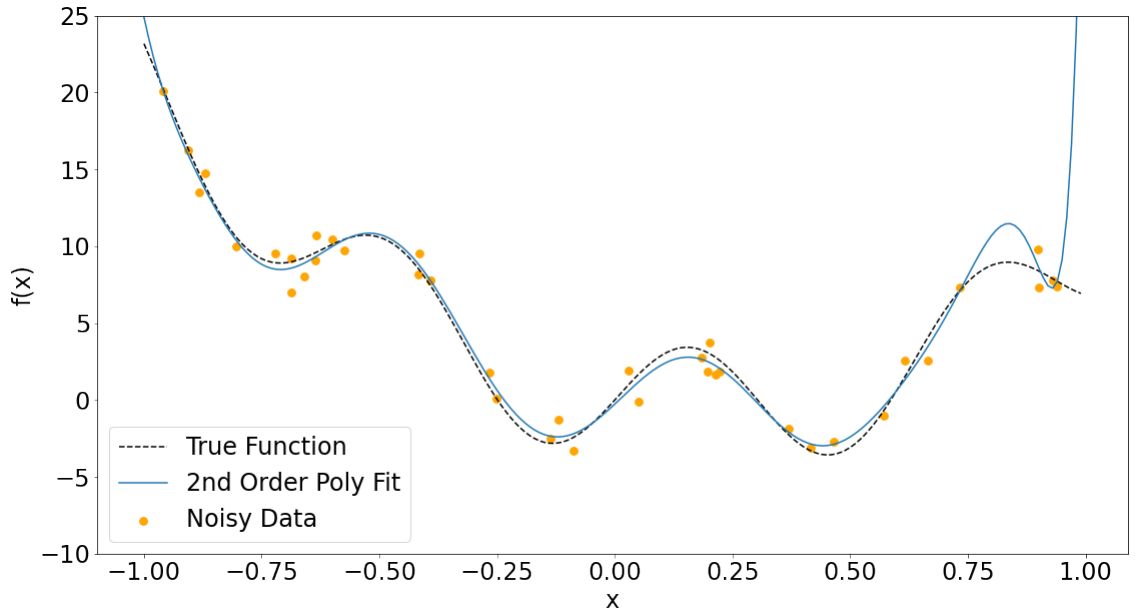
Easy to implement, just replacing y_i with $e_i^{(m)}$ in your favorite regression algorithm (e.g. regression trees).

Example: L2 Boosting (Regression)

Let's see this in practice.

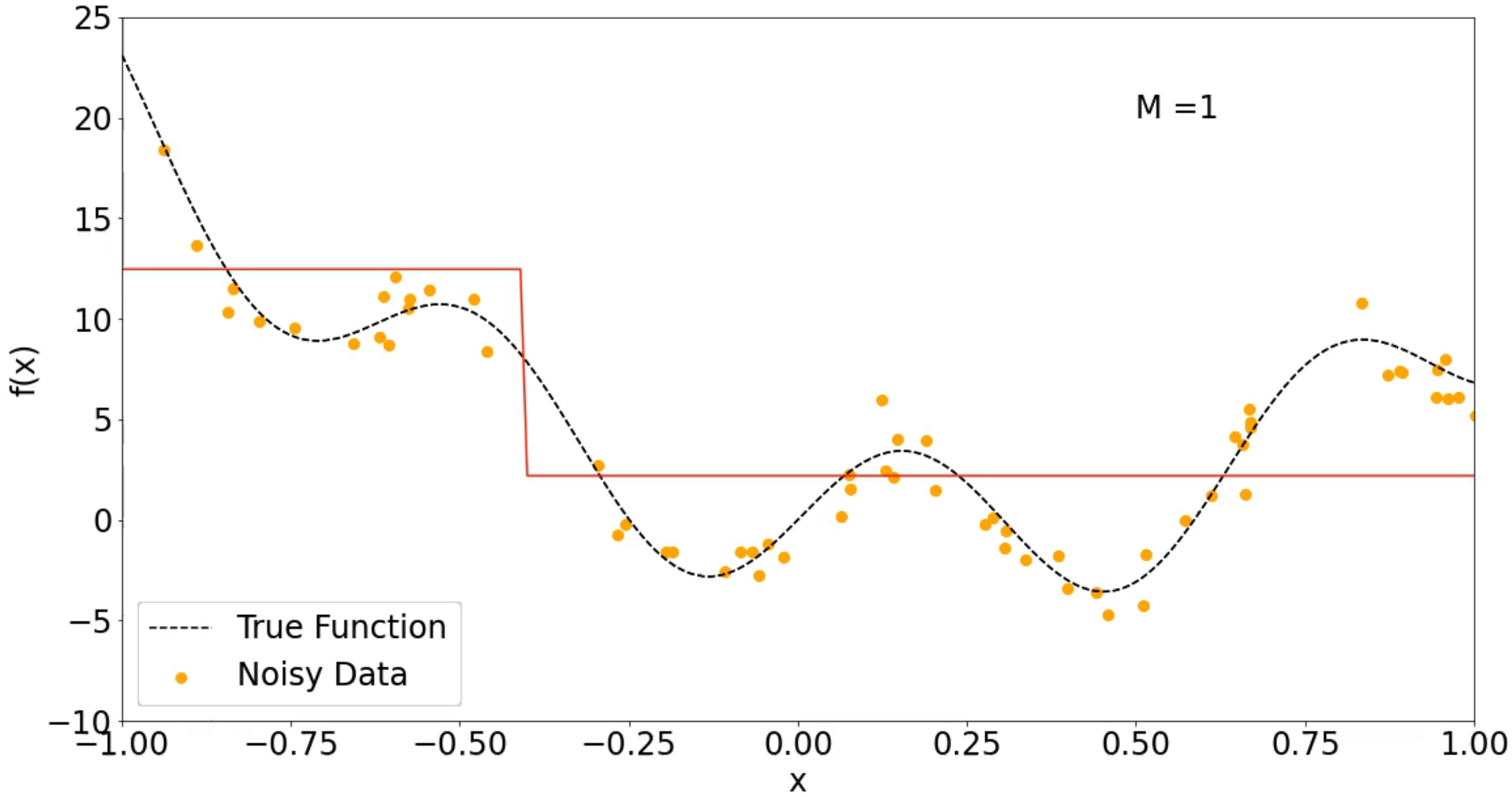
Model: Decision tree for regression, max depth 2.

Data: Same squiggly thing I keep showing.



Example: L2 Boosting (Regression)

M is number of depth-2 regression trees in the boosted ensemble.



Summary of Ensemble Methods

Ensembles are sets of models trained for the same problem.

- Ensemble output is often the average (regression) or majority (classification) class output.
- Ensembles almost always lead to improvements over single models and are widely used in practice.

Bagging is a way to make an ensemble that reduces variance of strong models.

- Resample dataset with replacement to learn each model in the ensemble.
- Models can be trained in parallel.
- One example we talked about is **Random Forests.**

Boosting is a way to make an ensemble that reduces bias of weak models.

- Sequentially learn classifiers that focus more on datapoints where the current models have errors.

Questions Someone Might Ask You About Ensemble Methods

Bagging: When is it useful to apply bagging? Does bagging help reduce bias or variance? How does bagging work? How do ensembles make predictions? How does the correlation between model outputs effect the performance of a bagged ensemble? What are Random Forests? Why do we want to introduce additional randomness in Random Forests?

Boosting: When is it useful to apply boosting? Does boosting help reduce bias or variance? How does boosting work in general? How does L2 boosting work?

Topic Area:

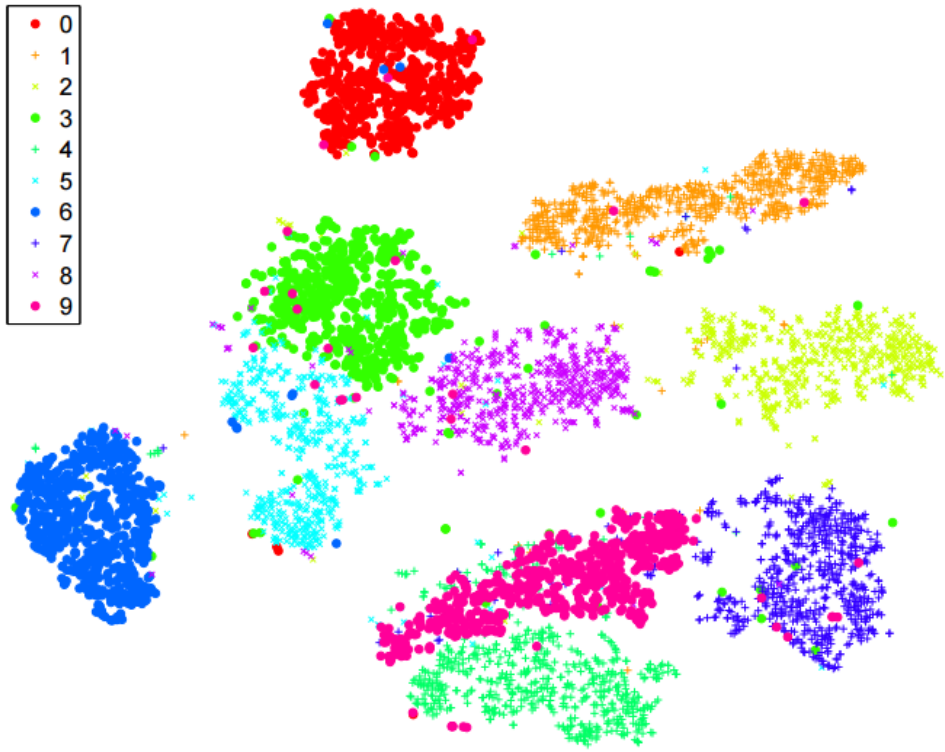
Clustering: k-means and GMM

New Topic: Clustering

In general, clustering can be viewed as an exploratory procedure for finding interesting subgroups in a dataset.

Most work in clustering is on the setting where we want to:

- Group all given examples **(exhaustive)** into disjoint clusters **(partitional)** such that
  - Examples within a cluster are similar
  - Examples in different clusters are different

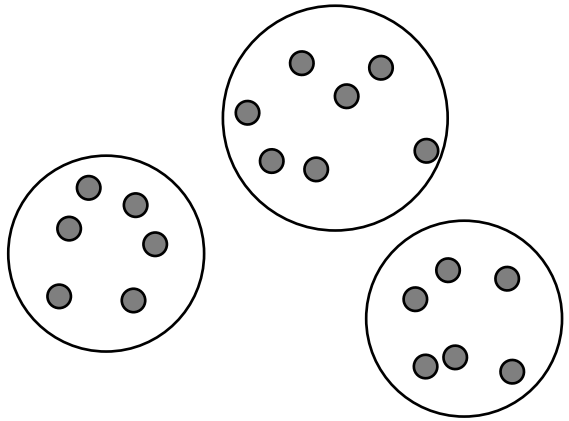


Slide adapted from Xiaoli Fern

Two Categories of Clustering Algorithms

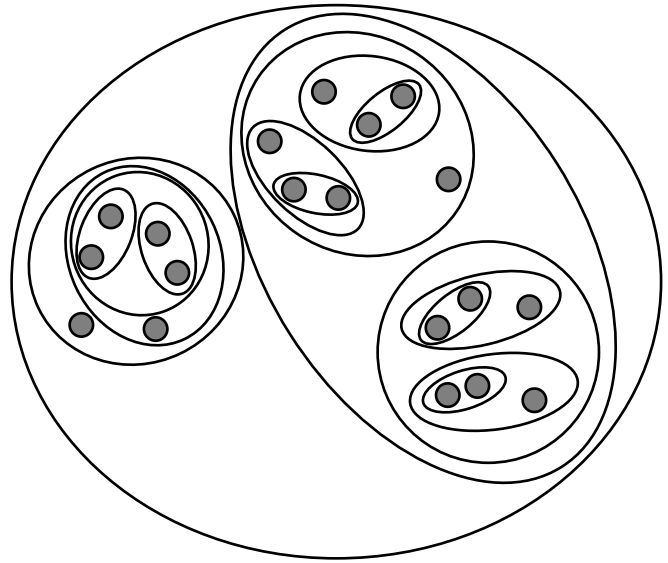
Partition Algorithms ("Flat" Clusterings)

- k-Means / k-Medians
- Gaussian Mixture Models



Hierarchical Algorithms

- Bottom-up – Agglomerative
- Top-down – Divisive

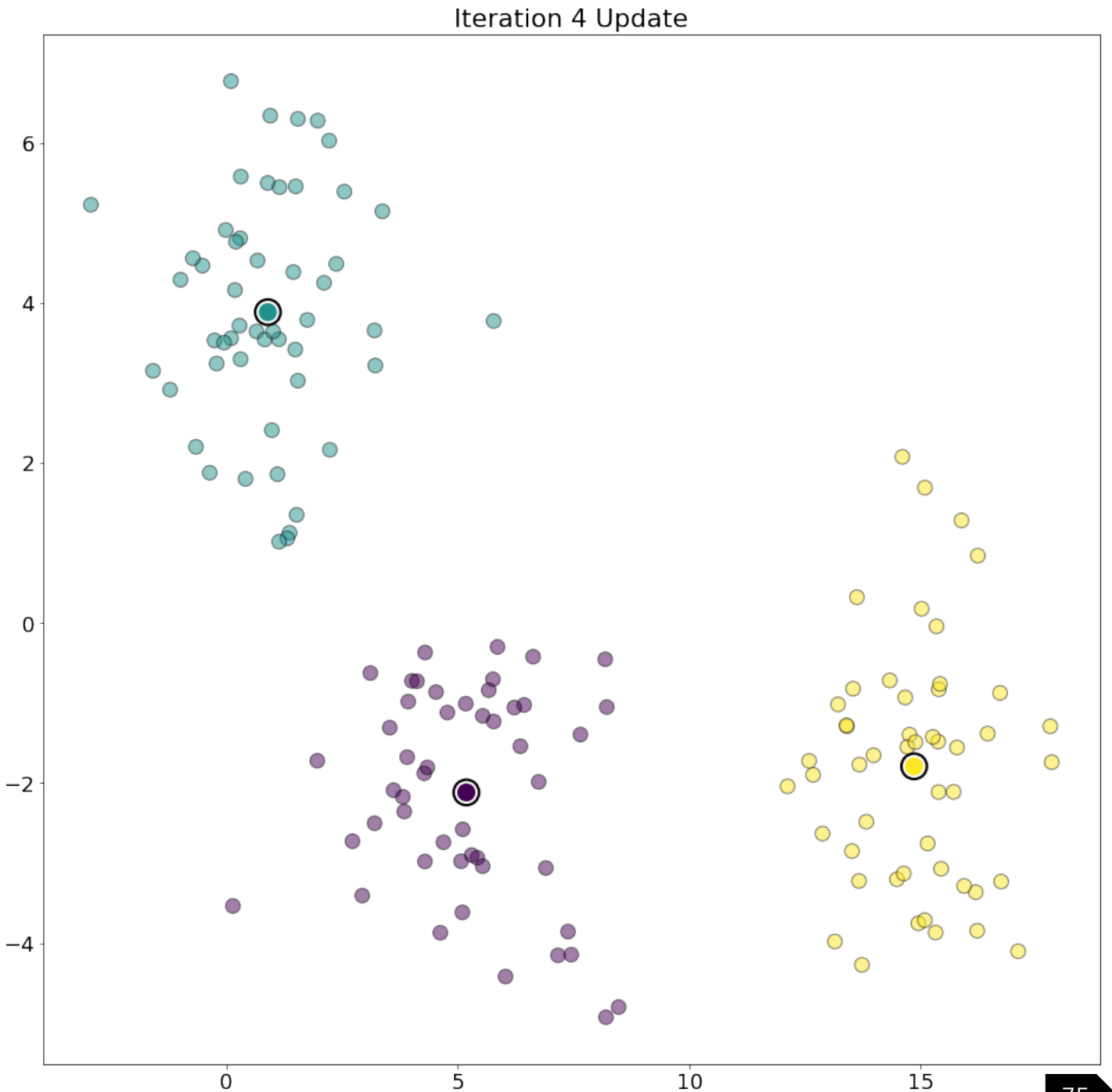


Slide adapted from Xiaoli Fern

K-Mean Clustering

Algorithm:

1. Initialize *k* centroids randomly
2. While not converged:
    a) Associate each point with its nearest centroid
    b) Update each centroid as the average of associated points
3. Return centroids and associations



Formalizing K-Mean Clustering

Given a dataset $X = \{\boldsymbol{x}_i\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and the number of desired clusters k, **produce** a set of assignments $Z = \{z_i\}_{i=1}^n$ where $z_i \in \{1,2,\dots,k\}$ and a set of centroids $C = \{\boldsymbol{c}_j\}_{j=1}^k$ where $\boldsymbol{c}_j \in \mathbb{R}^d$.

Do this by alternating the following steps:

a) Assignment: For each datapoint *i*, update z_i:

$$z_i = \underset{j=1,2,\dots,k}{\operatorname{argmin}} \left\| \boldsymbol{x}_i - \boldsymbol{c}_j \right\|_2^2$$

b) Update: For each centroid j, update $\boldsymbol{c}_j$:

$$\boldsymbol{c}_j = \frac{\mathbf{1}}{\sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z}_i = j]} \sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z}_i = j] \;\; \boldsymbol{x}_i$$

Formalizing K-Mean Clustering

Given a dataset $X = \{\boldsymbol{x}_i\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and the number of desired clusters k, **produce** a set of assignments $Z = \{z_i\}_{i=1}^n$ where $z_i \in \{1,2,\dots,k\}$ and a set of centroids $C = \{c_j\}_{j=1}^k$ where $c_j \in \mathbb{R}^d$.

Do this by alternating the following steps:

a) Assignment: For each datapoint *i*, update z_i:

$$z_i = \underset{j=1,2,\dots,k}{\operatorname{argmin}} \left\| \boldsymbol{x}_i - \boldsymbol{c}_j \right\|_2^2$$

b) Update: For each centroid j, update $\boldsymbol{c_j}$:

$$\boldsymbol{c_j} = \frac{\mathbf{1}}{\sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z_i} = j]} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \mathbb{I}[\boldsymbol{z_i} = j] \;\; \boldsymbol{x_i}$$

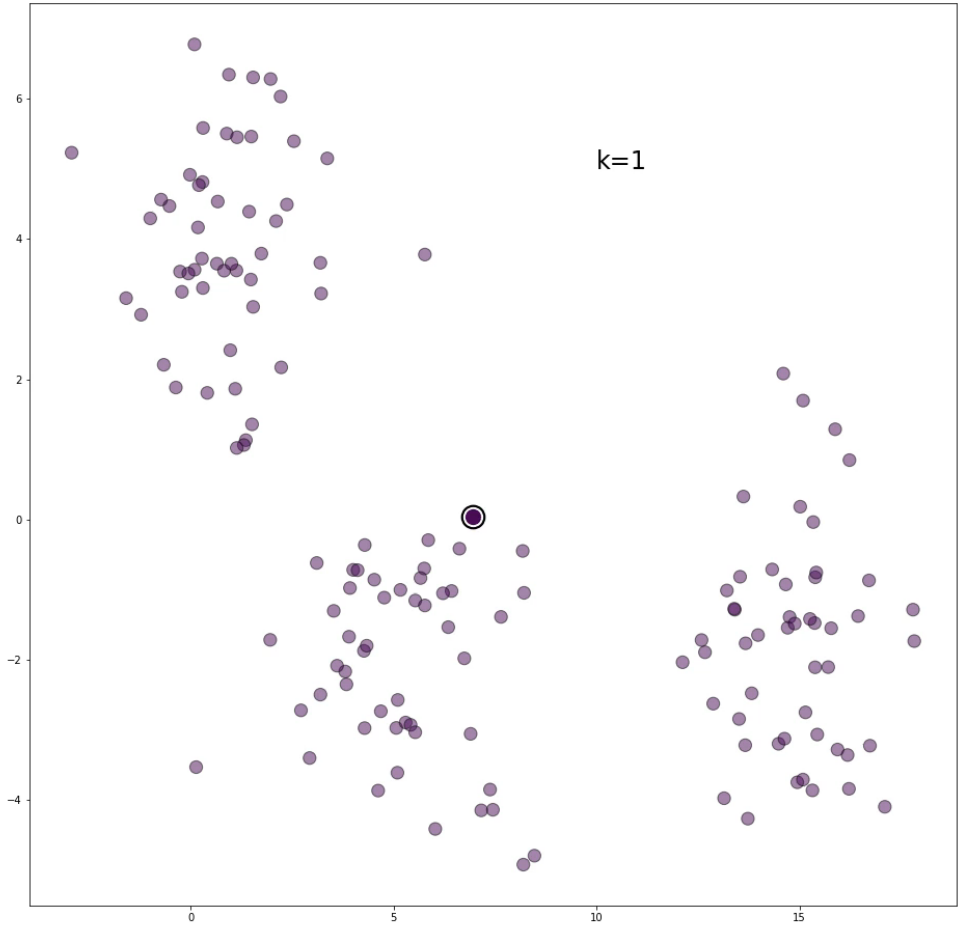
Now we know this is coordinate descent to minimize sum-of-squared error between centroids and datapoints!

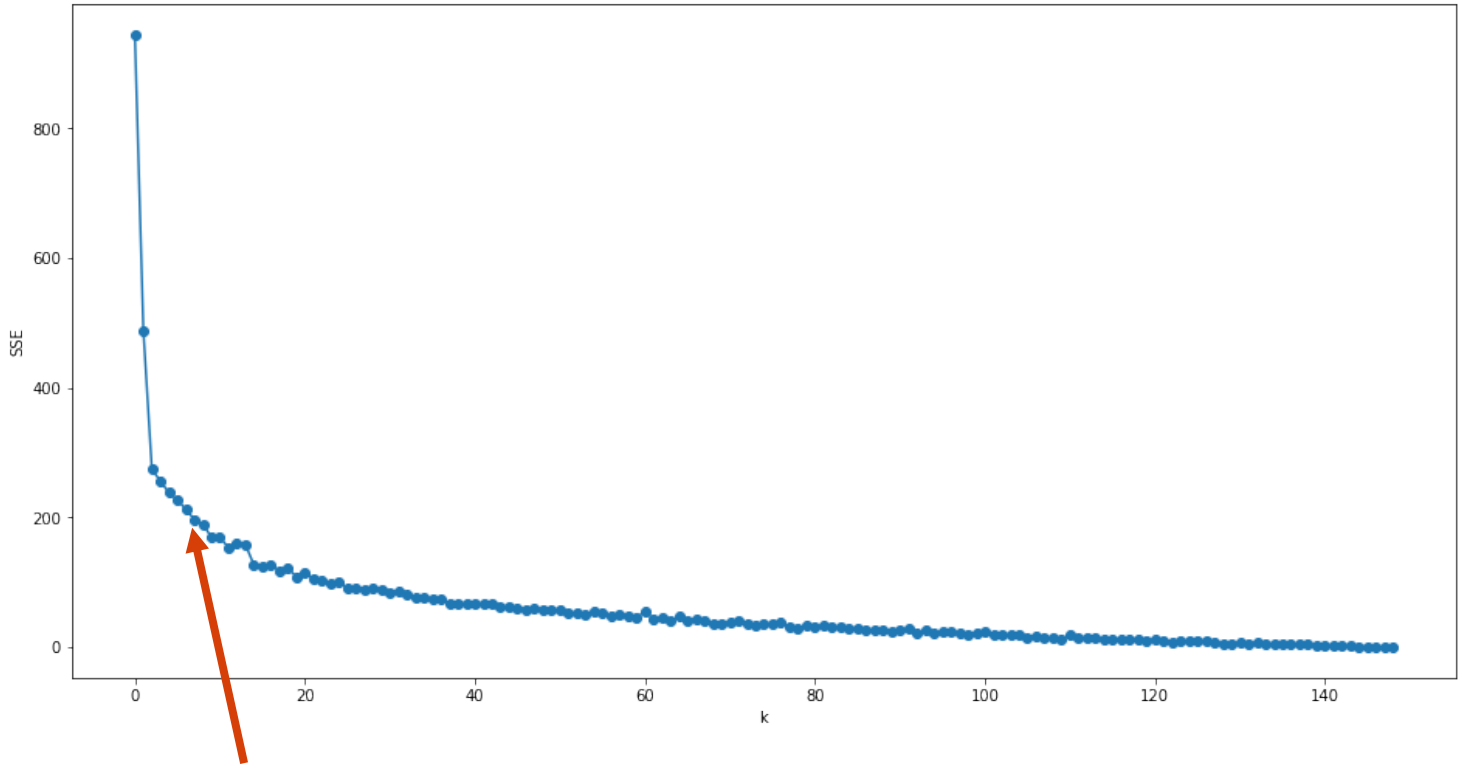
(There is a layer deeper than this we'll get to later.)

Properties: How do we choose **k**?

The number of clusters **k** is an important hyperpameter. How can we choose it?

- Unfortunately, SSE decreases with k so training set won't tell us much.



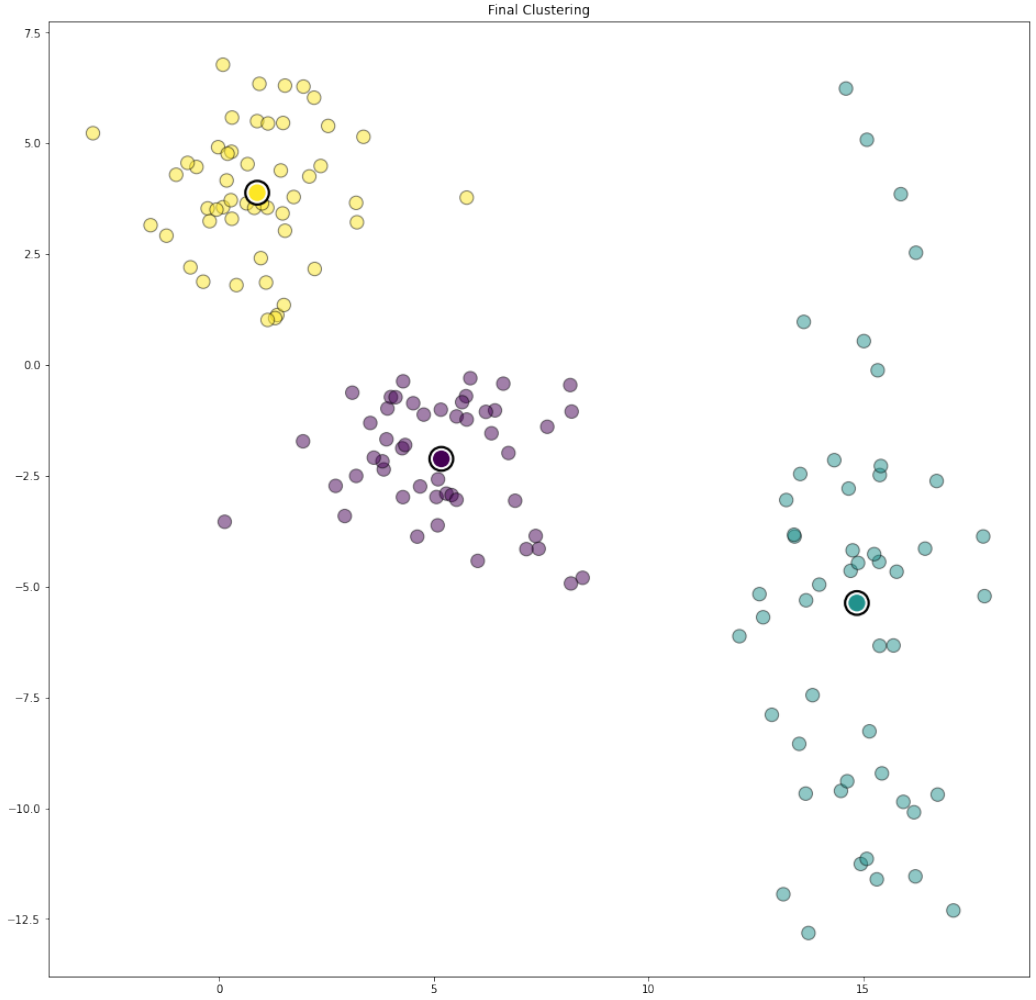


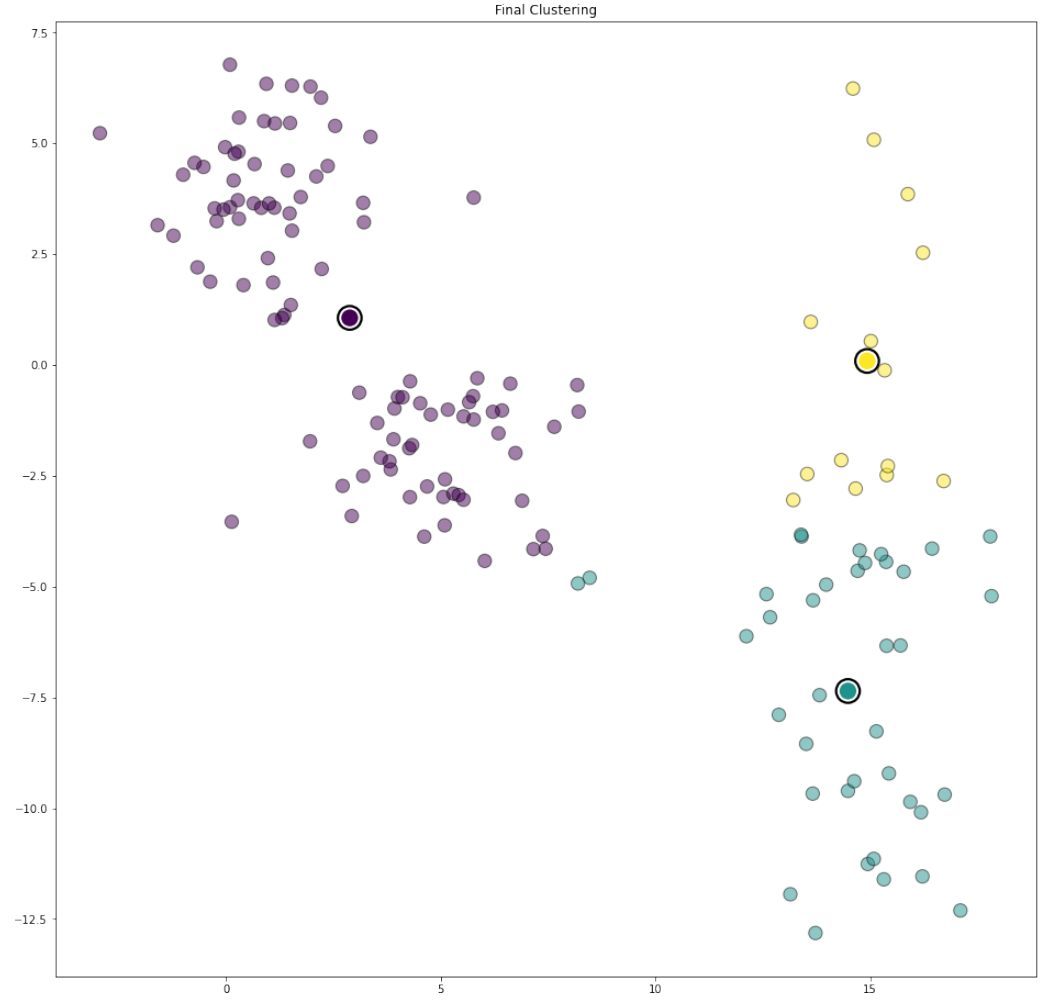
One common heuristic is to pick the "elbow/knee" of the SSE vs. k graph.

Properties: Highly Sensitive to Initialization

k-Means is highly sensitive to initialization of the centroids.

- Sum-of-squared error has many local minima where no local reassignments can reduce SSE.





Properties: Highly Sensitive to Initialization

How can we deal with this sensitivity?

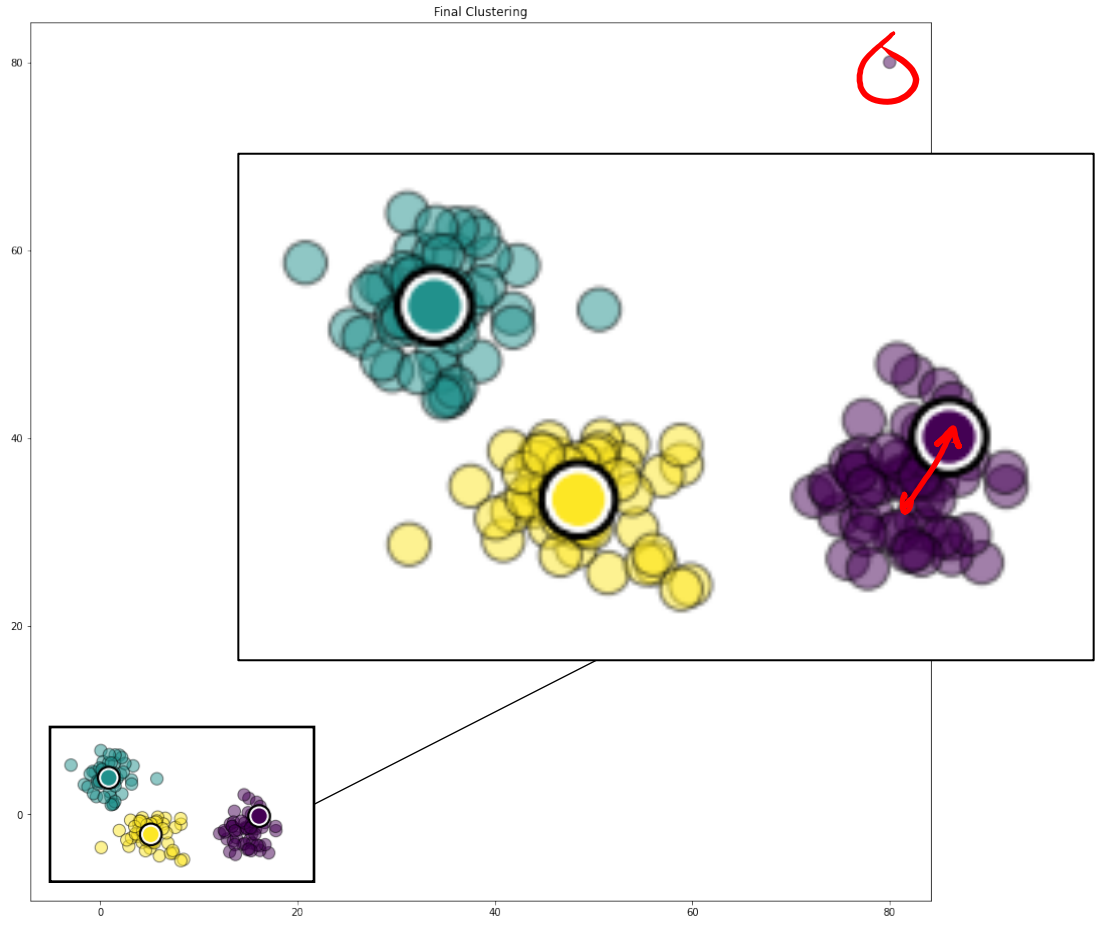
Run multiple trials and choose the one with the lowest SSE (often used in practice)

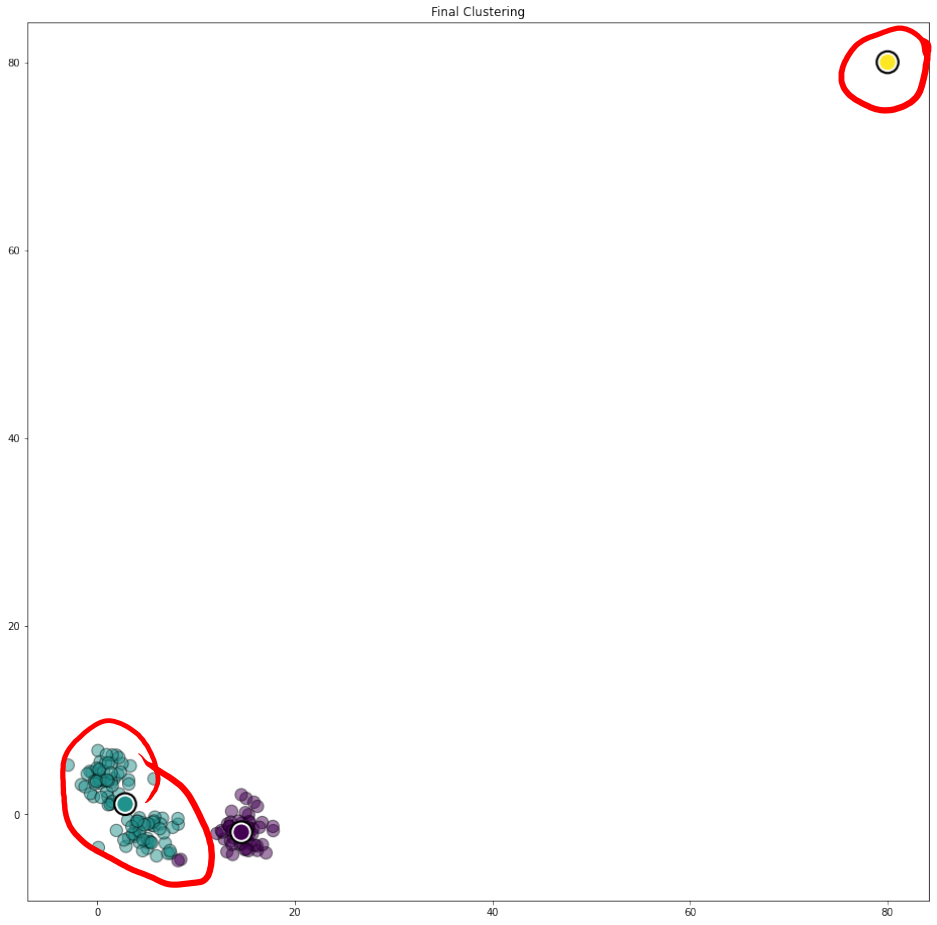
Try to initialize the centroids "well" – not clear how to do this. Some common heuristics:

- **Random Points:** Initialize the centroids by copying random datapoints – ensures your centroids are near data.
- **Far-Away Points:** Try to make the cluster centers far from each other:
  - Samples a datapoint and set the first centroid c_1 to its value
  - Compute a weight w_i for each datapoint proportional to its distance from c_1. Then sample according to this distribution.
  - Repeat with weights proportional to the nearest centroid.

Properties: Sensitive to Outliers

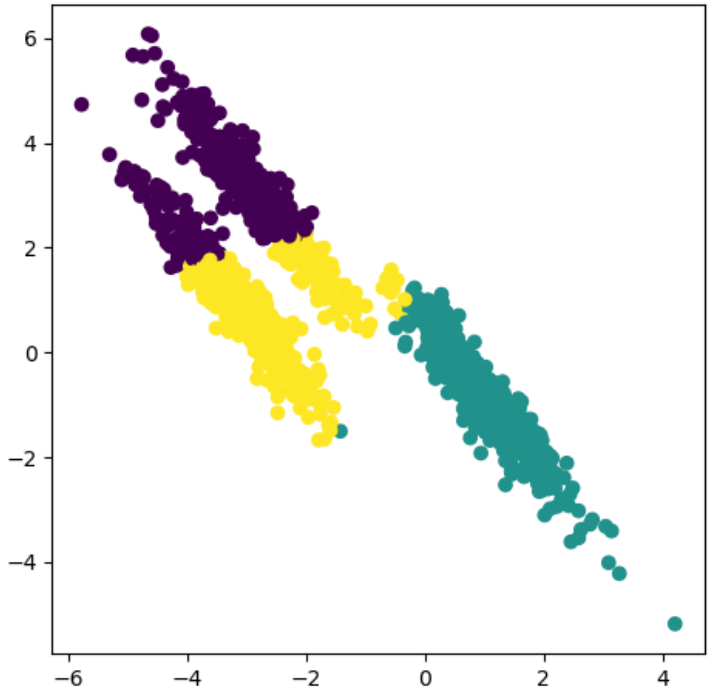
Every point needs to have a cluster and every cluster center is the average of its members.

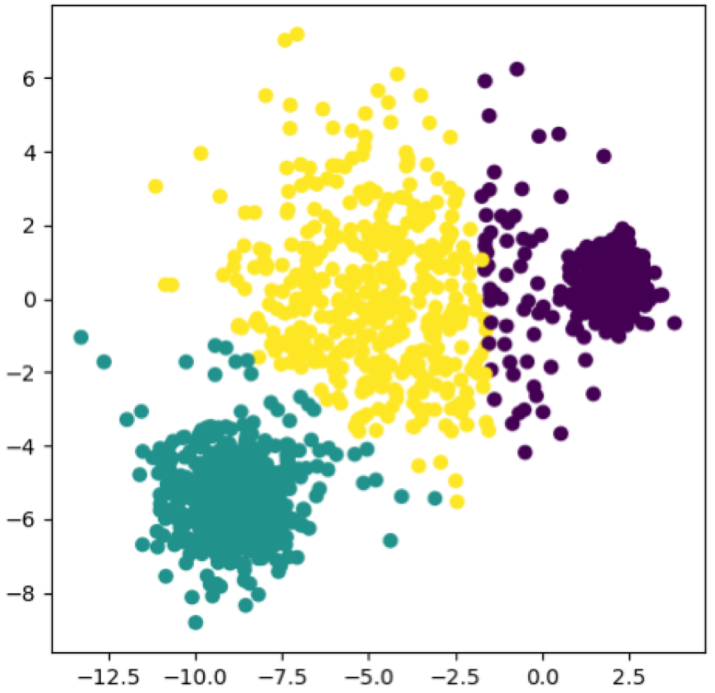




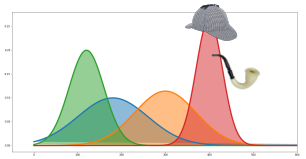
Properties: Some Hidden Assumptions

k-Means performs poorly when the clusters are not spherical or when different clusters have very different spreads. Seems like there are some hidden assumptions here…





Where do these assumptions come from? Spoiler: Its Gaussian again!



Figures adapted from Phil Roth

Summary of k-Means

Given a dataset, **k-Means splits it into k disjoint groups.**

- Setting k is largely heuristic as larger k produces lower SSE
  - Unsupervised learning has no validation set because it has no labels

- Guaranteed to converge in finite steps to a local minima
  - Often in a few iterations in practice

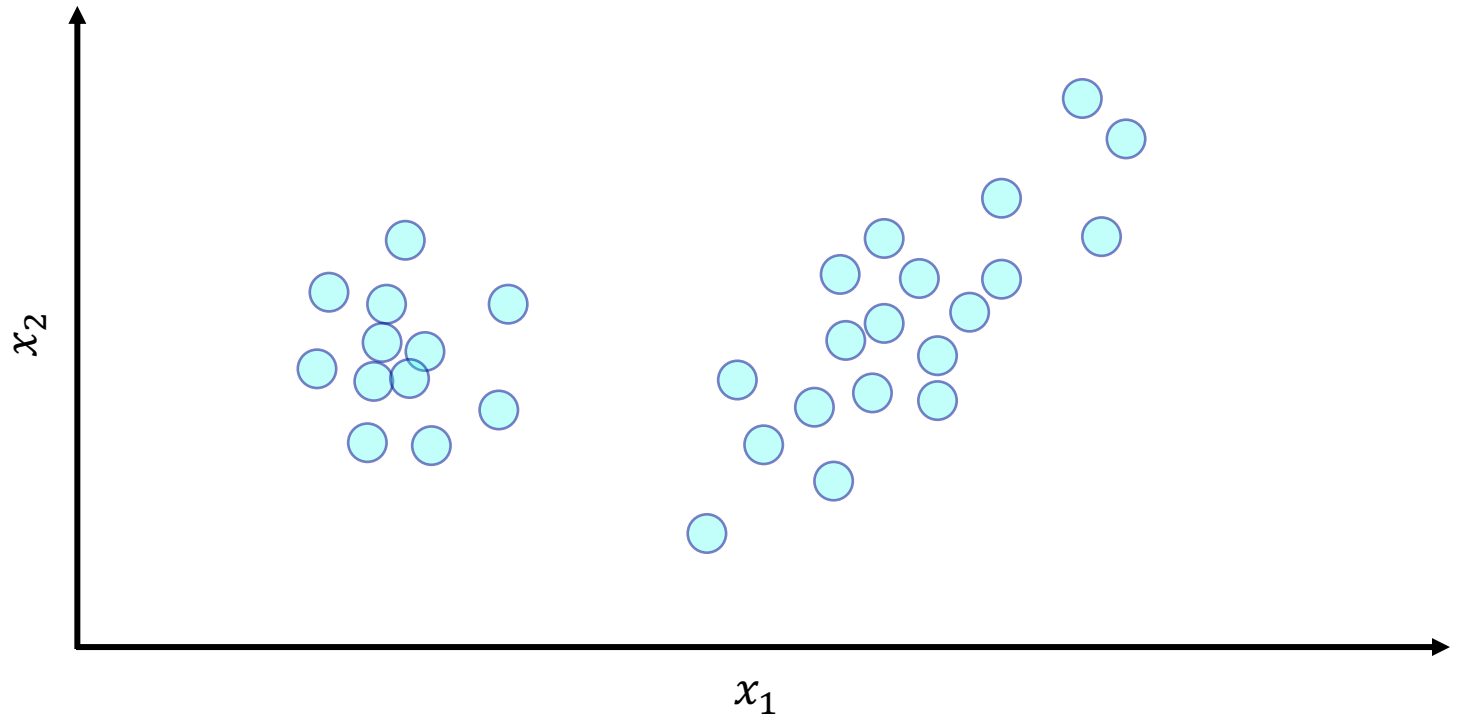
- O(mknd) computational complexity makes running k-Means tractable

Properties of k-means:

- Not particularly resistant to outliers
- Very sensitive to initialization of centroids – need to run multiple times
- **Only seems to work on spherical clusters with similar sizes**

Let's Start Simple – A 2d Case with k=2

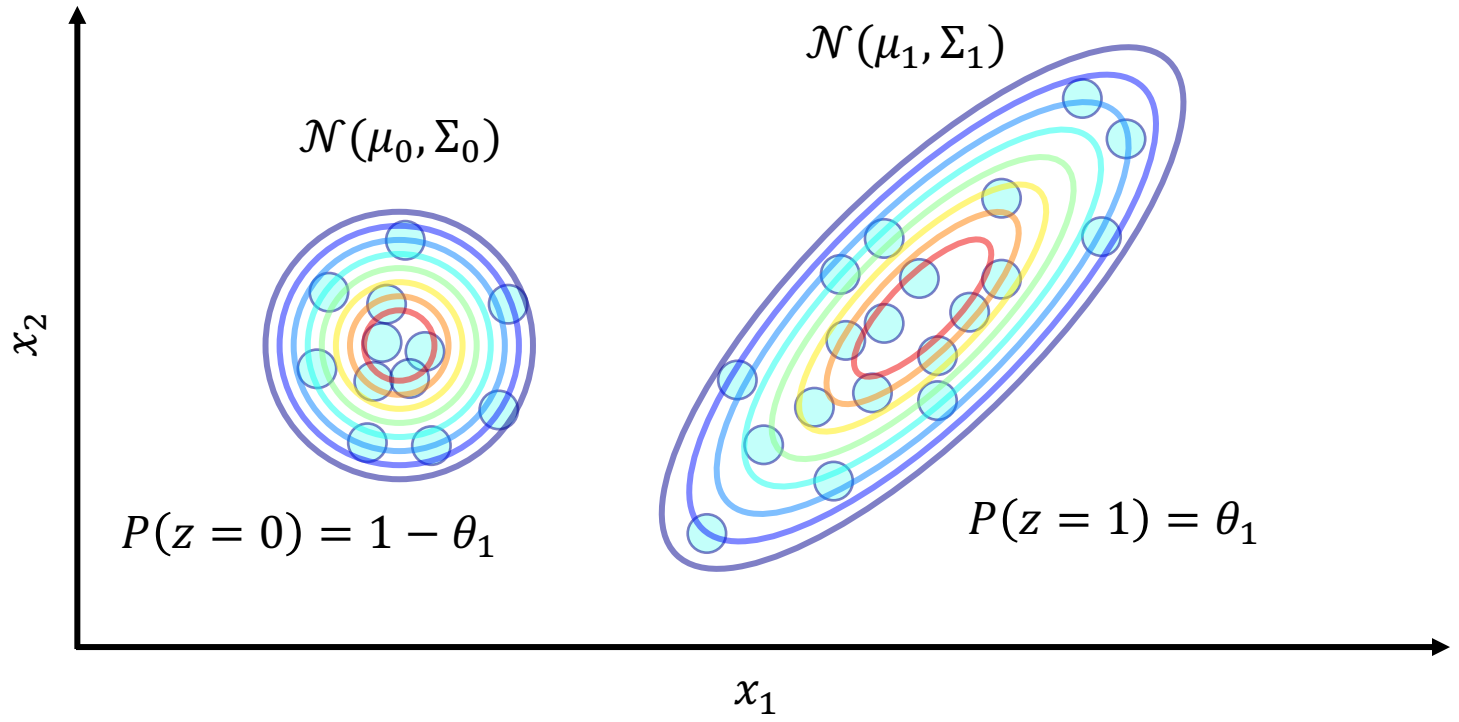
Suppose I observe the following 2-dimensional dataset. It appears to have two clusters, one on the left and one on the right.



Let's try modelling this as a mixture of two Gaussian distributions – that is, assume the data was generated by selecting one of two Gaussians and then sampling a point from the selected Gaussian.

Let's Start Simple – A 2d Case with k=2

Given a dataset of $D = \{x_i\}_{i=1}^n$ where $x \in \mathbb{R}$, want to learn $\mu_0, \mu_1, \Sigma_0, \Sigma_1$, and θ_1. $(\theta_0 = 1 - \theta_1)$



We don't know which Gaussian generated each point (especially when we haven't even learned the Gaussian's parameters yet). So we'll introduce the latent variables $\{z_i\}_{i=1}^n$

Note: We didn't derive this. It comes from maximizing a tight lower-bound of the log-likelihood. See the textbook if you are curious.

The Expectation Maximization (EM) Algorithm for GMM

Initialize: probabilities of being in each Gaussian $\theta_1, \dots, \theta_k$ all to 1/k
means of the Gaussians $\mu_1, \dots, \mu_k$ to random points
covariances of the Gaussians $\Sigma_1, \dots, \Sigma_k$ to identity matrices

E-Step: Compute fractional assignment of point $\boldsymbol{i}$ coming from class $\boldsymbol{c}$

$$P(z_i = c \mid \boldsymbol{x}_i) \propto P(z_i = c)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_c, \Sigma_c) \quad \text{denote this} \quad p_{c|x_i}$$

Normalize these probabilities such that $\sum_c p_{c|x_i} = 1$

M-Step: Update the parameters based on the current fractional assignments

$$\theta_c^* = \frac{\sum_i p_{c|x_i}}{n}$$

Fraction of mass assigned to c

$$\boldsymbol{\mu}_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} \boldsymbol{x}_i$$

Weighted mean of fractional points assigned to c

$$\Sigma_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T$$

Weighted covariance of fractional points assigned to c

An Example GMM

Iteration 20:

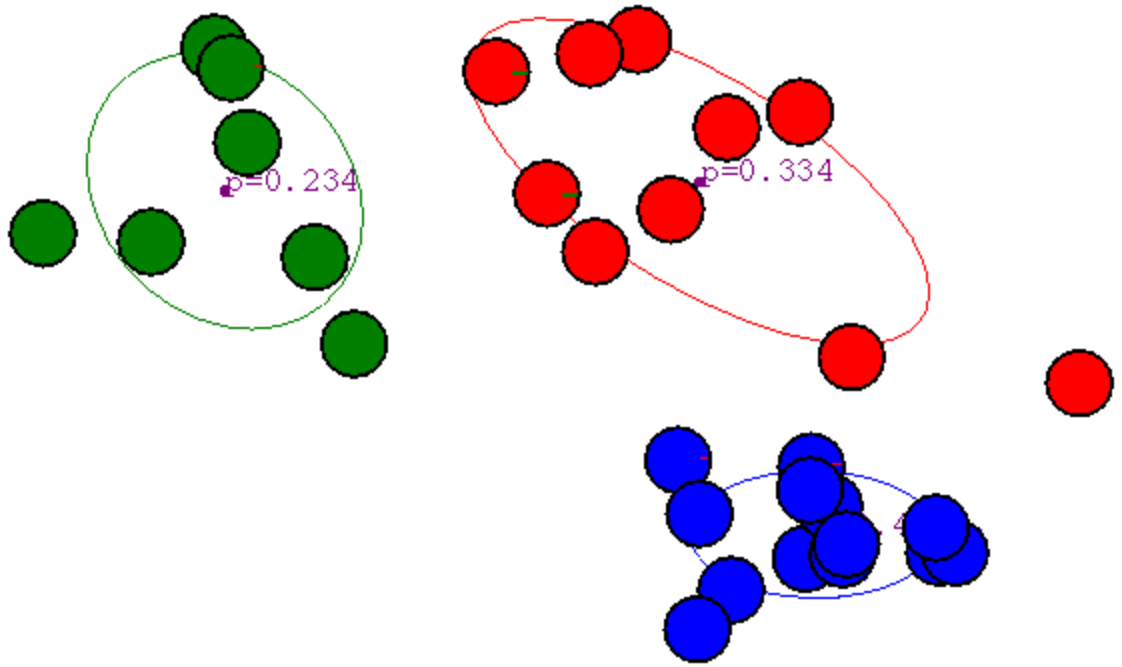
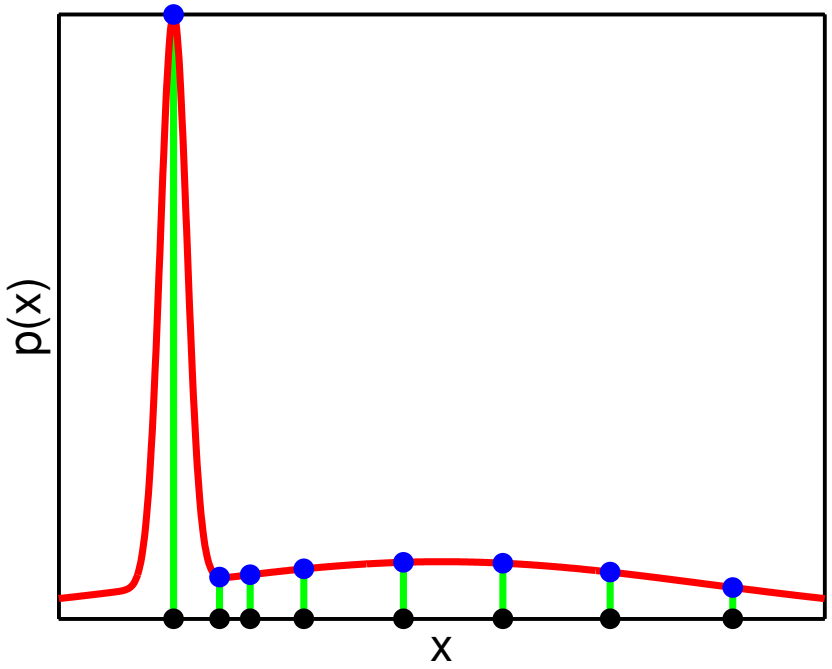


Figure credit Carlos Guestrin

Properties: Singularities in GMM

Log-likelihood can go to infinity if one of the Gaussians "collapses". Assume one of the Gaussians has only one point assigned and the covariance heads towards zero -- $\mu = x$ and $|\Sigma| \to 0$



$$\mathcal{N}(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}$$

$$\lim_{|\Sigma|\to 0} \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} = \infty$$

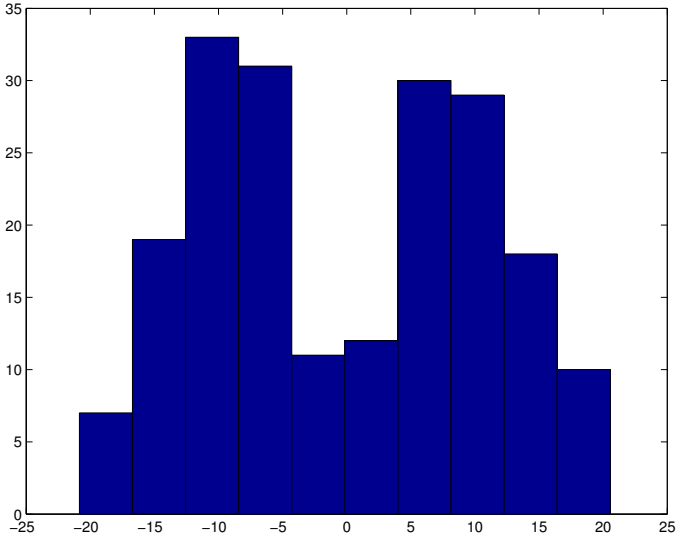
How to deal with this? Monitor each component and reset it to something random if it starts collapsing (random value, large covariance). Or add a prior to the covariance and do MAP in the M-Step of the EM algorithm.

Figure credit Chris Bishop from "Pattern Recognition and Machine Learning" book.

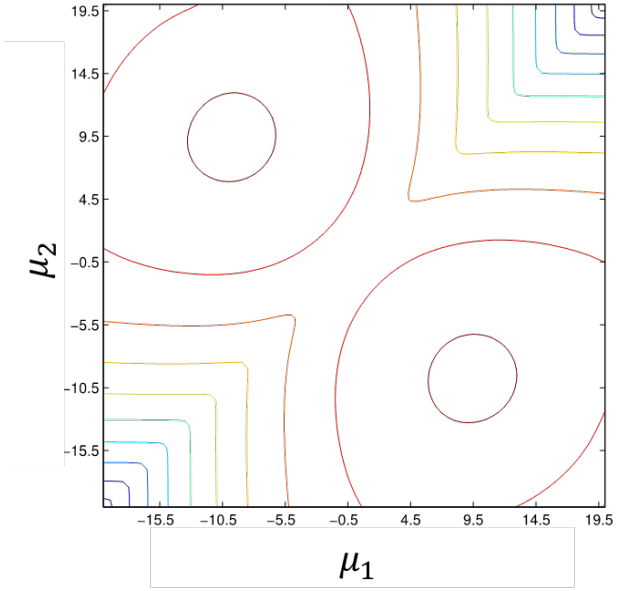
Properties: Identifiability Issues

The log-likelihood in Gaussian Mixture Models has multiple identical maxima. Simple consider swapping the parameters between different models to see why.

Histogram of Data



Log-Likelihood for Different Means of a 2-component GMM



No easy fix. Also limited harm, part of the reason convergence may be slow.

Figure credit Kevin Murphy from "Machine Learning: A Probabilistic Approach" book.

Behavior of Expectation Maximization for GMM

It is guaranteed to converge in finitely many steps:

- Proof looks like the k-Means version but harder – shows that log-likelihood must increase or remain the same between iterations

- In practice, it may converge slowly. Can stop early if no progress is made on log-likelihood for a long time.

Not guaranteed to converge to the global optima:

- Like k-Means, do multiple restarts and then choose the one with highest log likelihood.

- Has a couple of "divergent" solutions

Note: Expectation Maximization (EM) is a whole family of algorithms for dealing with hidden/latent variables. Not just used in Gaussian Mixture Models.

Summary of Gaussian Mixture Models

Assumes data is generated from *k* independent Gaussians:

- Can model more complex configurations than k-Means but is slightly more costly and difficult to implement.

- Produces a full density model of the data → enables you to sample new synthetic data or evaluate the probability of some new point.
  - More on density estimation next class.

- Fractional assignments can be turned into hard clusterings by taking the argmax for each point.

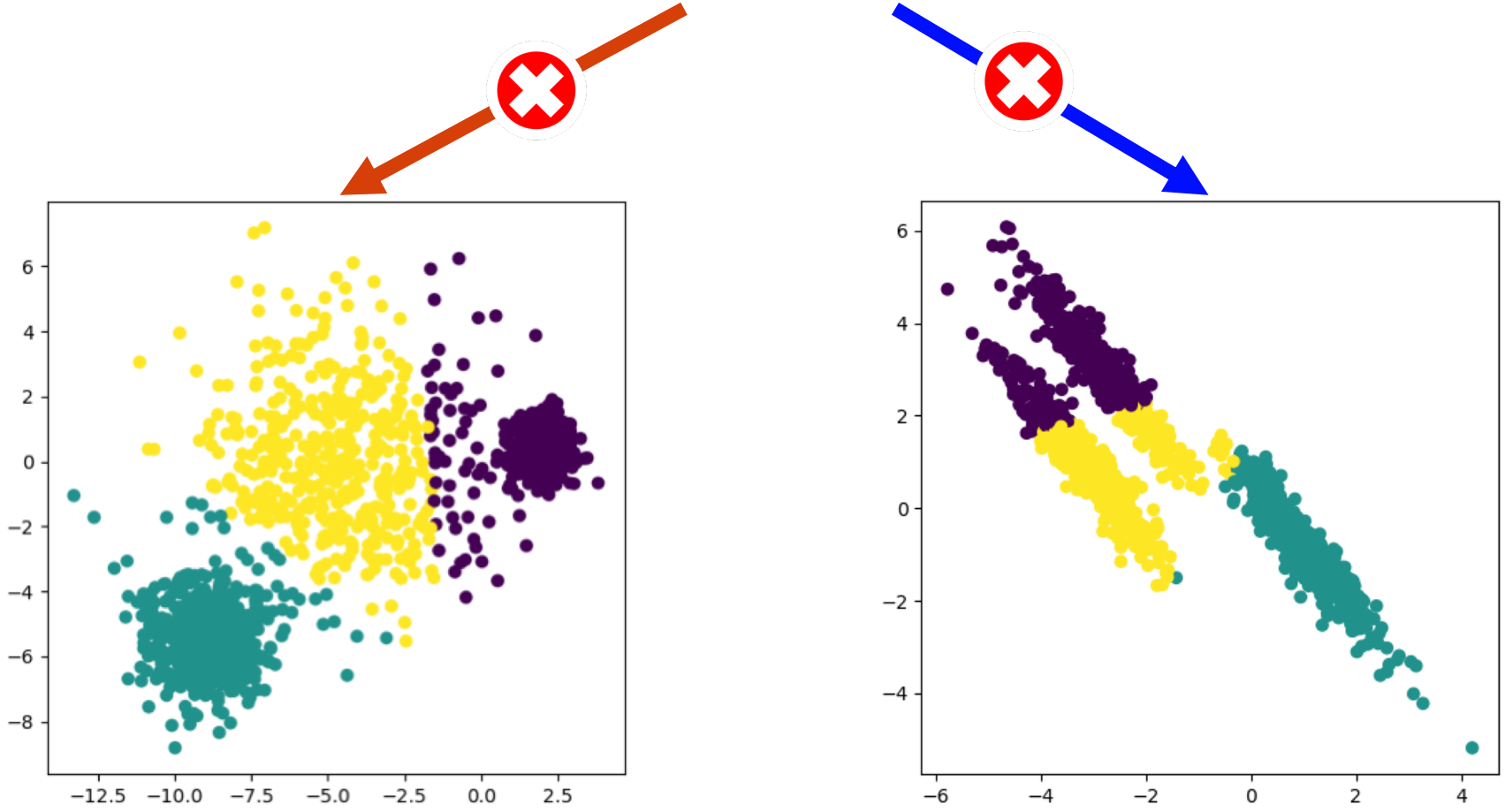
Uses the expectation maximization algorithm for optimization:

- May be slow to converge or end up in trivial optima.
- May need multiple restarts.

What can Gaussian Mixture Models Tell Us About k-Means?

How to get k-Means back from GMMs:

- Assume hard-assignment rather than fractional.
- Assume all Gaussians have the **same isotropic** covariance.



Figures adapted from Phil Roth

Questions Someone Might Ask You About kMeans and GMM

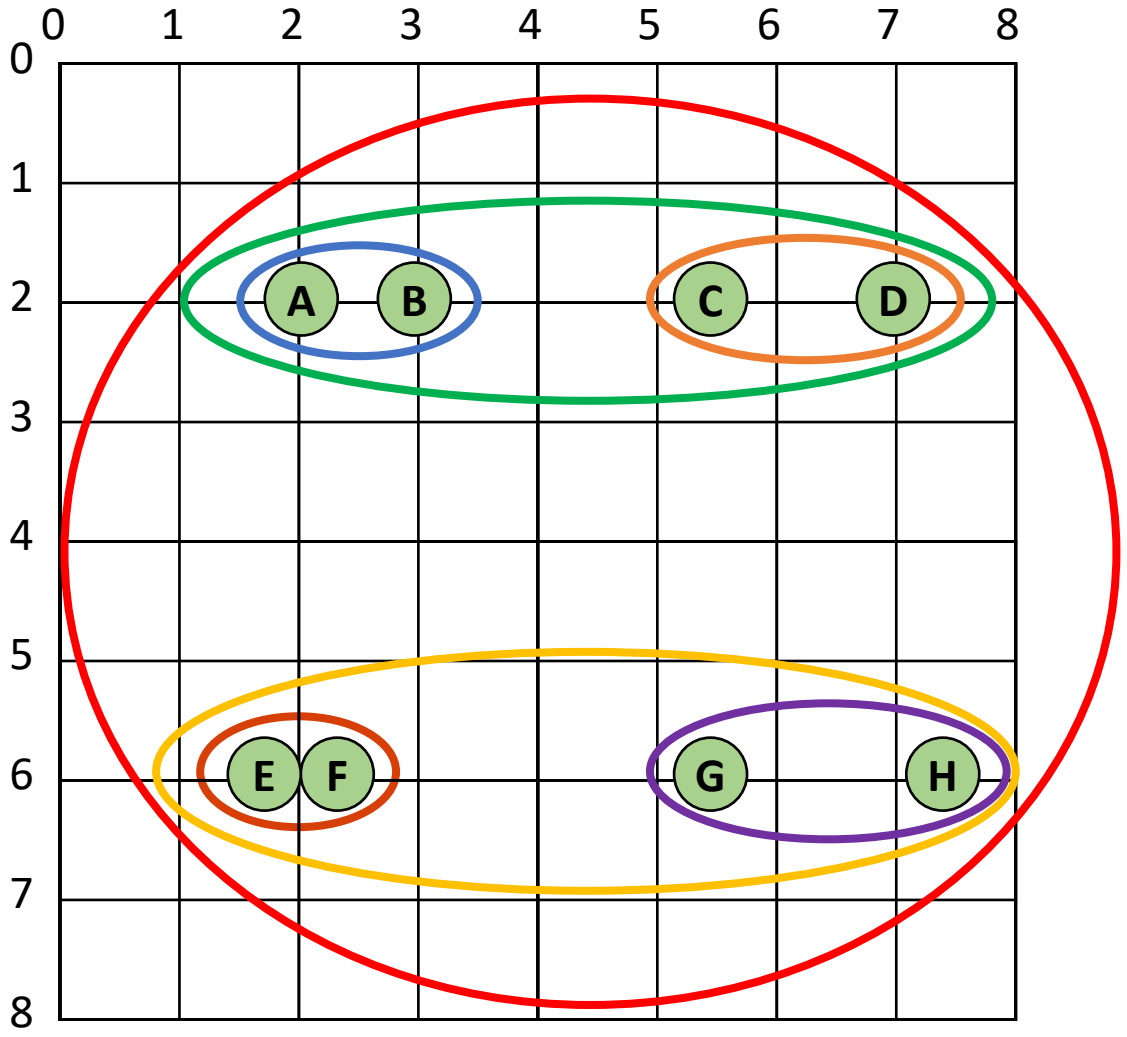
K-means: What is clustering? What is k-means? How is k-means implemented? What does k-means optimize? How is coordinate descent related to the k-means algorithm? Is the k-means algorithm guaranteed to converge? If so, to local or global optima? How do we pick hyperparameters for k-means? Is k-means sensitive to outliers? Is k-means sensitive to initialization? What types of clusters does k-means work best for?

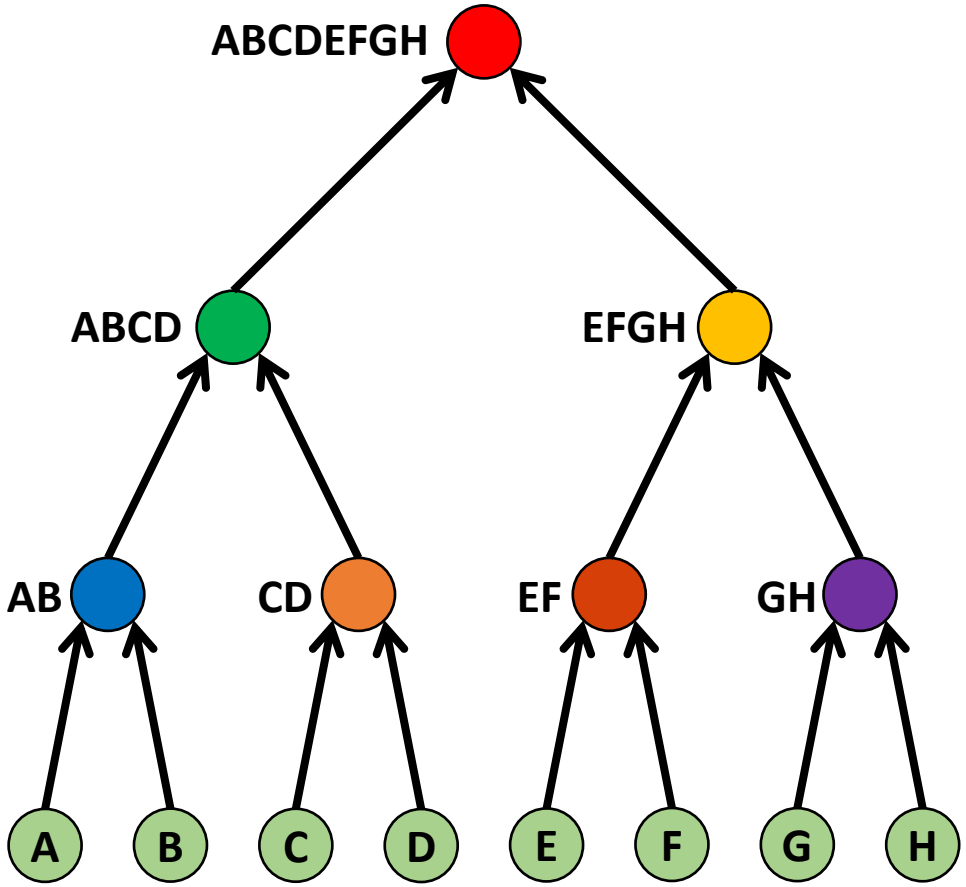
Gaussian Mixture Models: What sort of model does GMM assume generated the data? What can GMM do that k-means can't? Why is maximum marginal likelihood difficult to optimize? What is the Expectation Maximization algorithm do at a broad level? What are some challenges in GMM optimization? What assumptions must be made in GMM in order to recover the k-means algorithm?

Topic Area:

Clustering: HAC

Hierarchical Agglomerative Clustering (HAC)

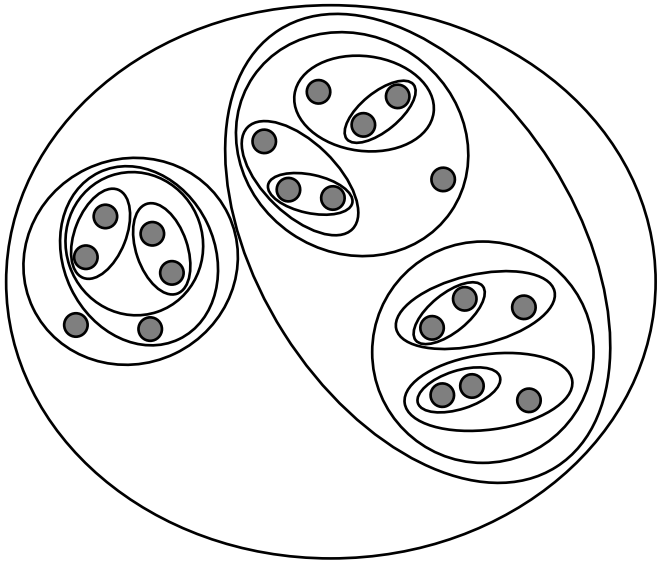




Hierarchical Agglomerative Clustering (HAC)

Given a dataset $X = \{\boldsymbol{x}_i\}_{i=1}^{n}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and some distance function $d(x_i, x_j)$ which measures the distance between two points:

1. Initialize every datapoint as its own cluster
2. Until there is only one cluster remaining:
    a) Merge the two closest clusters



Notice it doesn't output a specific number of clusters. Can save intermediate clusterings from k=n to k=1.

Question: We have a measure of distance between points, how do we use it to measure "closeness" of clusters?

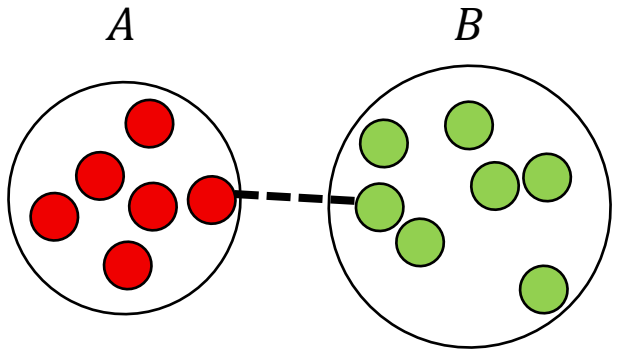
Hierarchical Agglomerative Clustering (HAC)

Consider two clusters A and B, consider the following distance measures $l(A, B)$ defined based on a point-wise distance function $d(x, y)$:

Single-link

- The distance between the nearest points

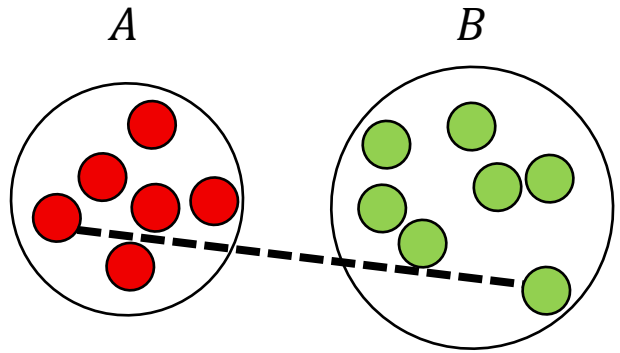
$$l(A, B) = \min_{x_a \in A, x_b \in B} d(x_a, x_b)$$



Complete-link

- The distance between the furthest points

$$l(A, B) = \max_{x_a \in A, x_b \in B} d(x_a, x_b)$$



Slide adapted from Xiaoli Fern

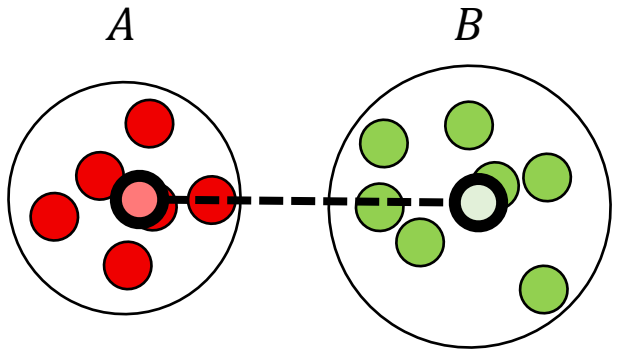
Hierarchical Agglomerative Clustering (HAC)

Consider two clusters A and B, consider the following distance measures $l(A, B)$ defined based on a point-wise distance function $d(x, y)$:

Centroid

- The distance between the cluster means

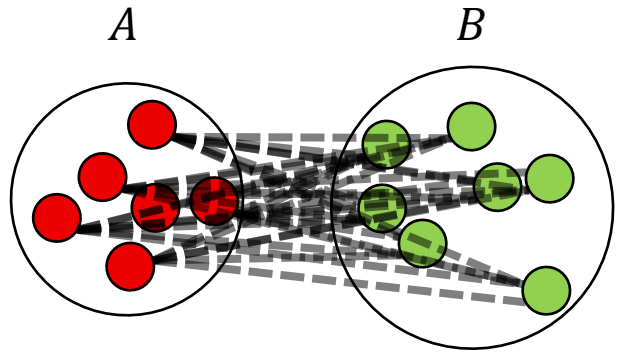
$$l(A, B) = d\left(\overline{x_a}, \overline{x_b}\right)$$



Average-link

- Average distance between all cross-cluster pairs

$$l(A, B) = \frac{1}{|A||B|} \sum_{x_a \in A} \sum_{x_b \in B} d(x_a, x_b)$$



- **Most robust and most commonly used**

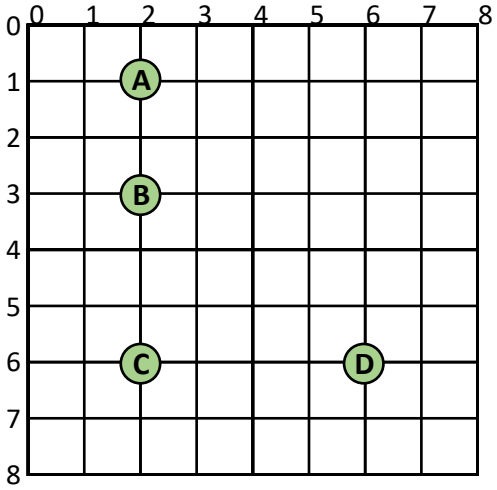
Slide adapted from Xiaoli Fern

Single-Link Method Example (L1 distance for simplicity)

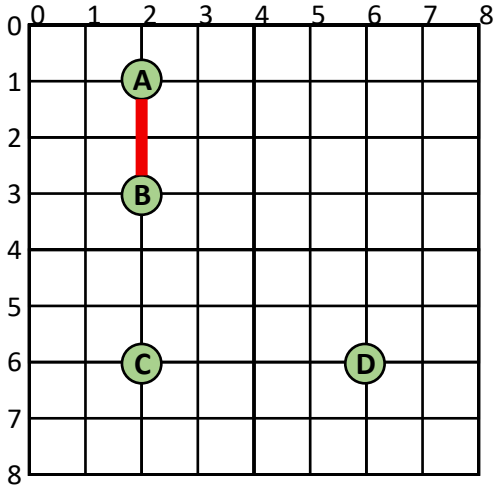
Single-link

- The distance between the nearest points

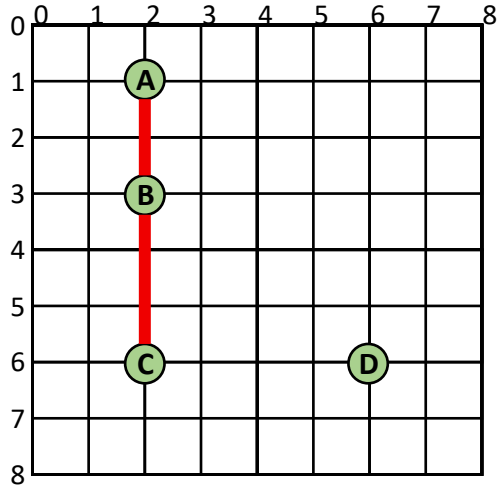
$$d(A,B) = \min_{x_a \in A, x_b \in B} d(x_a, x_b)$$



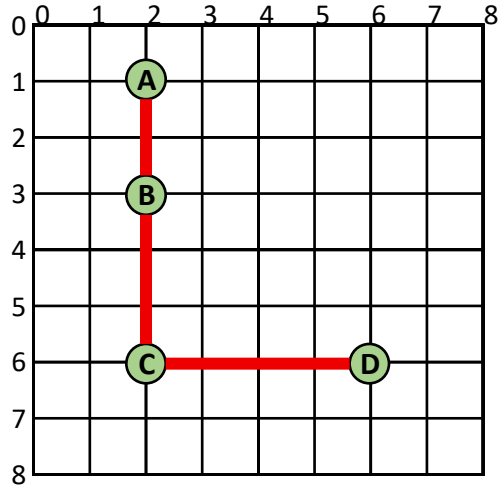
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-



	AB	C	D
AB	-	3	7
C		-	4
D			-



	ABC	D
ABC	-	4
D		-



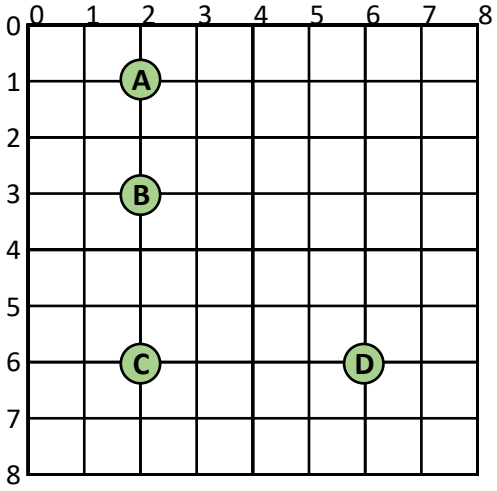
Slide adapted from Xiaoli Fern

Complete-Link Method Example (L1 distance)

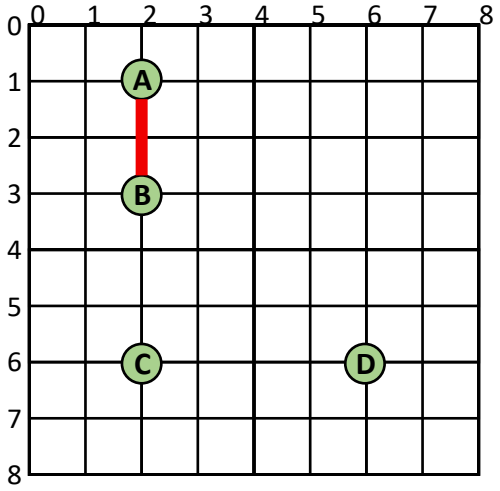
Complete-link

- The distance between the furthest points

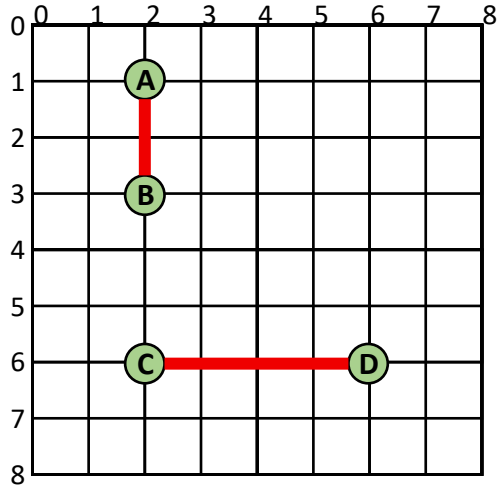
$$d(A,B) = \max_{x_a \in A, x_b \in B} d(x_a, x_b)$$



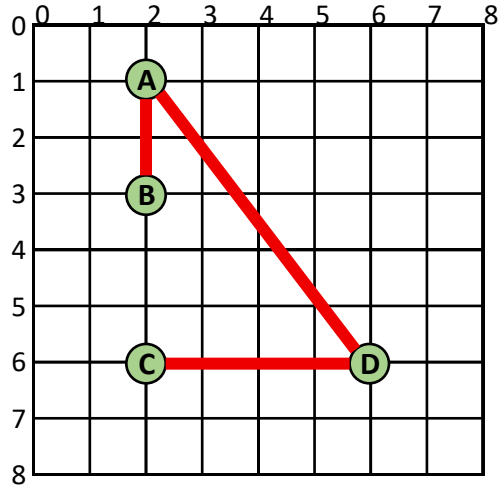
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-



	AB	C	D
AB	-	5	9
C		-	4
D			-



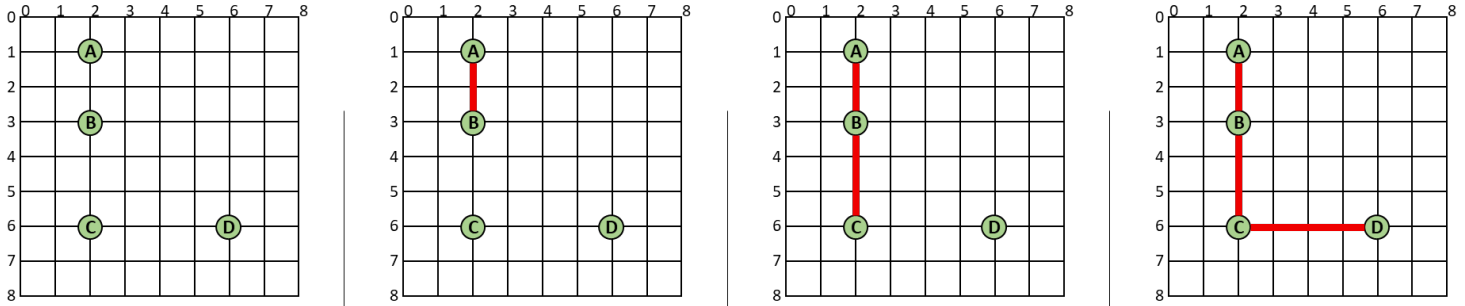
	AB	CD
AB	-	9
CD		-



Slide adapted from Xiaoli Fern

Visualizing Hierarchical Clusterings: Dendrogram

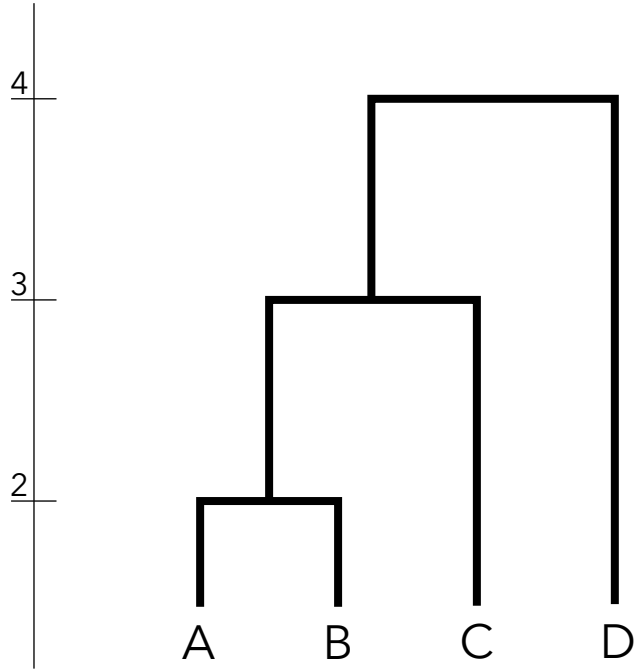
- Height of joint is the distance between the two merged clusters.
- Merge distance monotonically increased as we merge more for single / complete / average linking (not for centroid)



	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-

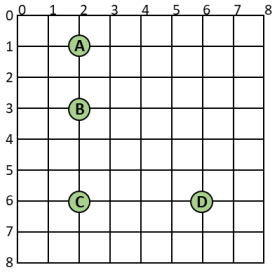
	AB	C	D
AB	-	3	7
C		-	4
D			-

	ABC	D
ABC	-	4
D		-

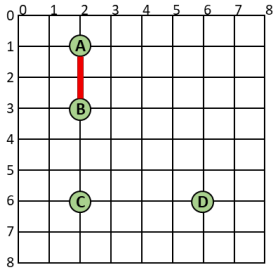


Slide adapted from Xiaoli Fern

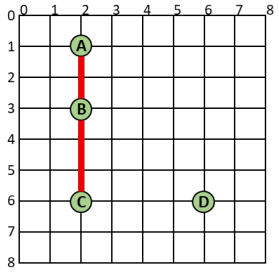
Visualizing Hierarchical Clusterings: Dendrogram



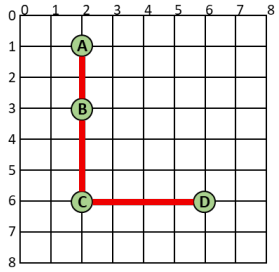
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-

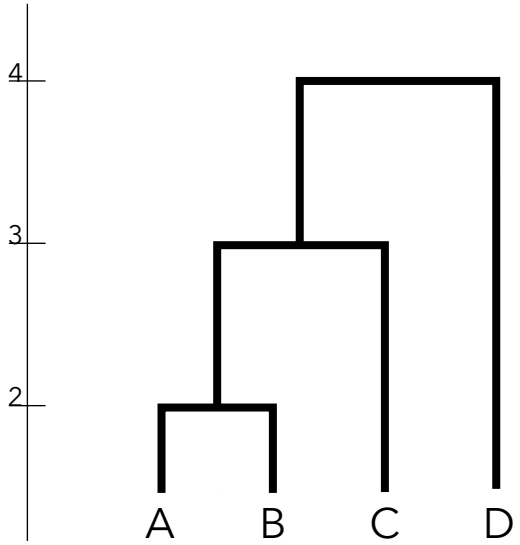


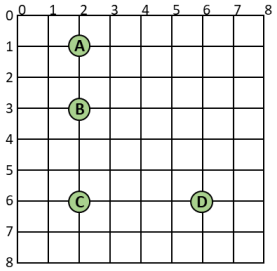
	AB	C	D
AB	-	3	7
C		-	4
D			-



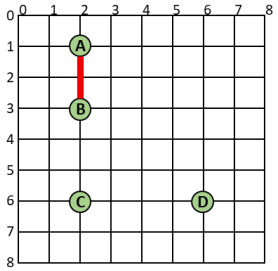
	ABC	D
ABC	-	4
D		-



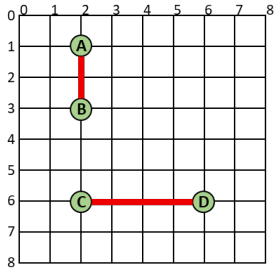




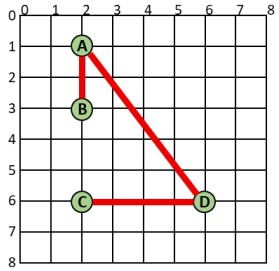
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-

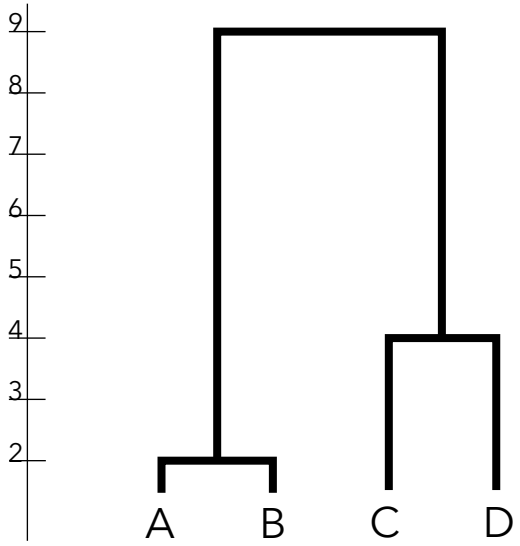


	AB	C	D
AB	-	5	9
C		-	4
D			-



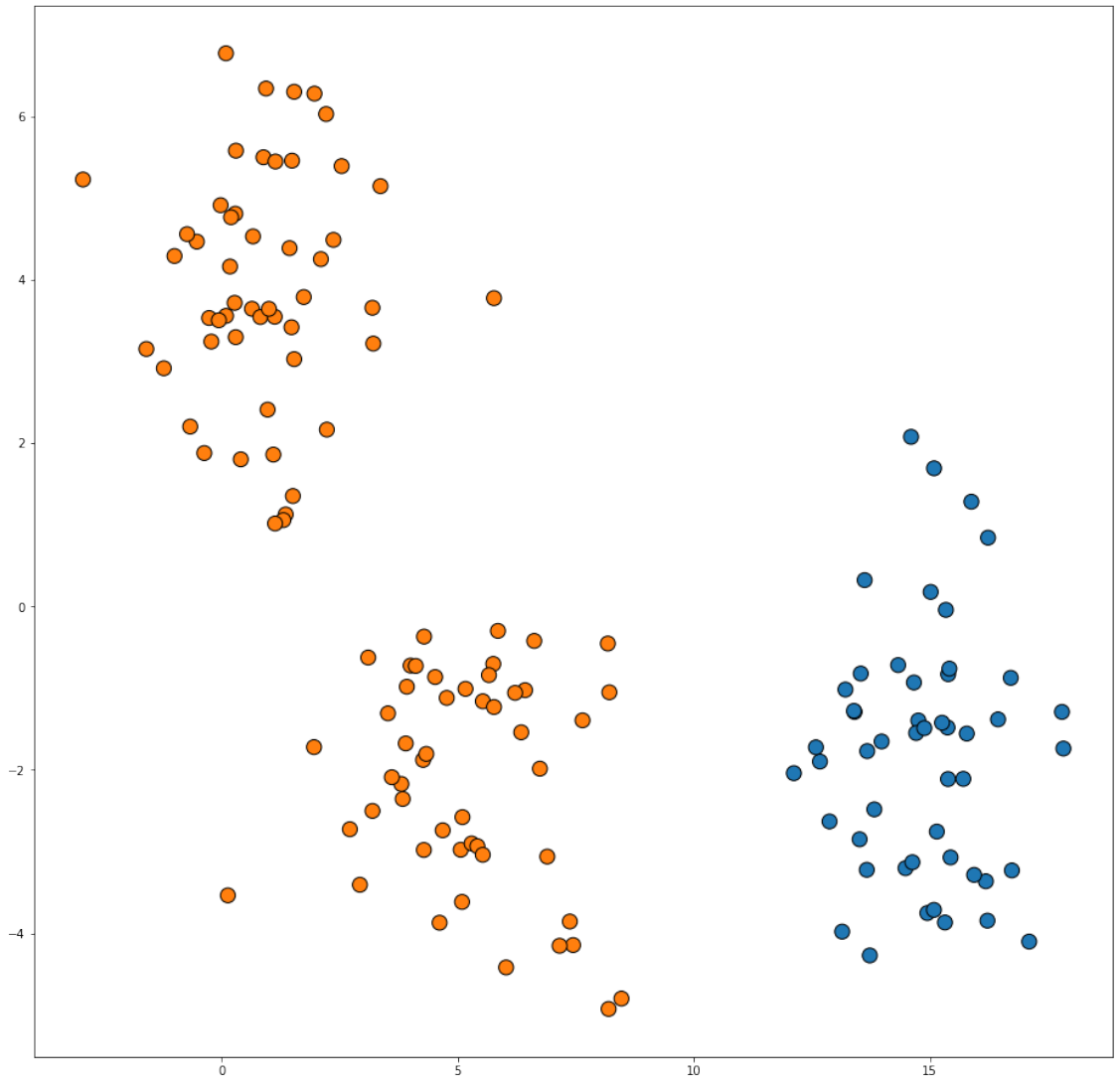
	AB	CD
AB	-	9
CD		-

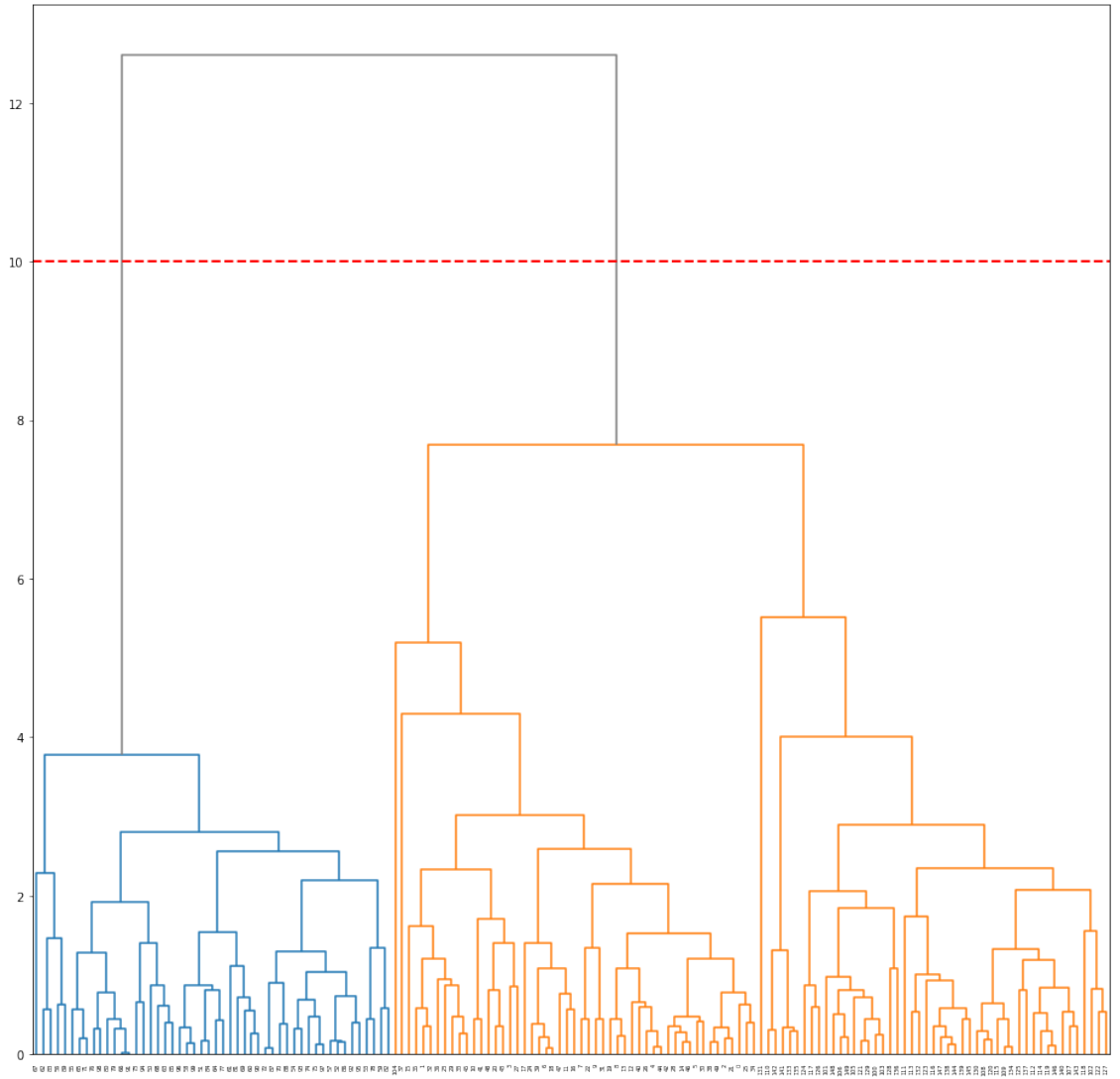




Slide adapted from Xiaoli Fern

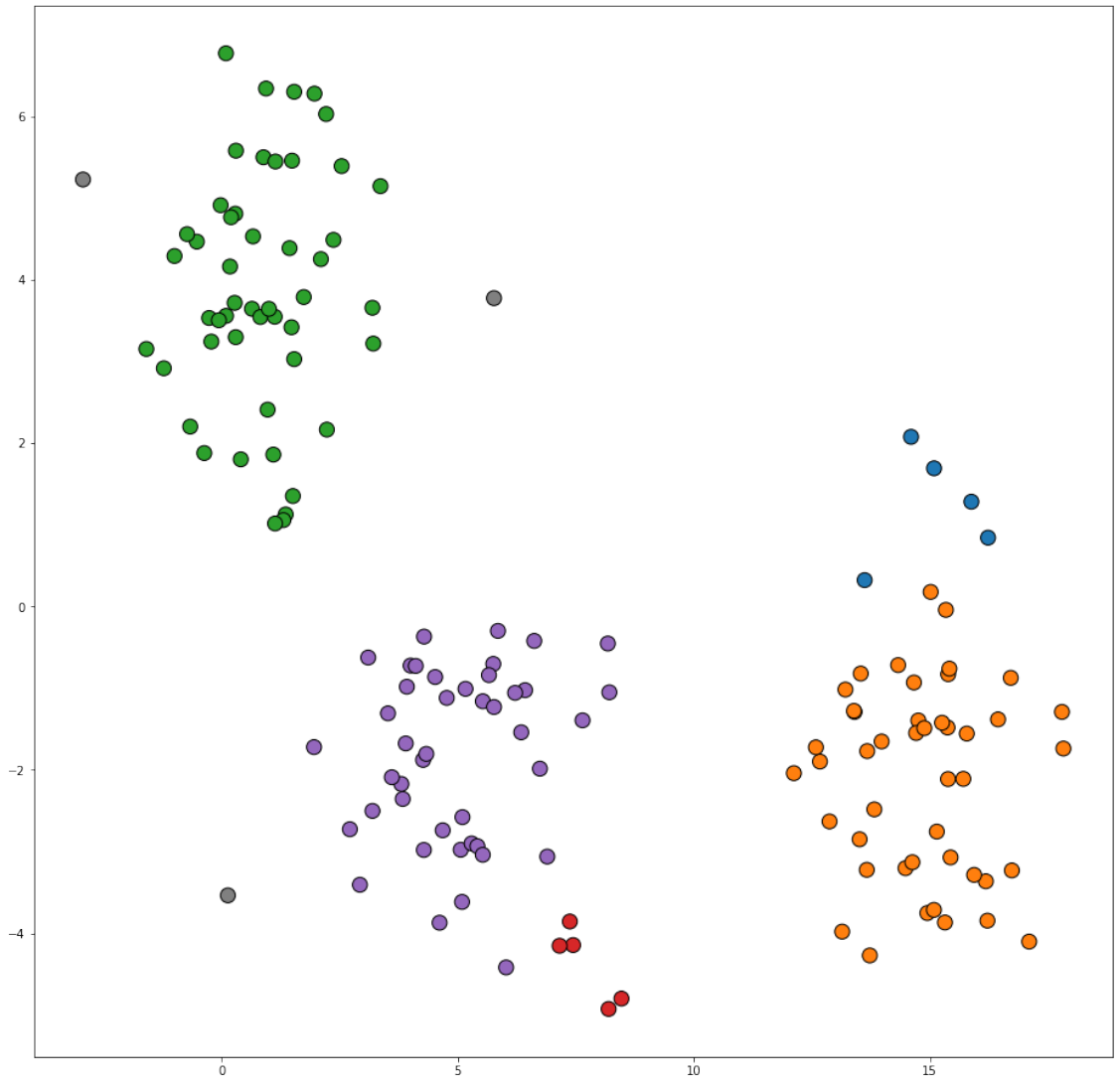
Visualizing Hierarchical Clusterings: Dendrogram

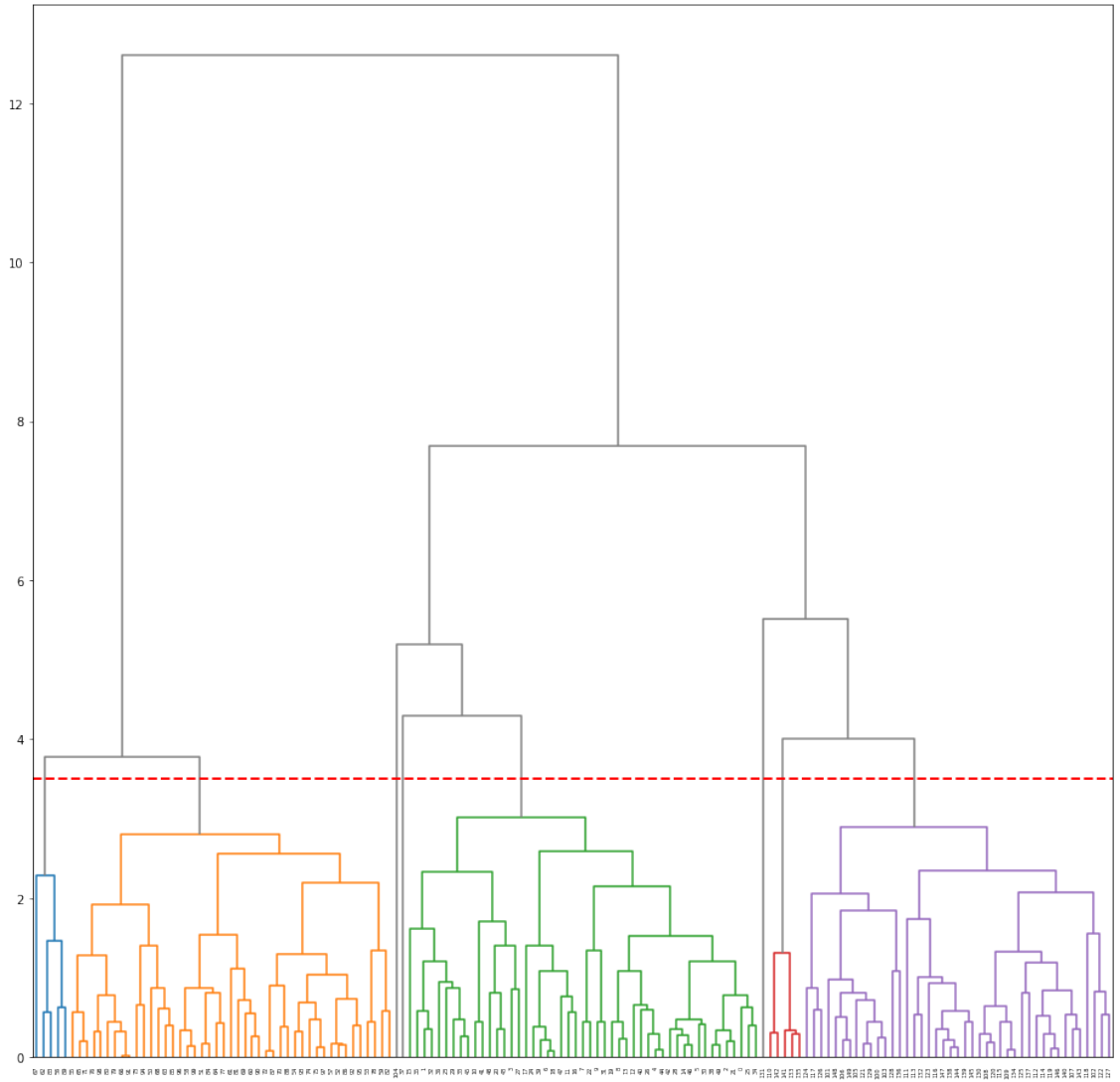




Average-link example

Visualizing Hierarchical Clusterings: Dendrogram



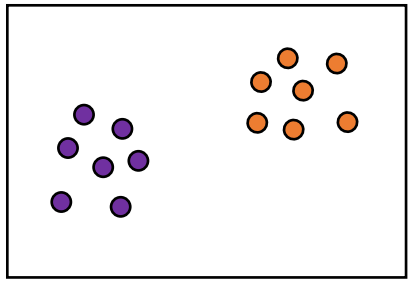


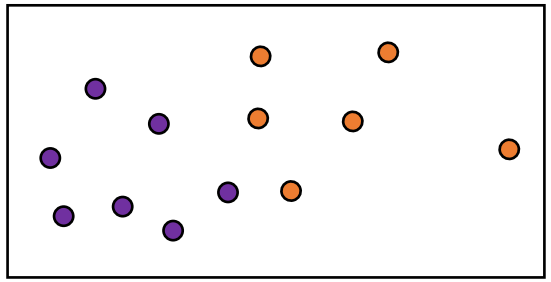
Average-link example

This is all cool, but how do I know if I made a good clustering?

Without external data:

- User inspection – (aka just look at it) Does a cluster seem to have a common theme?
  - CAUTION: HUMANS ARE GOOD AT IMAGINING PATTERNS
- Internal Criterion – measure properties of a clustering presumed to be "good"
  - High within-cluster similarity: $s_w = \sum_{j=1}^{k} \sum_{x,x' \in c_j} sim(x, x')$
  - Low between-cluster similarity: $s_b = \sum_{x \in c_i} \sum_{x' \notin c_i} sim(x, x')$





  - This measure depends on the dataset and measure of distance used.

Slide adapted from Xiaoli Fern

Evaluating Clustering

With external data:

- Suppose someone comes along afterwards and gives you labels for all the points, can evaluate the clustering by how well it separated points with different labels.
- **Rand Index** – Given a clustering P and a ground truth label set G, measure the number of vector pairs that are
  - a: in the same group in both P and G (same cluster, same labels)
  - b: in the same group in P but different in G (same cluster, different labels)
  - c: in different groups in P but same in G (different cluster, same labels)
  - d: in different groups in both P and G (different clusters, different labels)

$$Rand\ Index = \frac{a + d}{a + b + c + d}$$

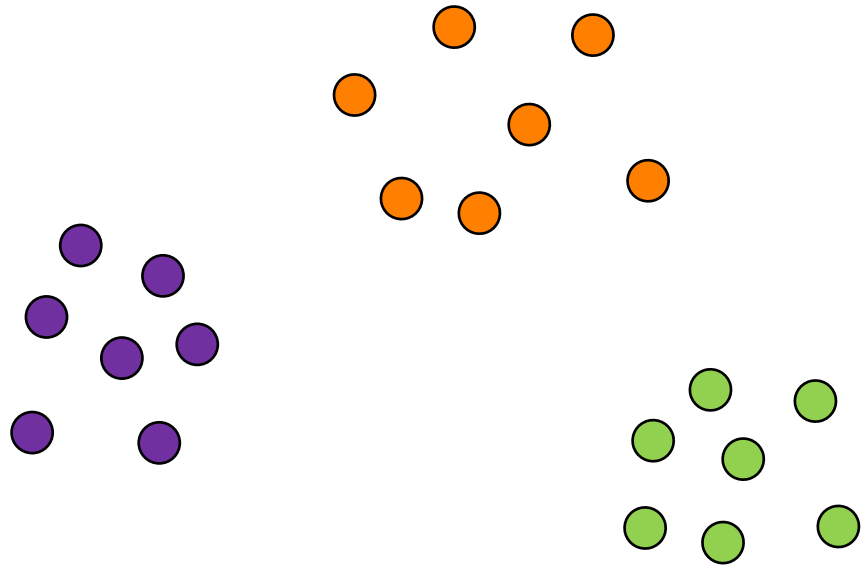
- **Adjusted Rand Index:** correct rand-index by the average rand-index of a random clustering of the data.

Slide adapted from Xiaoli Fern

Evaluating Clustering

With external data:

- Suppose someone comes along afterwards and gives you labels for all the points, can evaluate the clustering by how well it separated points with different labels.
- **Purity** – Fraction of points that would be correctly classified by a "majority vote" per cluster where all points get the label of the majority.

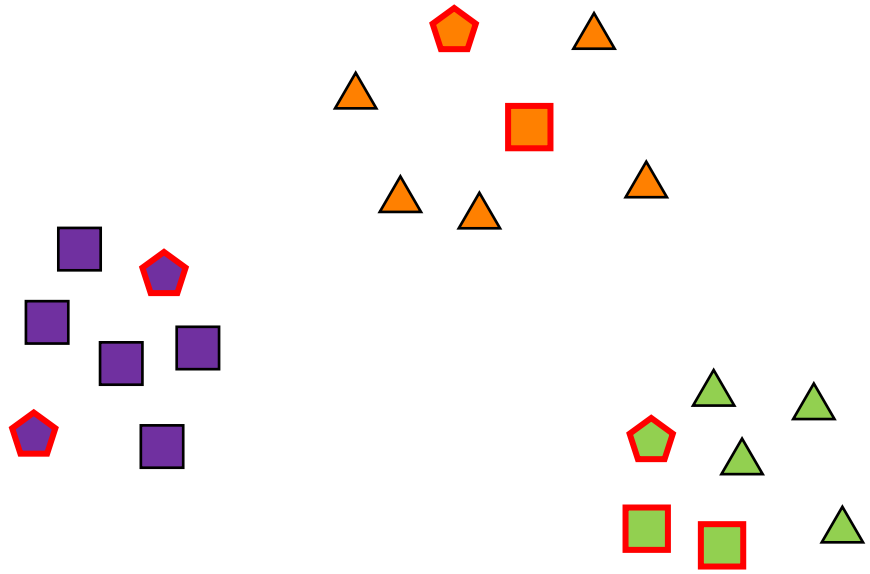


Slide adapted from Xiaoli Fern

Evaluating Clustering

With external data:

- Suppose someone comes along afterwards and gives you labels for all the points, can evaluate the clustering by how well it separated points with different labels.
- **Purity** – Fraction of points that would be correctly classified by a "majority vote" per cluster where all points get the label of the majority.



$$Purity = \frac{5 + 5 + 4}{21}$$

Slide adapted from Xiaoli Fern

Questions Someone Might Ask You About Hierarchical Clustering

Hierarchical Agglomerative Clustering: How does HAC work? How do you measure distance between different clusters? What is a "link function"? How do the different link functions behave? What is a dendrogram? How to read dendrograms? How to create "flat" clusterings from HAC?

Evaluating Clustering: What can we measure without knowing labels? What can we measure if we do know labels? How is the Rand Index computed?

Topic Area:

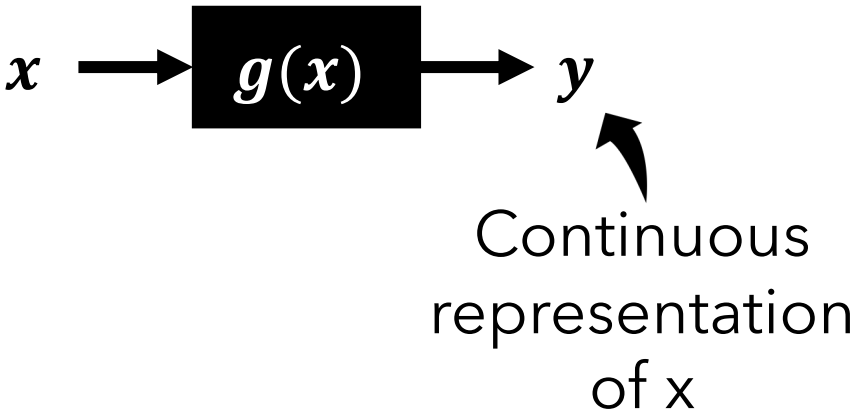
Dimensionality Reduction

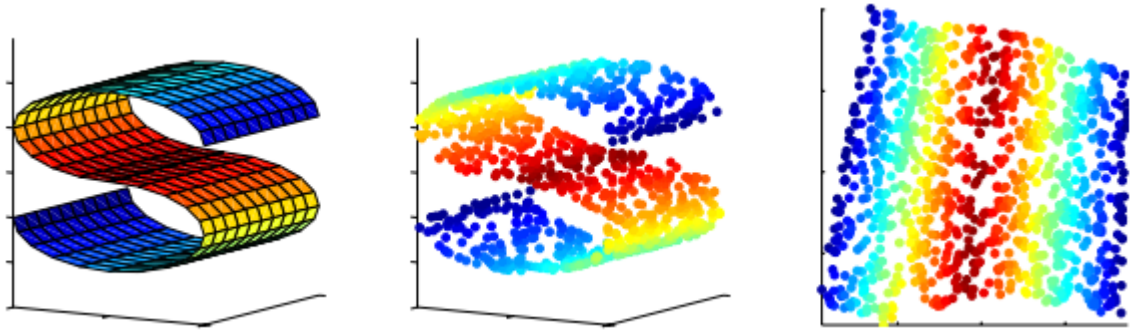
New Topic: Dimensionality Reduction

Unsupervised Learning

Only given a set of input instances (x)

Dimensionality Reduction: Represent high-dim data as low-dim data





Example: Finding a 2D subspace in a 3D feature space

New Topic: Dimensionality Reduction

Dimensionality reduction tries to find a more compact representation of the data -- creating new features with lower dimensionality that still represents the data well.

Why would we want to do this?

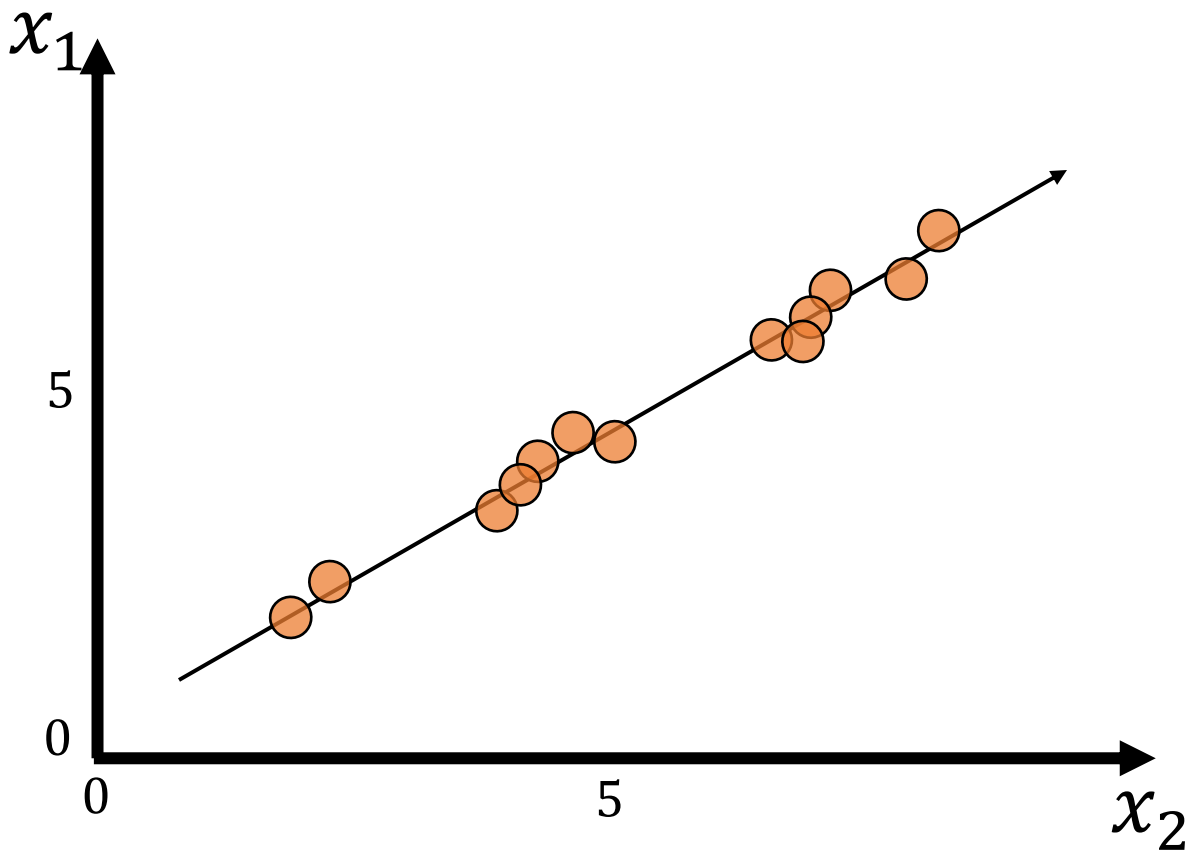
- **Visualization**: We can't plot things in >3 dimensions (and even 3D plots confuse me…). Can we find a 2-3 dimension view of our data that captures the meaningful relationships?

- **Preprocessing:** Many of the learning algorithms we've seen are more computationally expensive with large dimensionality. Further, many of them work better with lower dimensional data.

Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

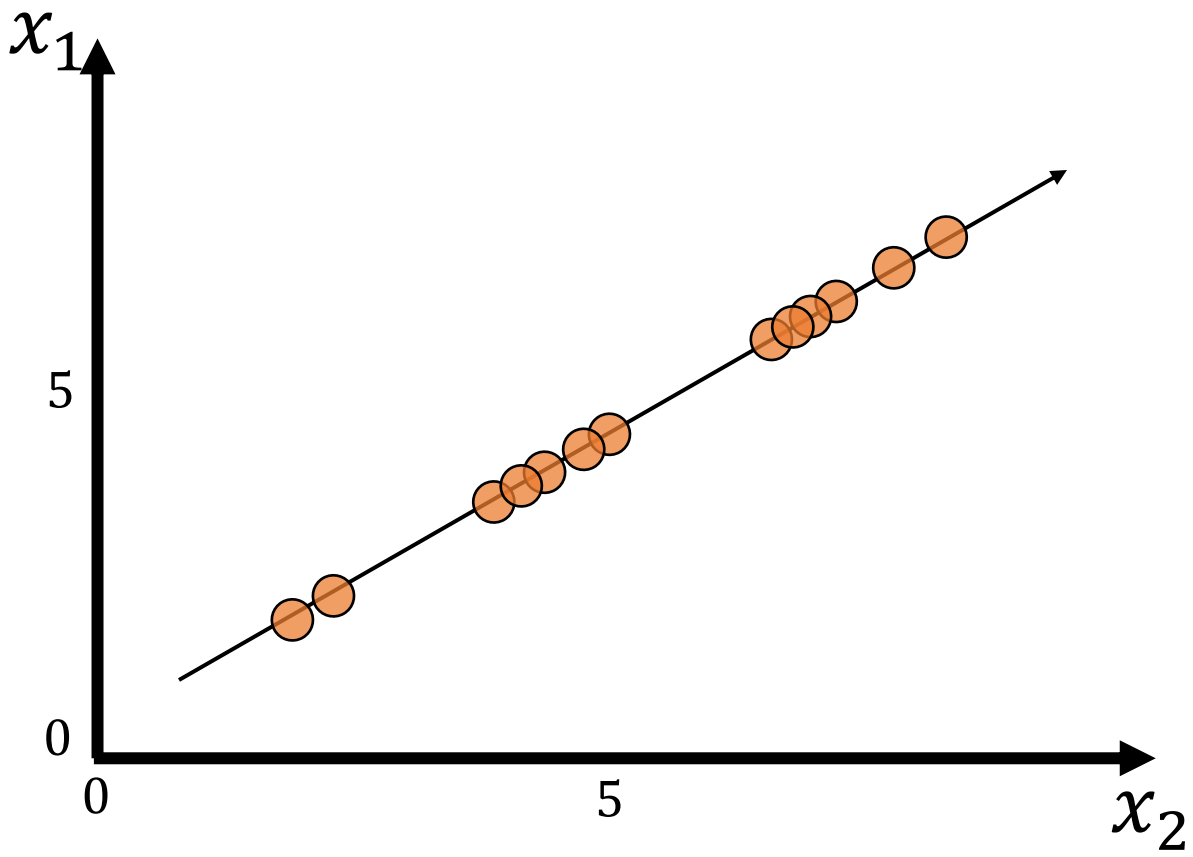
"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"



Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

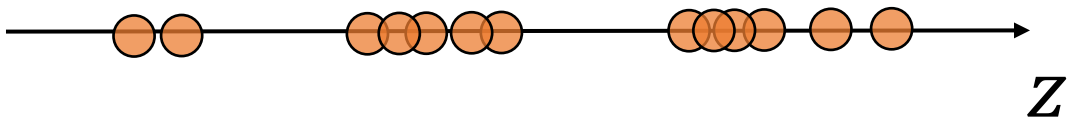
"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"



Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"

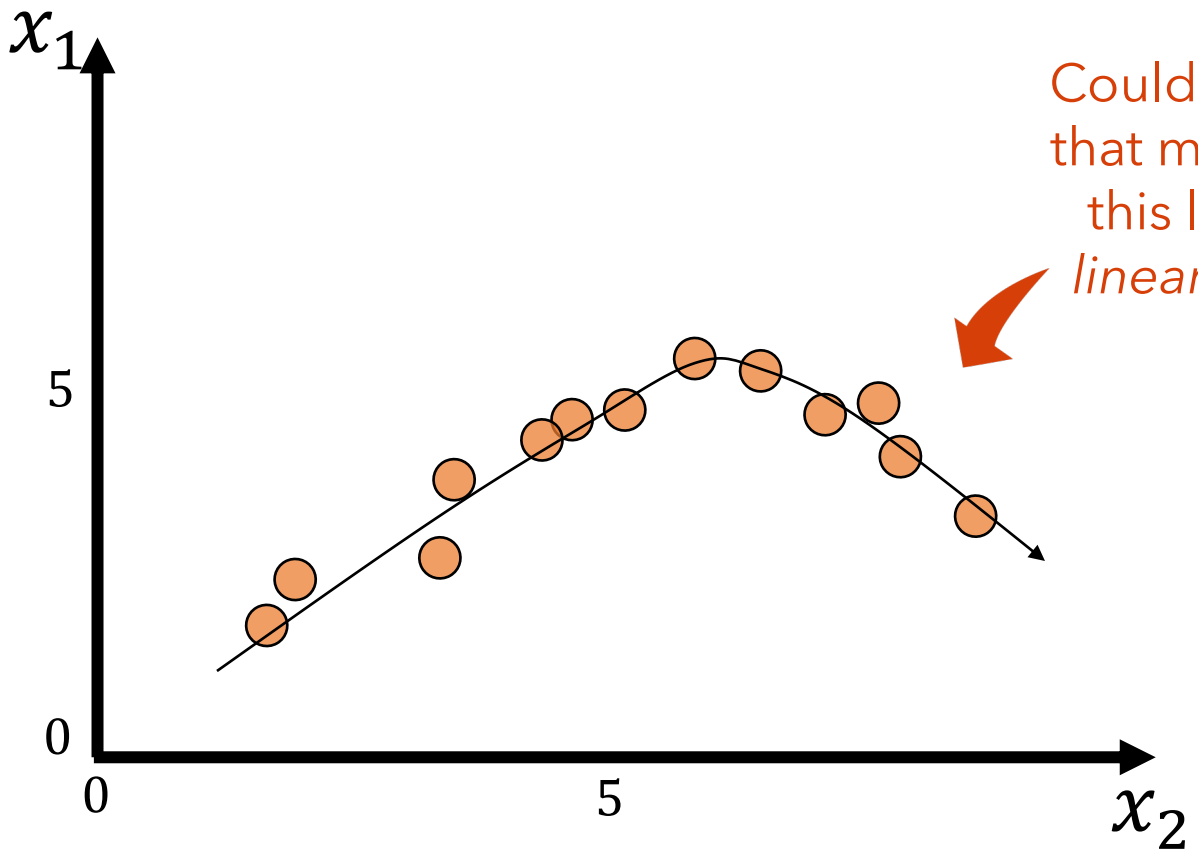


New representation just measures distance along the line we drew

Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"



Could have a new representation that measures the distance along this line – this would be a *non-linear dimensionality reduction.*

Principal Component Analysis (PCA)

A classic dimensionality reduction technique that is widely used

- Finds a **linear** projection from a d-dimensional vector space to a k-dimensional vector space while preserving variation in a dataset (k given):
  - E.g., projecting 4096-dimensional vectors down to 2-dimensional

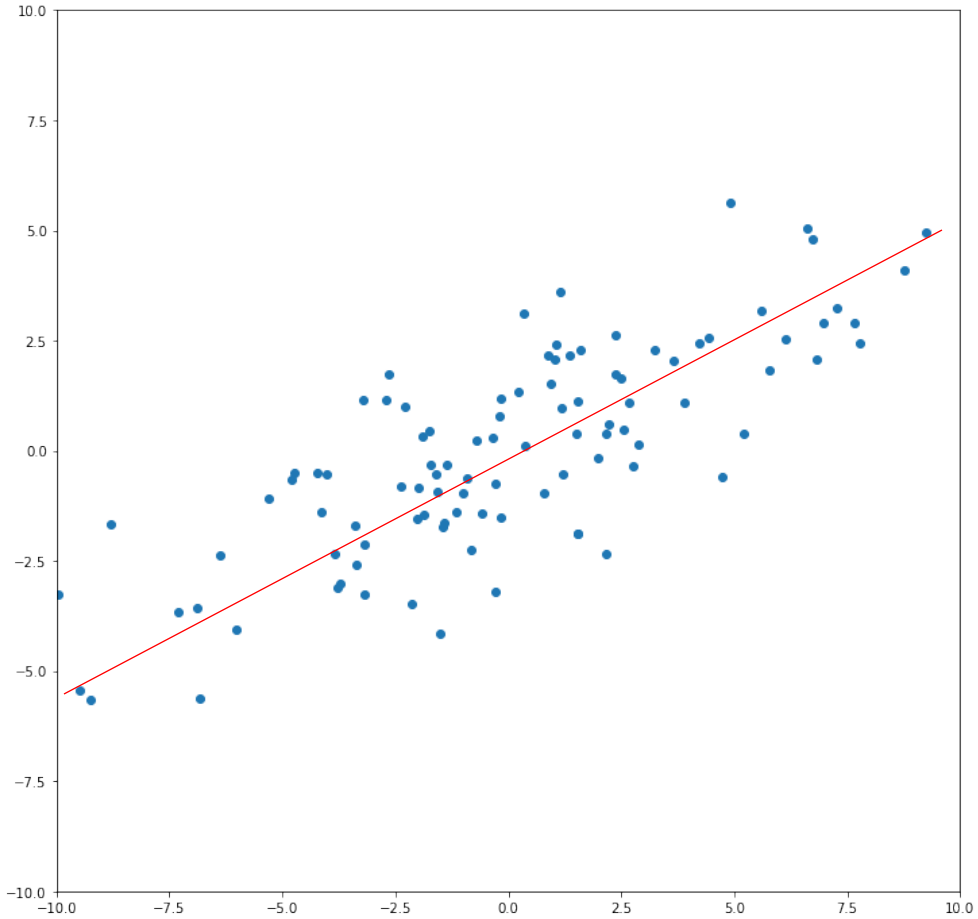
What might it mean to "preserve variation"? What rule were we using before?

- Suppose two features dim-0 and dim-1, but can only keep one:
  - For dim-0, most examples have similar values (small variance)
  - For dim-1, most examples differ (large variance)
- Keep the one with more variance! It explains more about the difference between items

Principal Component Analysis (PCA) -- Conceptually

Consider doing the following:

- Find a line such that most of the variance of the data follows along that line
- Or equivalently, a line where the dataset projected onto that line maintains as much variance.
- Call this line your first principal component.

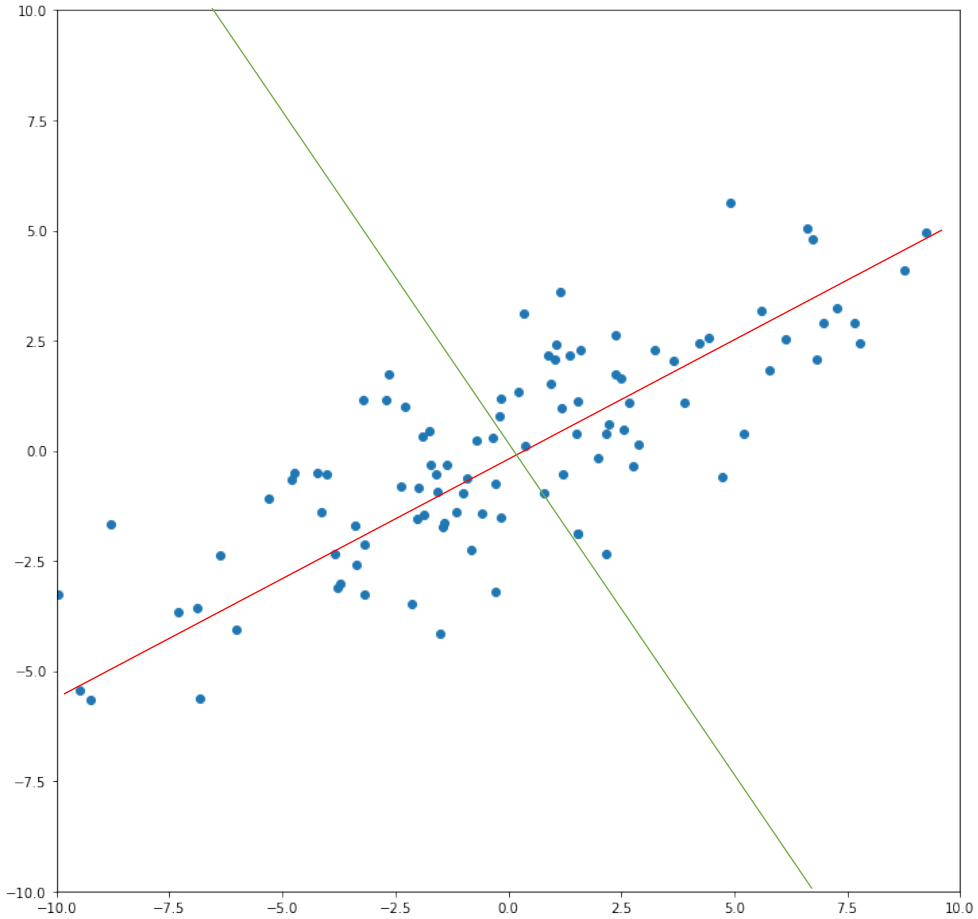


Consider doing the following:

- Repeat this by finding the line orthogonal from this that captures the most variance.
  - 2nd principal component
  - 3rd …
  - 4th …

- In 2D, there is only one line orthogonal to our 1st principal component so we must stop after 2.

- In higher dims, there will be many.



What have we just done?

We wanted to find a line to project our data onto that would preserve as much variance as possible. So to find that linear transform we:

1. Wrote down mathematically what we meant by a linear projection $(w^T x)$
2. Derived an expression for the variance of the data *after* projection $(Var(w^T x) = w^T \Sigma_x w)$
3. Set up an optimization problem to find the projection that maximizes the variance. But then noticed it had trivial solutions, so constrained the representation of the line (must have $w^T w = 1$)
4. Applied the Lagrange Multiplier method $(L(w, \lambda) = w_1^T \Sigma_x w_1 - \lambda_1 (w_1^T w_1 - 1))$
5. Took the derivative and set equal to zero to find all critical points (w's that potentially could maximize our variance). This related to Eigenvectors from linear algebra and can have multiple solutions $(\Sigma_x w = \lambda w)$
6. Observed that the linear transform w that maximizes the variance will be the Eigenvector of Σ_x with the largest Eigenvalue λ. This gives us the first dimension of our PCA!

Plenty of computational packages can find Eigenvectors of matrices.

Procedural View of PCA

Say you have data that is 100 dimensional and you want it down to 3.

```
# X is an n x d data matrix
X = X - np.mean(X,axis=0)
cov = X.T @ X / n
eigvals, eigvecs = np.linalg.eig(cov)
```

To perform PCA on your data matrix X (n x 100):

1. Compute the mean vector of your data (100 dim vector) and subtract it from X to center your date

2. Compute the covariance matrix by computing $\frac{1}{\mathrm{n}} X^T X$

3. Call a package to compute the Eigenvalues/Eigenvectors of the covariance.

4. Sort both in descending order by the Eigenvalues and return Eigenvectors corresponding to the top k. Usually as a (d x k) matrix W with each column being one Eigenvector.

To project your data to that lower dimension, simply compute the matrix product XW

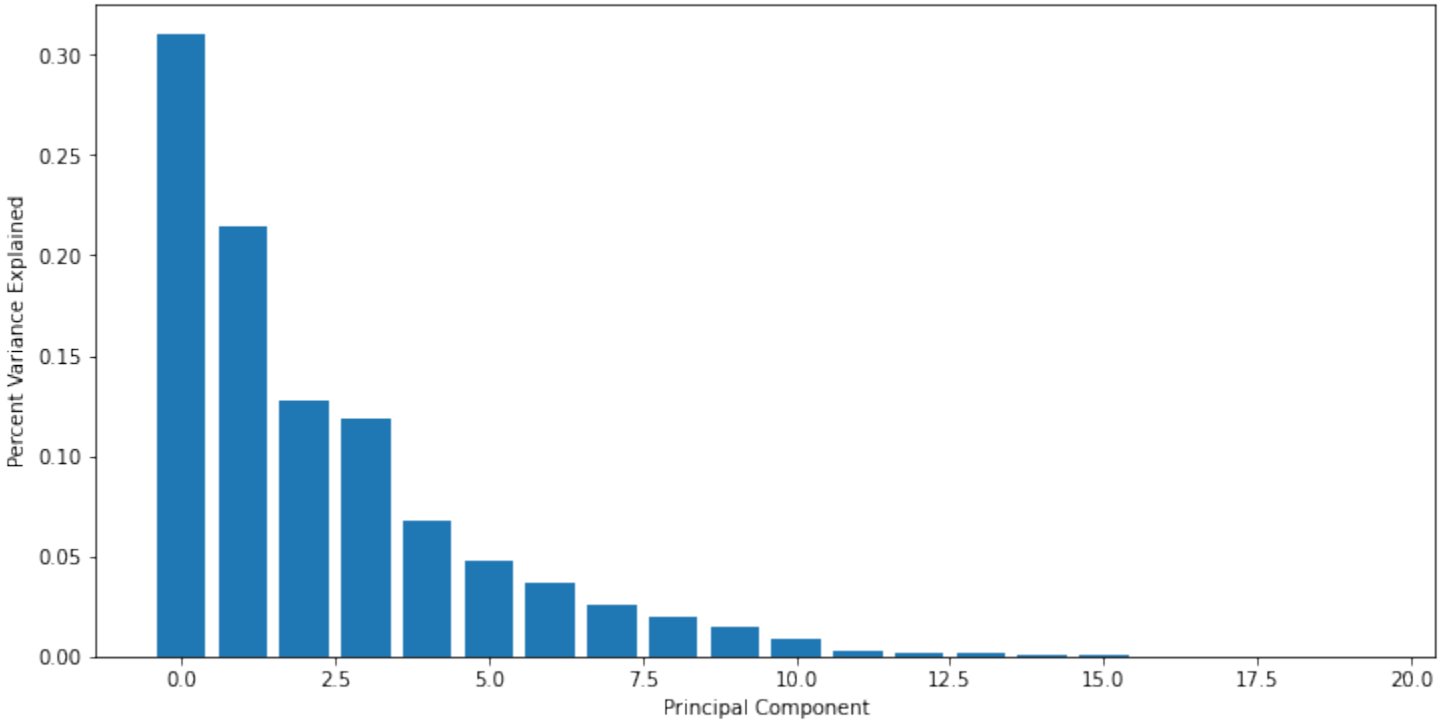
After this class is over: Just call a PCA API, they do basically the same thing.

Choosing How Far To Reduce Dimensionality

For some tasks like visualization, the dimensionality you are projecting to is fixed.

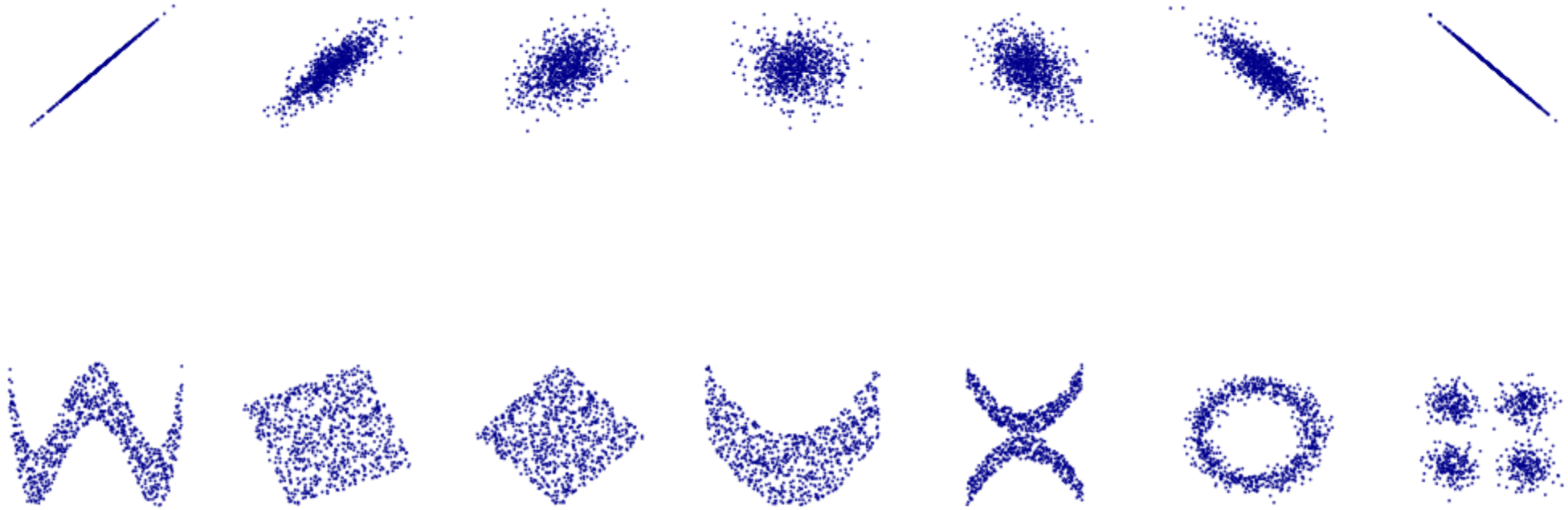
- Use fraction of variance explained to judge how representative the visualization is

If reducing dimension for other reasons, can look at plot of Eigenvalues / Sum Eigenvalues to judge what fraction of variance has been captured for a certain dimensionality:



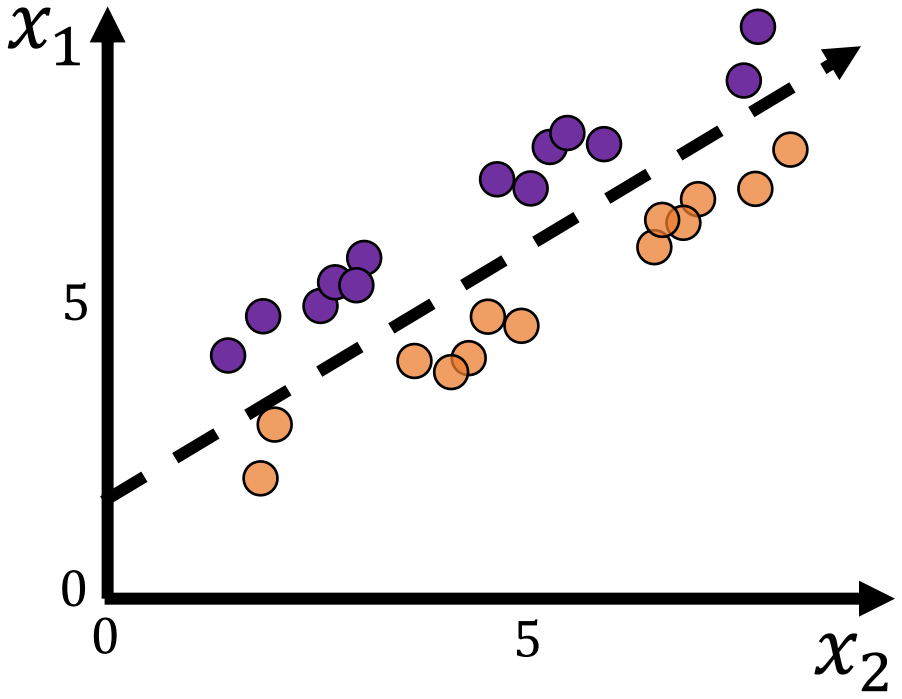
PCA is a Linear Dimensionality Reduction Model

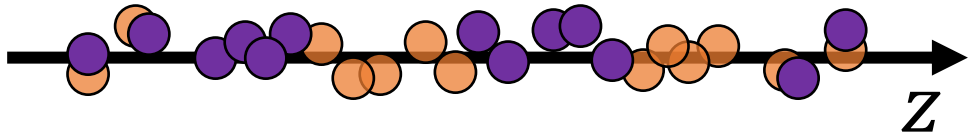
Only works at finding linear subspace…



PCA Doesn't Care About Labels

Direction of maximum variance may not be useful for classification:





See Linear Discriminant Analysis (LDA) if you have labels and want to do linear dimensionality reduction that tries to account for this.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Idea: For each **d-dimensional** input vector $\boldsymbol{x_i}$, directly optimize a **k-dimensional** vector z_i such that **spatial relationships between your high dimensional datapoints are preserved between the lower dimensional ones.**

How to operationalize this thought?

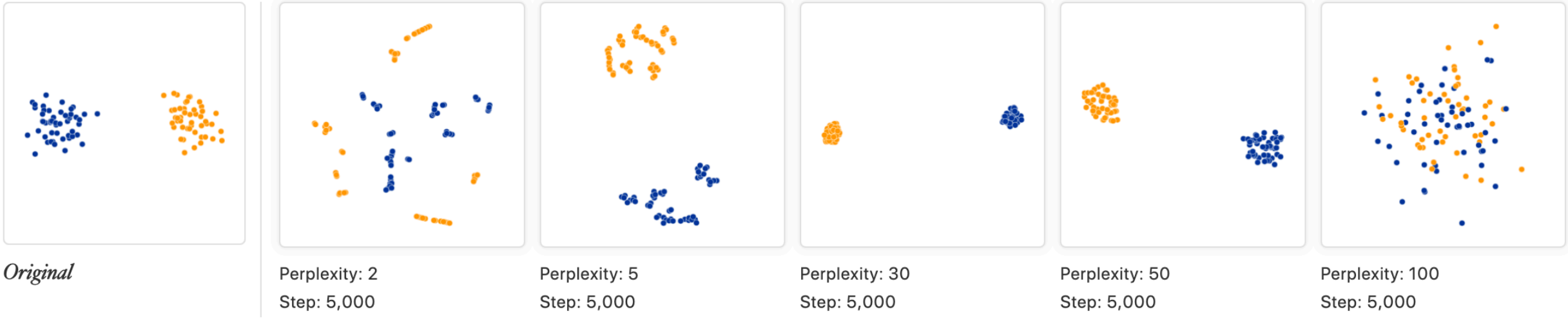
One path: The "neighborness" between points should stay the same.

We can start here with **Stochastic Neighbor Embeddings.**

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

Hyperparameters really matter.



Original

Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000

Perplexity: 30
Step: 5,000

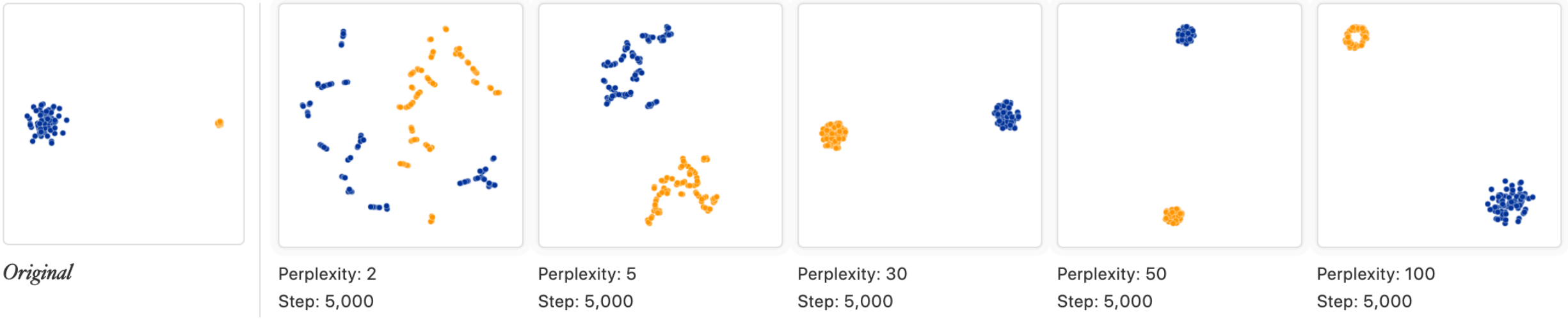
Perplexity: 50
Step: 5,000

Perplexity: 100
Step: 5,000

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

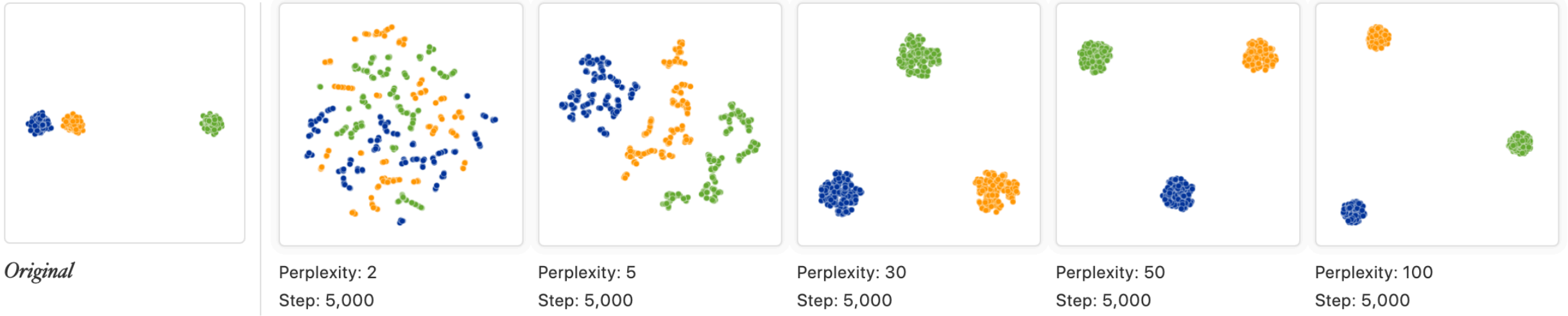
Cluster size doesn't matter. (Sigma's are set to normalize different local point densities.)



t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

Distance between clusters might not be meaningful.



Original

Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000

Perplexity: 30
Step: 5,000

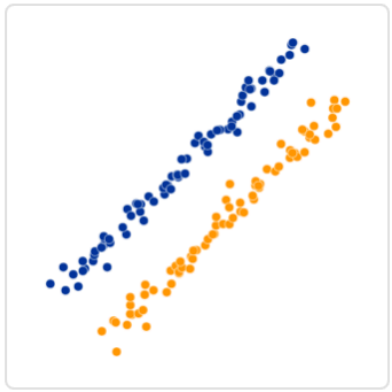
Perplexity: 50
Step: 5,000

Perplexity: 100
Step: 5,000

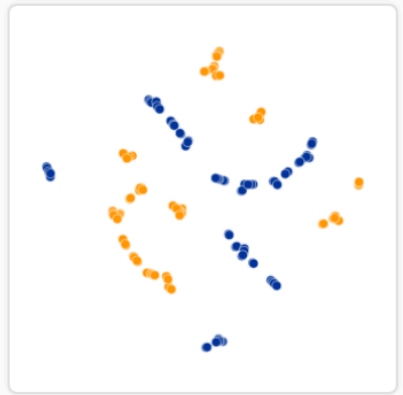
t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

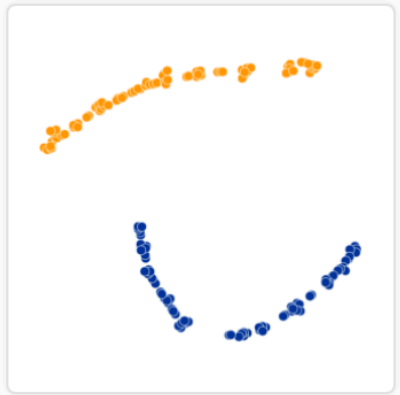
Shapes may be distorted.



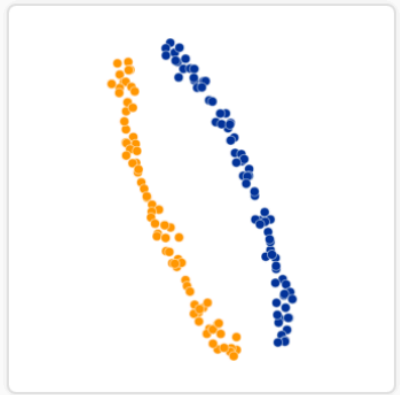
Original



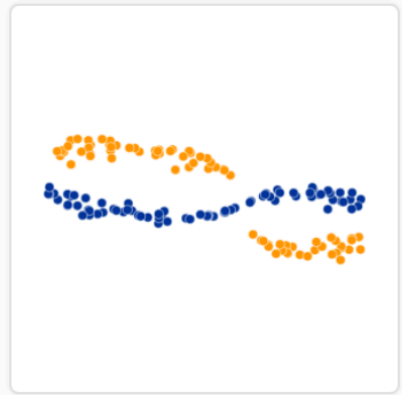
Perplexity: 2
Step: 5,000



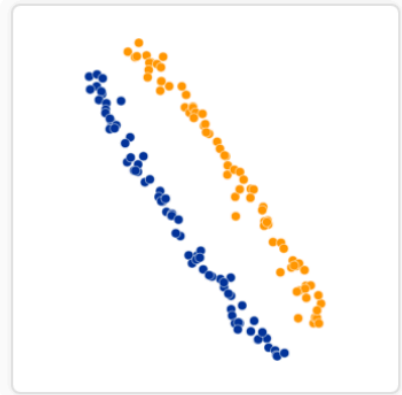
Perplexity: 5
Step: 5,000



Perplexity: 30
Step: 5,000



Perplexity: 50
Step: 5,000

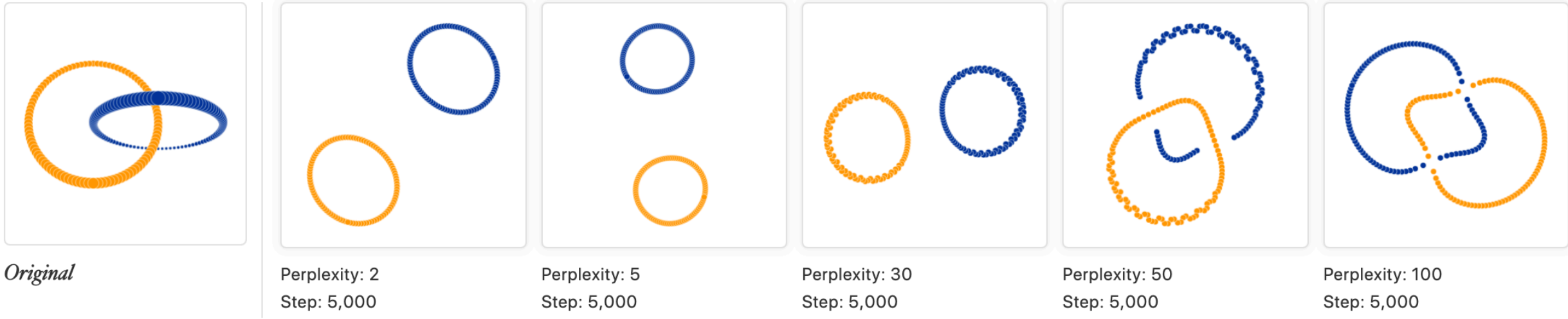


Perplexity: 100
Step: 5,000

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

Topology is not always preserved.



Original

Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000

Perplexity: 30
Step: 5,000

Perplexity: 50
Step: 5,000

Perplexity: 100
Step: 5,000

Questions Someone Might Ask You About Dimensionality Reduction

What is PCA? What sort of subspaces can it find? How does PCA choose directions for the lower dimension representation? How can PCA be solved? How can we see what fraction of variance a dimension of PCA's output captures? How to project a point to the PCA lower dimension space? What sort of relationships can PCA not identify? Why might applying PCA as a preprocessing step lead to issue?

What is t-SNE? What can it do that PCA can't? How does the perplexity parameter affect the output? What attributes of a t-SNE plot are interpretable?

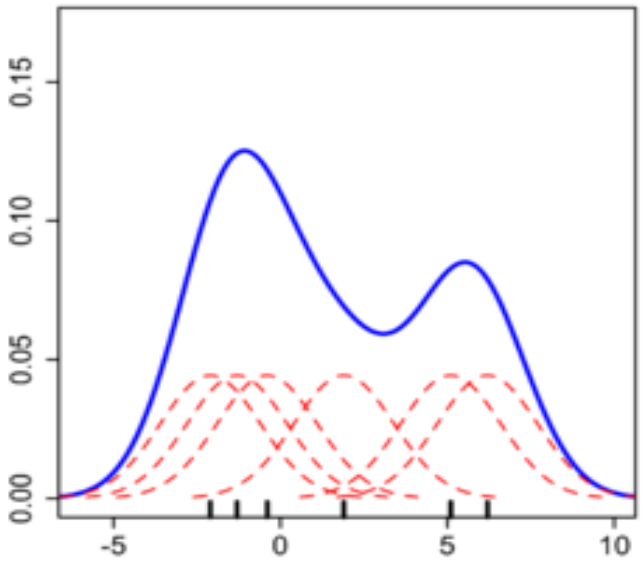
Topic Area:

Kernel Density Estimation

Kernel Density Estimation (KDE)

Core idea for KDE: Place a "blob" of probability around each observed datapoint called a Kernel. Main hyperparameter is what shape this kernel should take. Estimate probability at some point x as:

$$p(x) = \frac{1}{n}\sum_{i=0}^{n} \kappa(x, x_i)$$



In this 1D example:

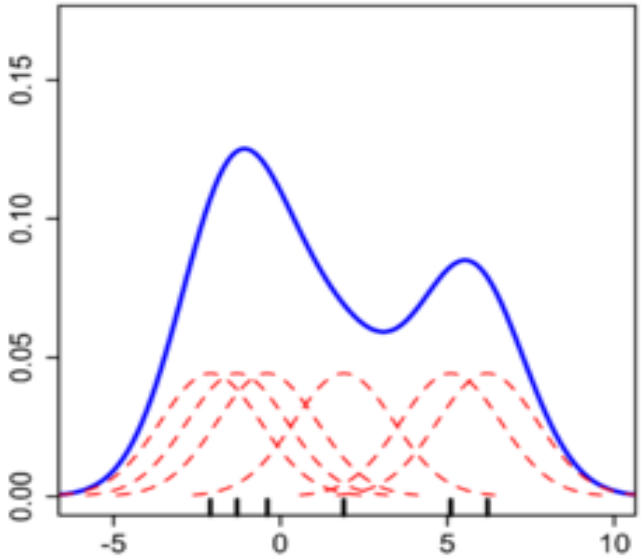
- Black dashes at x-axis are datapoints
- Red dashed lines are Gaussian Kernels
- Blue line is the KDE as defined in the equation above

Kernel Density Estimation (KDE)

A popular kernel is the Gaussian such that:

$$p(x) = \frac{1}{n}\sum_{i=0}^{n}\kappa(x, x_i) = \frac{1}{n}\sum_{i=0}^{n}\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-x_i)^2}{2\sigma}}$$

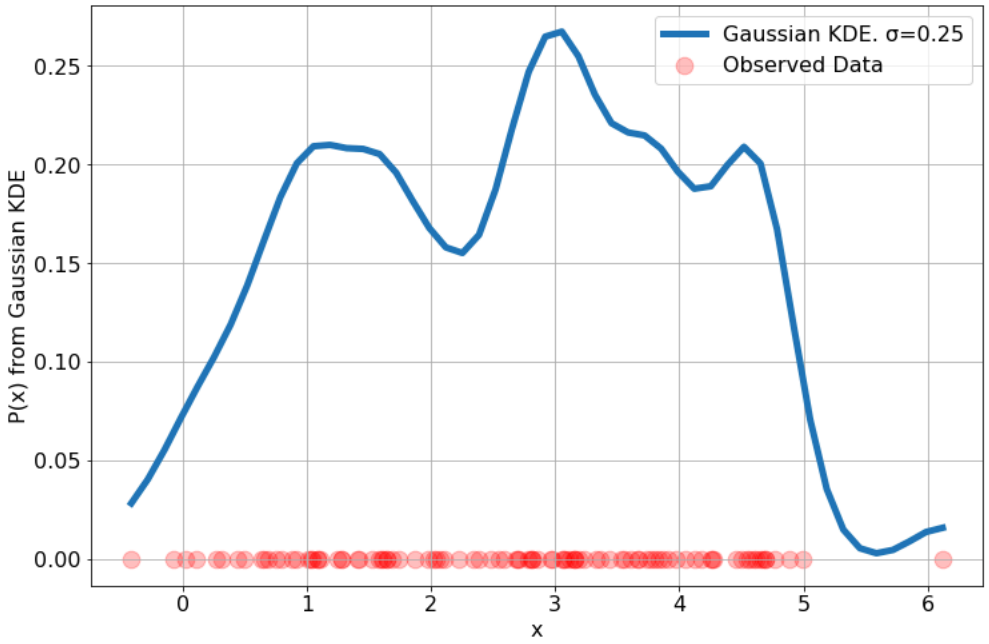
How to set the variance sigma? (Often referred to as a bandwidth)

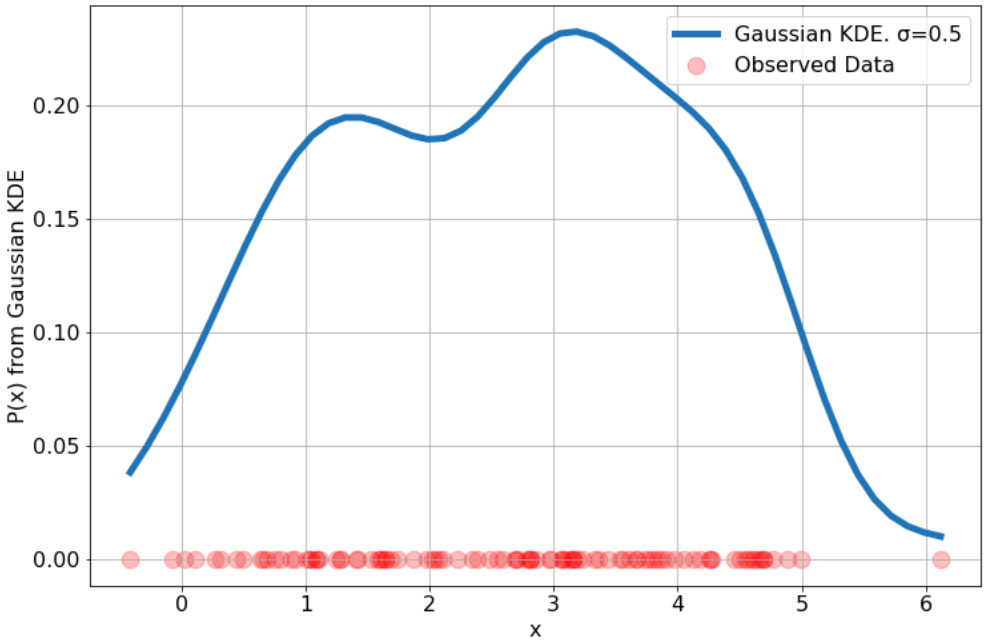


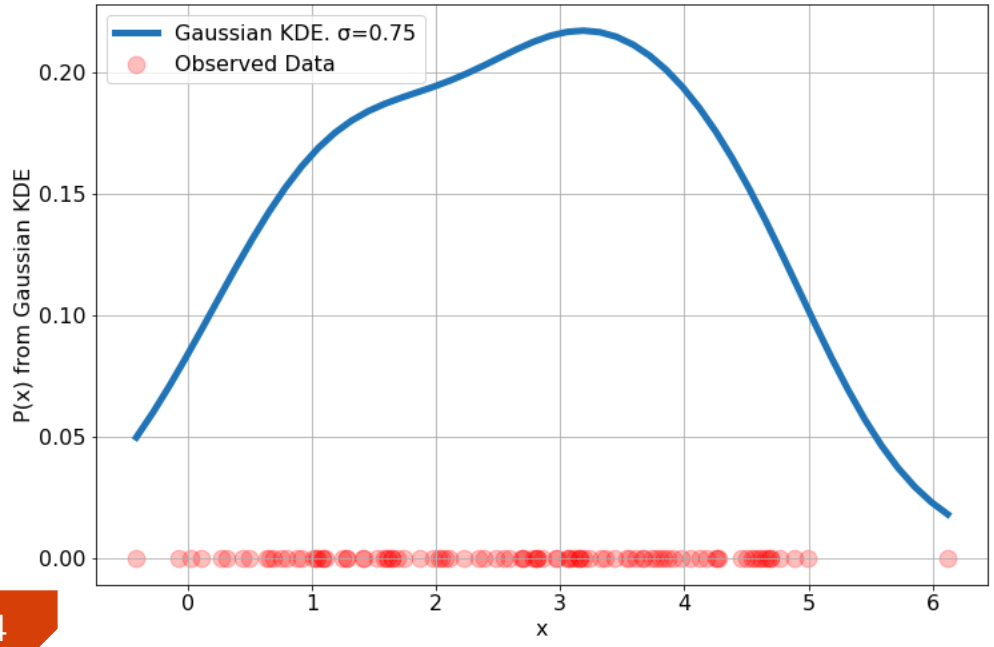
In this 1D example:

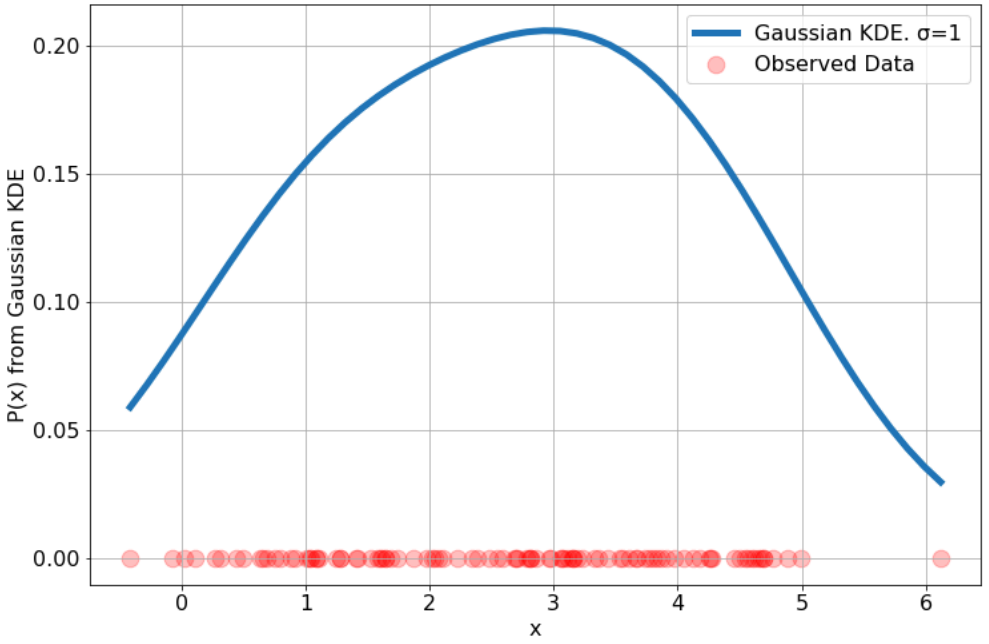
- Black dashes at x-axis are datapoints
- Red dashed lines are Gaussian Kernels
- Blue line is the KDE as defined in the equation above

Kernel Density Estimation (KDE)









Kernel Density Estimation (KDE)

Kernel Density Estimation is a flexible non-parametric way to fit a probability density given some observed data.

Basic idea is to put a "blob" (or kernel) of probability around each observed datapoint. The probability of a point in the space is the average probability assigned by these blobs at that point.

$$p(x) = \frac{1}{n}\sum_{i=0}^{n} \kappa(x, x_i)$$

Lots of kernels are possible but a common one is a Gaussian. Higher bandwidths result in smoother densities.

Topic Area:

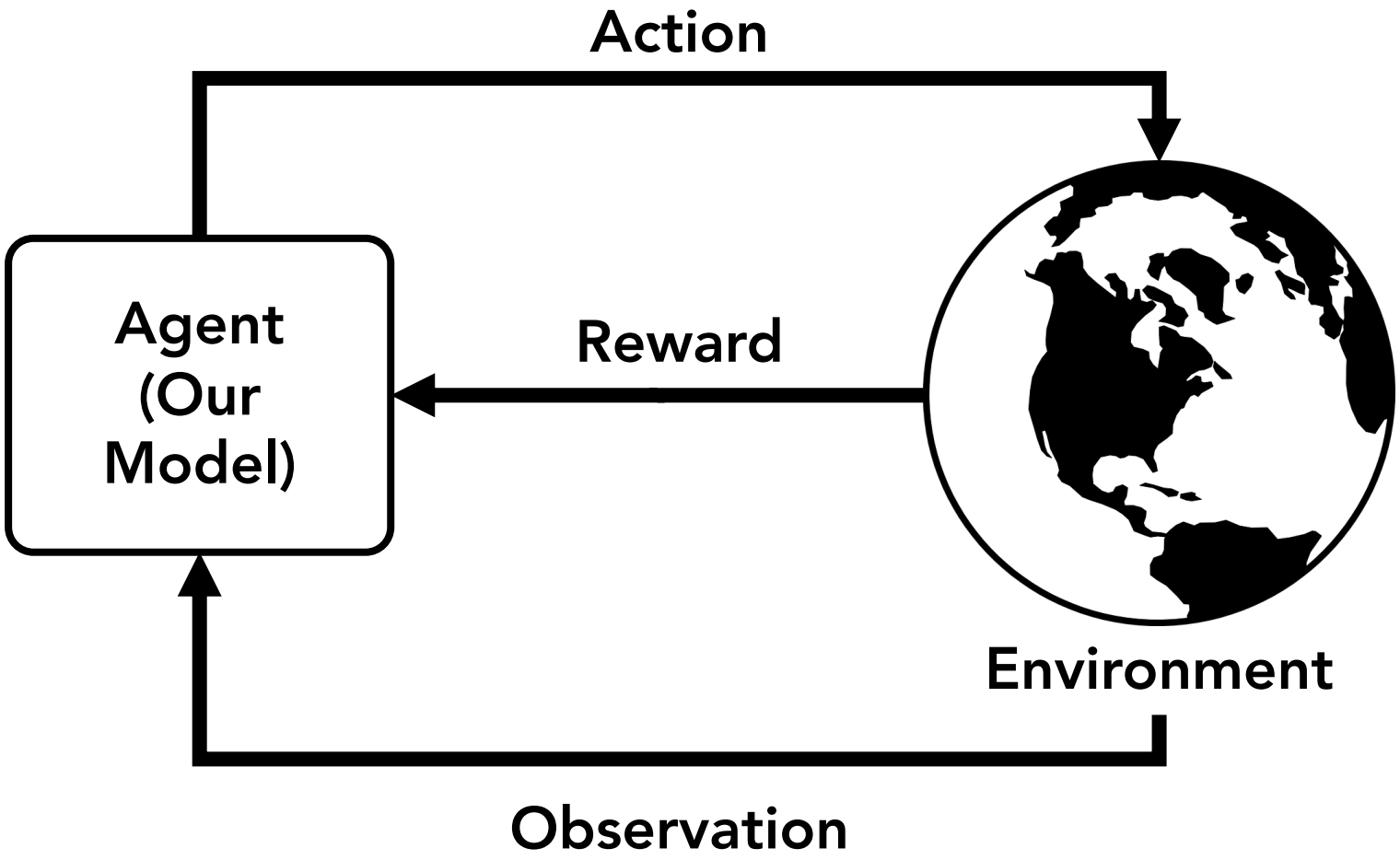
Reinforcement Learning

Sequential Decision Making

Contrast this with a **sequential decision making** task where an algorithm's output changes the next input that will be received.

For example, a robot walking along a narrow cliff path:

Sequential Decision Making



Markov Decision Processes

We will very often consider these sorts of problems within the framework of Markov Decision Process (MDP)s or Partially Observable Markov Decision Processes (POMDPs)

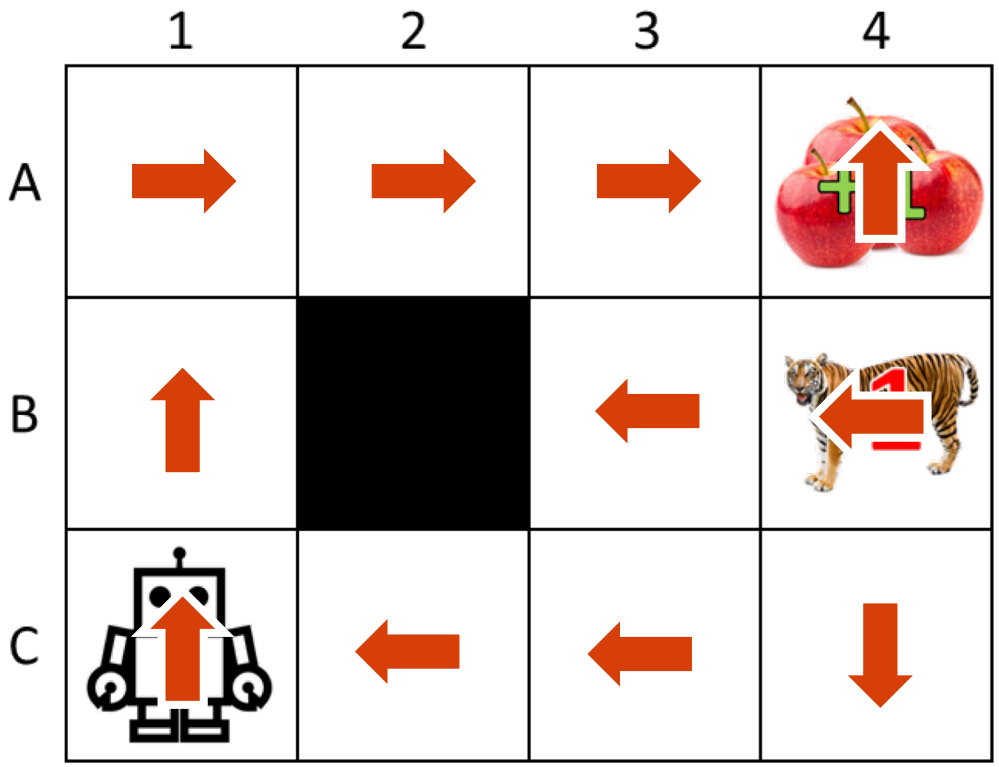
Decision process defined by

- $\mathcal{S}$ Set of possible **states**
- $\mathcal{A}$ Set of possible **actions**
- $\mathrm{R}: \mathcal{S} \rightarrow \mathbb{R}$ **Reward function** – mapping between states and rewards
- $\mathrm{T}: \mathcal{S} \times \mathcal{A} \rightarrow \Omega_{\mathcal{S}}$ **Transition** function – mapping between state-action pairs and a distribution over next states

"Markov" assumption says transitions depend only on current state – mathematically that $P(s_{t+1}|a, s_0, \ldots, s_t) = P(s_{t+1}|a, s_t)$

Behavior Policies

1. What is a behavior policy?



A behavior policy is a mapping from states to actions (or distributions over actions):

Usually use the symbol π to denote policy:

Deterministic policy: $\mathbf{a} = \boldsymbol{\pi}(\boldsymbol{s})$

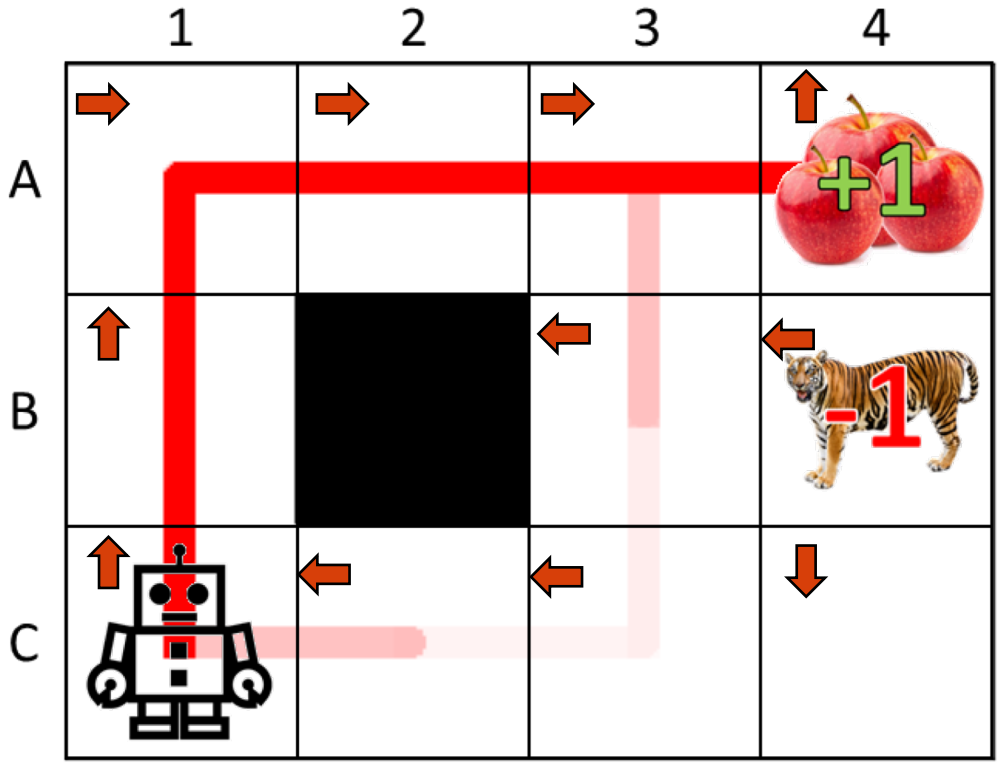
or

Stochastic policy: $P(A \mid s) = \pi(s)$

Denote the "best' behavior policy as π^*

Behavior Policies

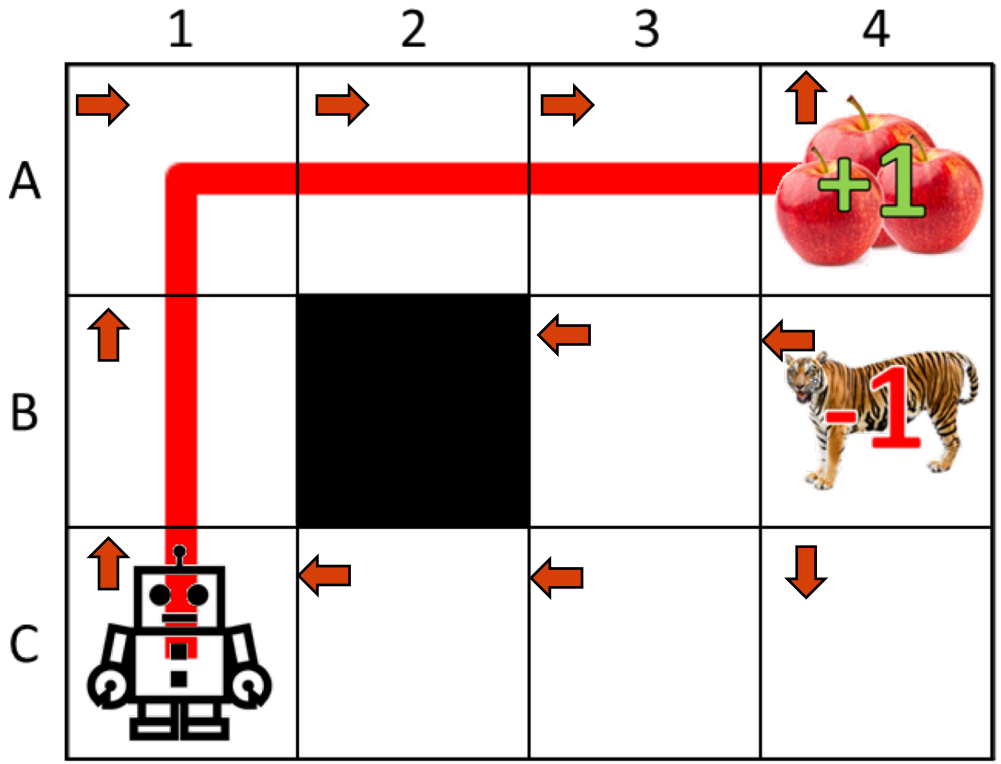
1. What is a behavior policy?



Recall that our environment has some randomness in the action space. Running the same policy from the same start, may result in different trajectories.

Intensity of red line show fraction of 1000 simulations that took that path.

Reward of a Trajectory



Given a trajectory s_0, s_1, s_2, s_3, s_4 … can compute the **additive reward** as:

$$\sum_{t=0}^{\infty} R(s_t) = R(s_0) + R(s_1) + R(s_2) + \cdots$$

Will usually want to use **discounted rewards**:

$$\sum_{t=0}^{\infty} \gamma^t R(s_t) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

where $0 \geq \gamma \geq 1$ is the discount factor.

What algorithms can we use to find a behavior policy that performs well?

If we know the transition function and reward function (and state space isn't huge):

Value Iteration / Policy Iteration. No real learning happens here typically, just dynamic programming to iteratively refine our estimates.

If we don't know transition and reward functions:

Model-based Reinforcement Learning: Learn transition function and reward function from observation and then use these to find best policy.

Model-free Reinforcement Learning: Directly learn a policy to maximize reward based on experience.

Imagine playing a new game whose rules you don't know;

after a hundred or so moves, your opponent announces,

"You lose".

- Russel and Norvig – Introduction to Artificial Intelligence

Reinforcement learning describes algorithms for producing good policies in MDPs when transition dynamics and reward functions are initially unknown.

Challenges in Reinforcement Learning

- Actions have unknown and possibly non-deterministic effects which are initially unknown and must be learned
  - **Explore/Exploit Tradeoff:** How much time should we spend trying to explore the environment vs learning how to maximize reward in the area we've already explored?
- Rewards / punishments can be infrequent and are often at the end of long sequences
  - **Credit Assignment Problem:** How do we determine which actions are *really* responsible for the reward or punishment?
- The environment may be massive and exploring all possible actions and states infeasible
  - The game of Go has 2.08×10^{170} legal board configurations!

Intuition:

$$\nabla_\theta J(\theta) \approx \sum_n \sum_{s_i, a_i \in \tau_n} \nabla_\theta \log \pi(a_i | s_i; \theta)\, r(\tau_n)$$

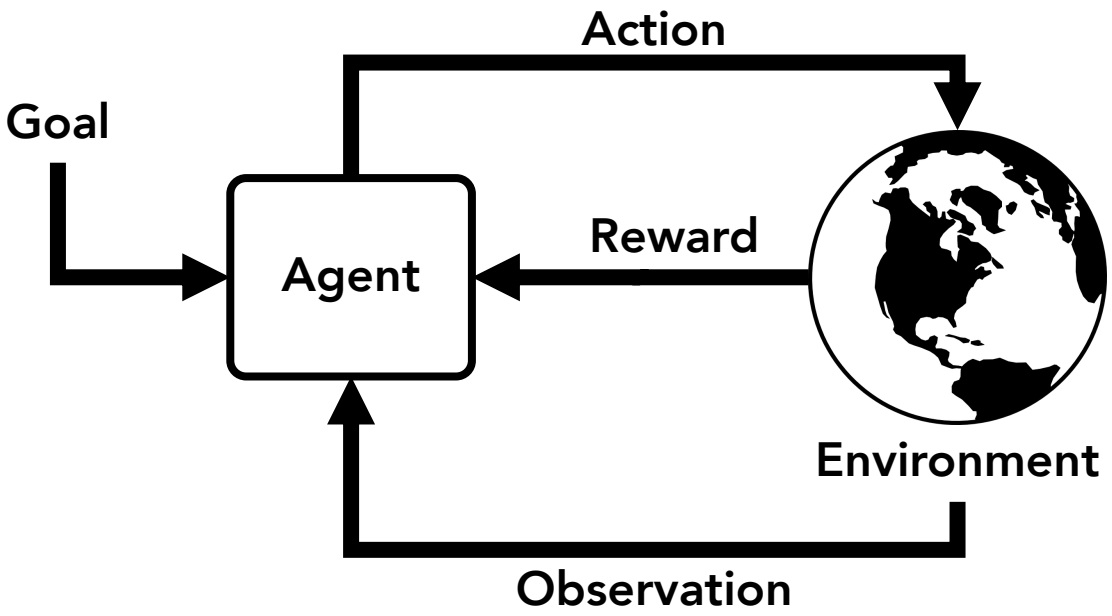
- If trajectory reward is positive, push up the probabilities of all the actions involved
- If trajectory reward is negative, push down the probabilities of all the action involve

All actions in trajectory move in same direction based on reward?!?

I know it seems too simple but it averages out.

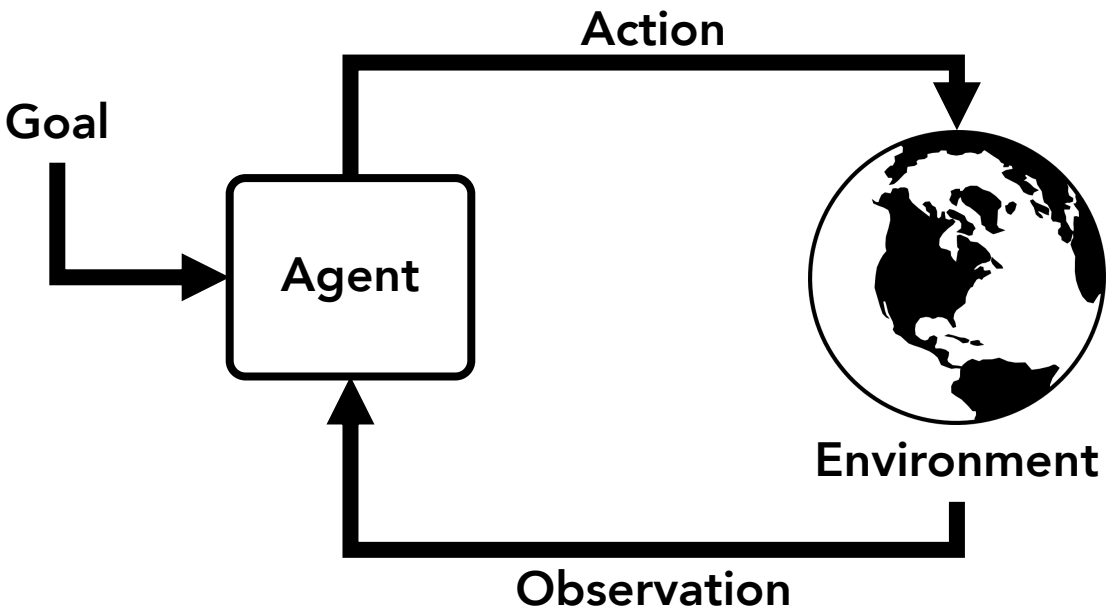
Reinforcement Learning

- Environment provides feedback
- No examples of optimal policy



Imitation Learning

- Have expert demonstrations (possibly interactive)



Behavior Cloning

- **Strengths:**
  - Dead simple. Seriously. It is just supervised learning.
  - Works well when minimizing 1-step deviation is sufficient.

- **Weaknesses:**
  - Compounding errors.
  - Data distribution miss-match.

Summary of Behavior Cloning

Use set of demonstrations as if they were targets for a supervised learning task and minimizing 1-step error for your policy. Super simple but has trouble with dataset miss-match.

- **When to use this?**
  - When the state space is well-covered by the demonstrator.
  - When recovering from 1-step deviations is easy.
  - To pre-train before doing a full RL approach.

Questions Someone Might Ask You About Reinforcement Learning

What is a sequential decision process and how does it differ from standard machine learning problems? What is a Markov Decision Process? What is a state? An action? A transition function? A reward function? What is a behavior policy? What is model-free vs model-based reinforcement learning? What is the principal behind policy-gradient methods like REINFORCE? What is imitation learning and how does it differ from reinforcement learning? What is behavior cloning and what are its drawbacks?

Note: No in-depth questions will be asked about RL on the final.

Next time

Next Time: We take the final exam and then that's it.