6.2 - Neural Networks I

What is a 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 $\mathbf{b} \in \mathbb{R}$

Neural networks are a class of biologically-inspired models for classification and regression

- **Key Concepts:**
  - Artificial Neurons
    - Activation Functions
  - Neural Networks
    - Linear Layers
    - Universal Approximators (in the limit)
  - Training Neural Networks
    - Backpropagation
    - Stochastic Gradient Descent

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

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

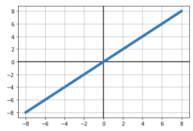
$\underset{d\times1}{\boldsymbol{w}}$ $\underset{d\times1}{\boldsymbol{x}}$ $\underset{1\times1}{\boldsymbol{b}}$

How does it relate to logistic regression?

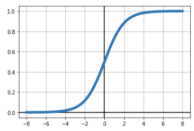
It has a non linear activation function with a linear decision boundary.

What are common activation functions?

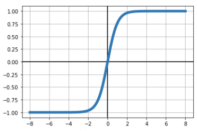
None $\sigma(z) = z$



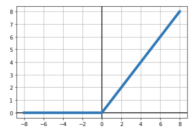
Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$



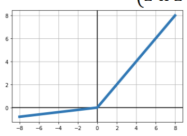
tanh $\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$



ReLU $\sigma(z) = \max(0, z)$

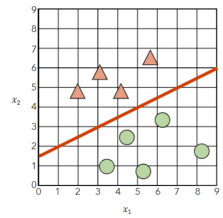


Leaky ReLU $\sigma(z) = \begin{cases} \alpha z & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$



Example:

What set of weights would make it so $w^T x + b > 0$ for all the green points and <0 for red points?



Step 1) Draw a line separating the classes

Step 2) Figure out the equation of that line

$$1x_2 = w_1 x_1 + b$$

$$1x_2 = 0.5x_1 + 1.5$$

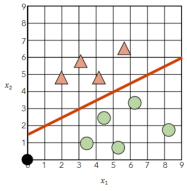
A $-0.5x_1 + 1x_2 - 1.5 = 0$ $\quad w = [-0.5, 1]\ b = -1.5$

B $0.5x_1 - 1x_2 + 1.5 = 0$ $\quad w = [0.5, -1]\ b = 1.5$

Which to choose?

Hands On With Linear Classifiers

What set of weights would make it so $w^T x + b > 0$ for all the green points and <0 for red points?



Step 1) Draw a line separating the classes

Step 2) Figure out the equation of that line

A $-0.5x_1 + 1x_2 - 1.5 = 0$

B $0.5x_1 - 1x_2 + 1.5 = 0$

Step 3) Check the sign on each side of the line
(I usually use (0,0) such that value is just -b).

A $-0.5 * 0 + 1 * 0 - 1.5 = -1.5$

B $0.5 * 0 - 1 * 0 + 1.5 = 1.5$

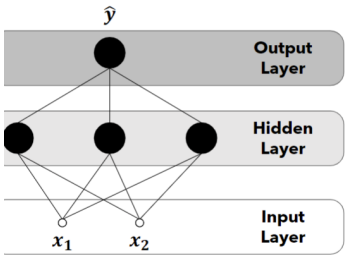
Step 4) Choose weights that match pos/neg classes

$$w = [0.5, -1]\ b = 1.5$$

CS 434 Lines Demo: https://colab.research.google.com/drive/1DELXo_sC1kP22_N_PB2P7bBwoWe_pHG?usp=sharing 30

What is fully-connected 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

Build some intuition for training

How do I train these things? An Example from Regression

How can I update my parameters to reduce the loss $\mathcal{L}$?

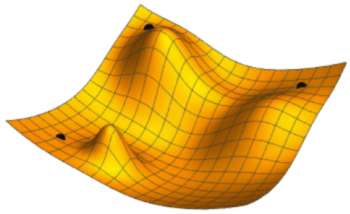
Gradient Descent Algorithm

$w \leftarrow random(\ )$

while$|\nabla_w \mathcal{L}| > \epsilon$ *or iters remain*

$w = w - \alpha \nabla_w \mathcal{L}(w)$

Learning Rate / Step Size



Questions:

Question 1

Artificial and biological neurons behave identically.

- True
- False

False! False! A million times false! Artificial neurons (and neural networks) are inspired by biological neurons but have very very different behavior and computation.

Question 2

A single neuron with a logistic activation can represent non-linear boundaries in classification problems.

- True
- False

False. A single neuron computes a linear function of its inputs and passes it through a logistic function -- matching logistic regression.

Question 3

What is a loss function?

Your Answer:

Determines innaccuracies

A loss function is a function that measures how bad a network's output is, usually relative to some gold-standard for what the output should be.

Question 4

Networks with Sigmoid activations tend to converge faster than ReLU networks.

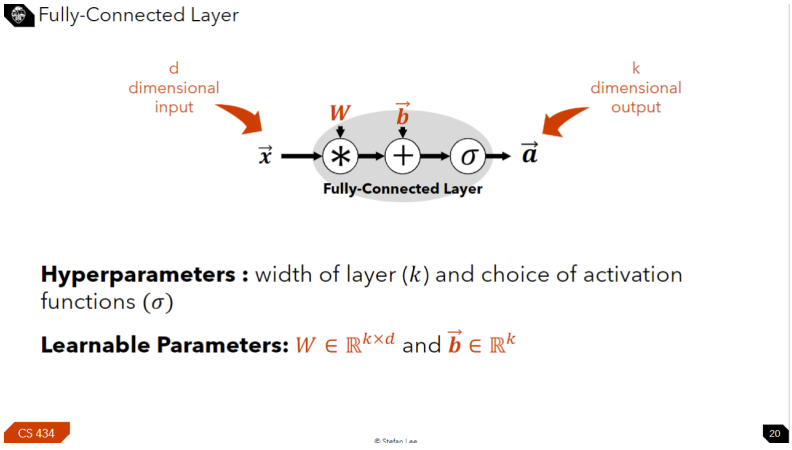
- True
- False

False. ReLUs tend to converge faster because they do not saturate for positive inputs like Sigmoid.

7.1 Neural Networks II

What is a fully-connected layer?

Notation:



What is a loss function?

How bad a neural network's output is. (L).

Squared Error

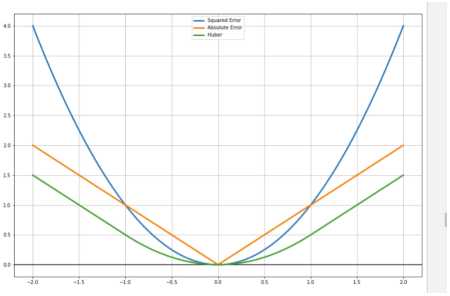
$$L(a,b) = (a-b)^2$$

Absolute Error

$$L(a,b) = |a-b|$$

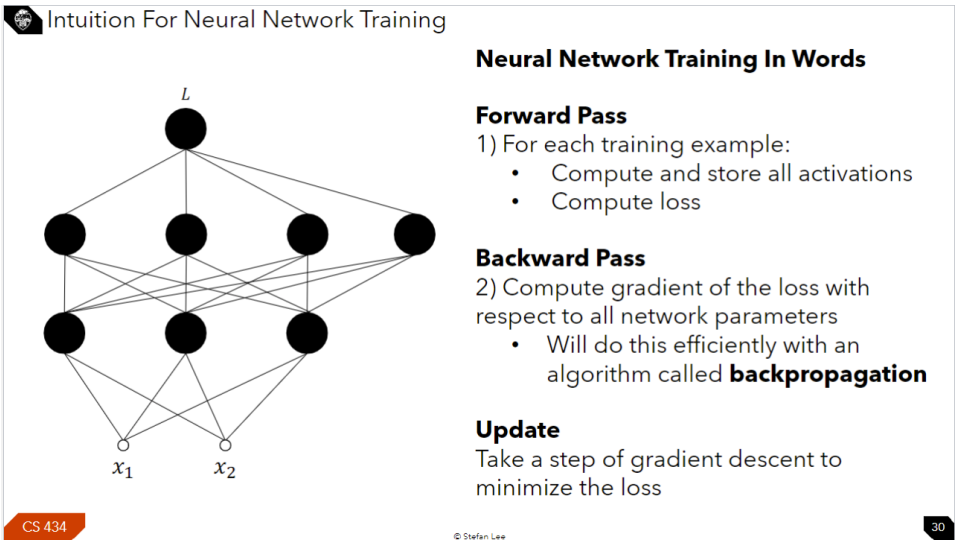
Huber

$$L(a,b) = \begin{cases} \frac{1}{2}(a-b)^2 & \text{if } |a-b| < \delta \\ \delta|a-b| - \frac{1}{2}\delta^2 & \text{else} \end{cases}$$



(Quick review of vector calculus)

What is forward / reverse mode differentiation?



How does backpropagation work?

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.

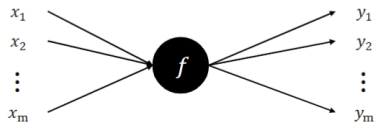
2) We shouldn't recompute something we already know.

What is a computation graph?

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: $\frac{\delta y_i}{\delta x_j}\ \forall i, j$



L7.2 - Neural Networks III and Decision Trees.pdf

What are the intuitions for some advanced structures in neural networks?

Intuition 1: Translational Invariance / Locality Assumption

Intuition 2: Recurrence

What are decision trees?

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

Can use discrete or continuous inputs.

Algorithm Name: Decision Trees
Type: Supervised Classifiers / Regressor
Synonyms: Classification and Regression Trees (CART)

Key Idea: Recursively subdivide the input space.
Works for discrete or continuous inputs.

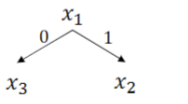
Algorithms for learning decision tress:
- ID3
- C4.5 c5

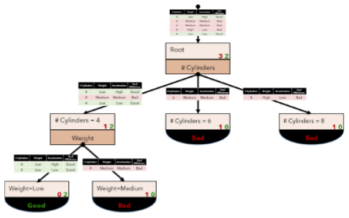
Related concept: random forests (multiple decision trees)

Decision trees are when you split up a matrix of data by each individual attribute. For example, if you had 15 attributes about humans and wanted to make a prediction about weight, you could split the attributes by how tall someone is.

As the number of nodes (or depth) of the tree increases, the model can represent more complex functions (aka the hypothesis space grows):

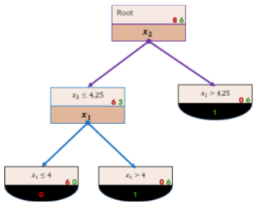
- **Depth 1** ("decision stump") can represent any Boolean function of one feature.
- **Depth 2** Any Boolean function of two features and some Boolean functions of three features (e.g., $(x_1 \text{ and } x_2) \text{ or } (\text{not } x_1 \text{ and not } x_3)$)
- Etc..





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



Consider the simple recursive algorithm below:

BasicGreedyAlgorithm(dataset S):
Based on S, choose "best" test t to split the data
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)

How are they learned?
What is entropy, conditional entropy, and information Gain?

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:

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

Entropy: The degree of uncertainty in a system.
Conditional Entropy: Conditional entropy is the condition based on each individual random variable.

What about the entropy of the class labels in a dataset?

# Cylinders	Weight	Acceleration	Fuel Efficiency
4	Low	High	Good
6	Medium	Medium	Bad
4	Medium	Medium	Bad
8	High	Low	Bad
4	Low	Low	Good

$$P(Y = Good) = 2/5$$

$$P(Y = Bad) = 3/5$$

$$H(Y) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} \approx 0.971$$

What is the conditional entropy of the class labels in the following split?

Split on # Cylinders

# Cylinders	Weight	Acceleration	Fuel Efficiency
4	Low	High	Good
4	Medium	Medium	Bad
4	Low	Low	Good
6	Medium	Medium	Bad
8	High	Low	Bad

$P(Y = Good \mid \#Cylinders = 4) = 2/3$
$P(Y = Bad \mid \#Cylinders = 4) = 1/3$
$P(\#Cylinders = 4) = 3/5$
$P(Y = Good \mid \#Cylinders = 6) = 0$
$P(Y = Bad \mid \#Cylinders = 6) = 1$
$P(\#Cylinders = 6) = 1/5$
$P(Y = Good \mid \#Cylinders = 8) = 0$
$P(Y = Bad \mid \#Cylinders = 8) = 1$
$P(\#Cylinders = 8) = 1/5$

$$H(Y|X) = -\frac{3}{5}\left(\frac{2}{3}\log_2\frac{2}{3}+\frac{1}{3}\log_2\frac{1}{3}\right) - \frac{1}{5}\left(\frac{0}{1}\log_2\frac{0}{1}+1\log_2 1\right) - \frac{1}{5}\left(\frac{0}{1}\log_2\frac{0}{1}+1\log_2 1\right)$$

Information Gain: $\mathrm{IG(X)} = H(Y) - H(Y|X)$
Entropy of class label before splitting on an attribute minus the conditional entropy after the split.

$$\mathrm{IG(\#Cylinders)} = H(Y) - H(Y|\#\mathrm{Cylinders}) = 0.971 - 0.551 = 0.42$$
$$\mathrm{IG(Weight)} = H(Y) - H(Y|\mathrm{Weight}) = 0.971 - 0 = 0.971$$

We reduce uncertainty *more* by choosing to split on the Weight attribute.

We've discussed training neural networks with gradient descent. To compute the gradients of the loss with respect to our parameters, we proposed the backpropagation algorithm. Explain the benefits of backpropagation.

Your Answer:

i dont rememeber but would with my notes

Backpropagation is an efficient way to compute gradients of the loss with respect to model parameters. Firstly, it is a reverse-mode differentiation and computes the product of intermediate Jacobians from the output of the network backwards -- reducing cost of matrix multiplication in computing gradients. Secondly and more important, the backwards pass allows us to store and reuse loss gradients as we work our way backwards through the network.

What is a Jacobian?

Your Answer:

Idk

We refer to the gradient of vector-valued functions with respect to vector inputs as a Jacobian matrix. If y(x) is a function that outputs a vector given an input vector x, then the Jacobian dy(x)/dx would be a matrix where the i,j'th entry would be the partial derivative of the i'th element of y(x) with respect to the j'th element of x.

Neural networks are universal approximators regardless of their size.

- (•) True
- ( ) False

False. For any given function and error tolerance, there exists a neural network of finite size that achieves that level of error; however, a fixed-size network may or may not be large enough for any given function.

What is a computational graph and how can it be used to implement backpropagation?

Your Answer:

idk

A computational graph is a directed acyclic graph with node corresponding to units of computation and edges corresponding to the results of these computations. If each node implements both a forward and backward (i.e. computing a Jacobian of output with respect to input) operation, then the graph can be used to calculate derivatives of the output of arbitrary combinations of operations via backpropagation.

Question 5

Convolutional neural networks make the assumption that relevant features in an input can be found by examining local regions of the input -- e.g. windows of time in a sequence or regions of an image.

- (•) True
- ( ) False

True! The locality assumption is a key part of convolutional neural networks.

True

Question 6

Decision trees are composed of internal nodes that provide label outputs and leaf nodes that implement tests to divide datapoints.

- (•) True
- ( ) False

False. The other way around -- internal nodes implement tests to divide datapoints and leaf nodes are labels.

Question 7

Splits in standard decision trees can be diagonal lines.

- (•) True
- ( ) False

False. In standard decision trees, the splits are axis-aligned because they test only single attributes.

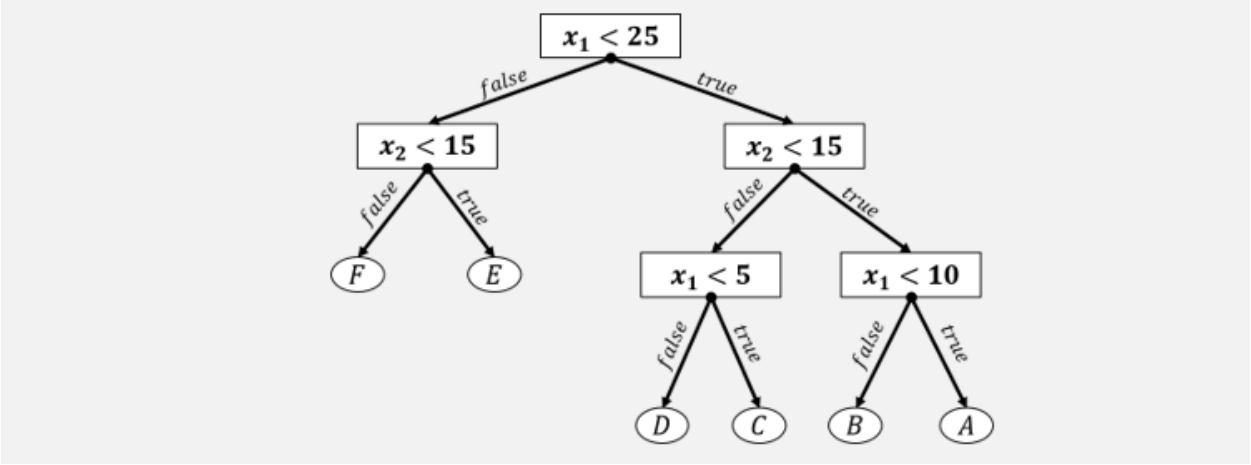
Question 8

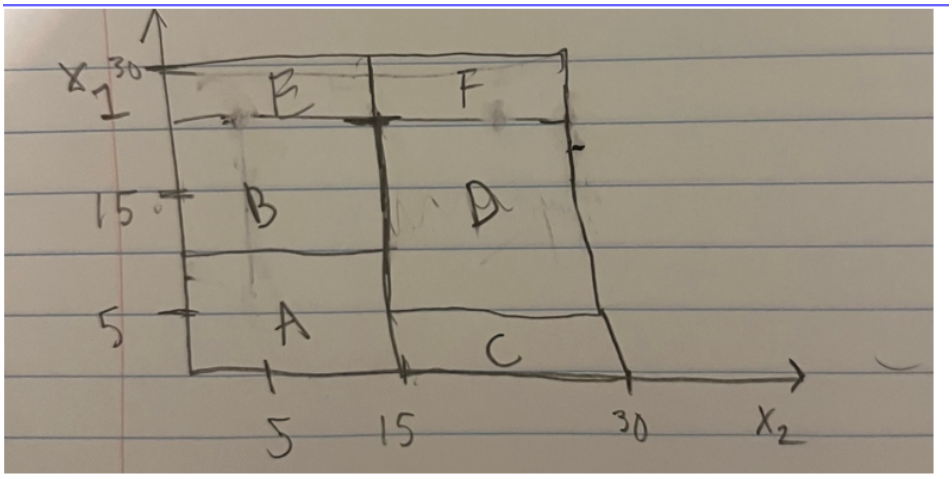
Finding a decision tree with minimum depth that achieves minimum error on training data has polynomial computational complexity.

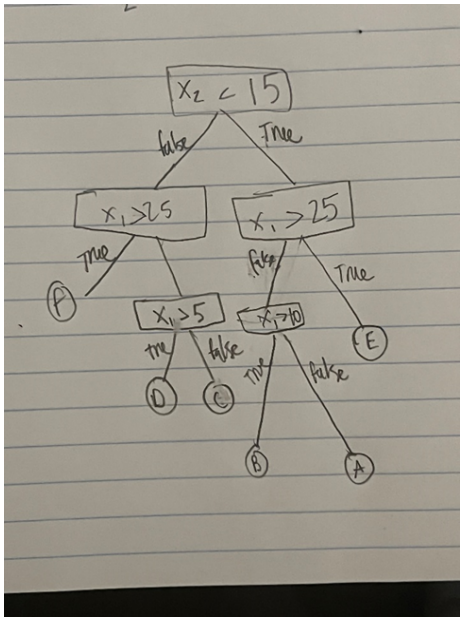
- (•) True
- ( ) False

False. This problem is NP-Hard because all possible combinations of tree structures need to be considered.

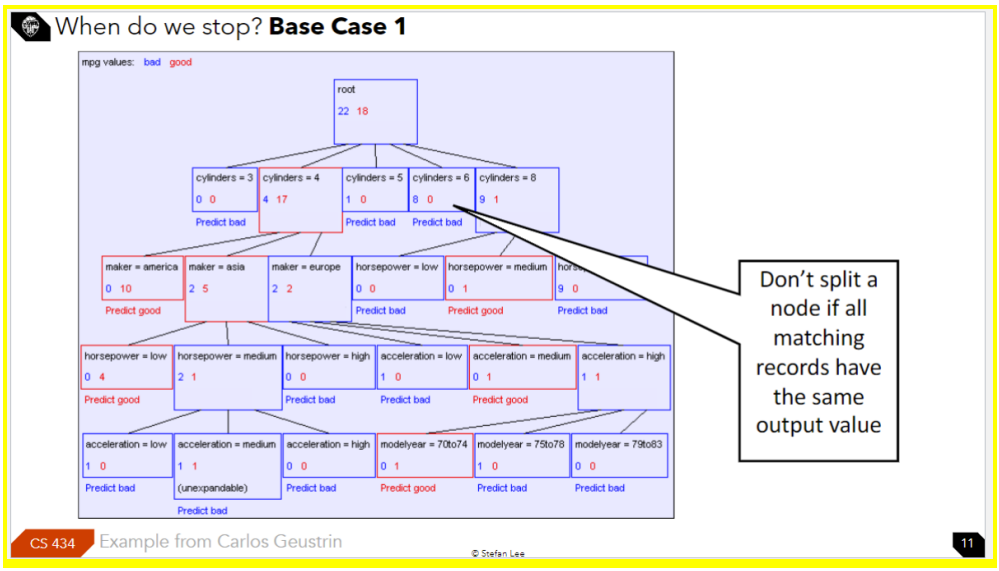
Drawing tree example:

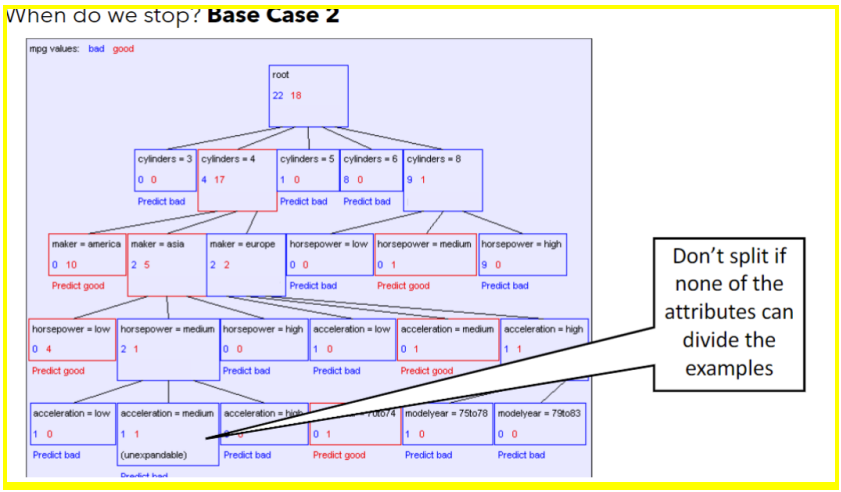






8.1 Random Forests and Ensemble Methods





- **Base Case One:** If all records in current data subset have the same output/class then don't recurse because no further improvement could be made
- **Base Case Two:** If all records have exactly the same set of input features/attributes then don't recurse because nothing will split them

An Intuitive Sounding Base Case 3:

If all attributes have zero information gain, then don't recurse

How do they deal with continuous features?

Suppose we have both continuous and discrete features:
- **Continuous:** Cylinder Volume, Vehicle Weight
- **Discrete:** Manufacturer, Diesel/Regular

Don't really need to do anything special for this case.

- At each step consider all possible splits including:
  - Splits on discrete attributes
  - Threshold splits in-between datapoints for continuous attributes

Essentially just use a range to split. E.g. 10-20, 20-30, 30-40 etc, rather than hard medium easy.

What do you do about 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.

Example:

Married	Education	Credit Score	Approve Loan
No	Highschool	680	No
Yes	Highschool	720	Yes
Yes	College	720	Yes
No	Grad School	800	Yes
No	Grad School	540	No
Yes	Grad School	680	No
No	College	680	Yes

$$IG(X) = H(Y) - H(Y|X)$$

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

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

1) List Possible Splits:
- Married, Education, Credit Score > t (t = 610, 700, 760)

2) Compute Information Gain for Each Possible Split:

- **Married**: (4 3) → No (2 2) Yes (2 1)

$$H(Y|X) = -\frac{4}{7}\left(\frac{2}{4}\log_2\frac{2}{4}+\frac{2}{4}\log_2\frac{2}{4}\right) - \frac{3}{7}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right) =$$

- **Education**: (4 3) → Highschool (1 1) College (2 0) Grad School (1 2)

$$H(Y|X) = -\frac{2}{7}\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right) - \frac{2}{7}\left(\frac{2}{2}\log_2\frac{2}{2}\right) - \frac{3}{7}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right) =$$

- **Credit Score > 610**: (4 3) → ≤610 (0 1) >610 (4 2)

$$H(Y|X) = -\frac{1}{7}\left(\frac{1}{1}\log_2\frac{1}{1}\right) - \frac{6}{7}\left(\frac{4}{6}\log_2\frac{4}{6}+\frac{2}{6}\log_2\frac{2}{6}\right) =$$

- **Credit Score > 700**: (4 3) → ≤700 (1 3) >700 (3 0)

$$H(Y|X) = -\frac{4}{7}\left(\frac{1}{4}\log_2\frac{1}{4}+\frac{3}{4}\log_2\frac{3}{4}\right) - \frac{3}{7}\left(\frac{3}{3}\log_2\frac{3}{3}\right) =$$

- **Credit Score > 760**: (4 3) → ≤760 (3 3) >760 (1 0)

$$H(Y|X) = -\frac{6}{7}\left(\frac{3}{6}\log_2\frac{3}{6}+\frac{3}{6}\log_2\frac{3}{6}\right) - \frac{1}{7}\left(\frac{1}{1}\log_2\frac{1}{1}\right) =$$

Married	Education	Credit Score	Approve Loan
No	Highschool	680	No
Yes	Highschool	720	Yes
Yes	College	720	Yes
No	Grad School	800	Yes
No	Grad School	540	No
Yes	Grad School	680	No
No	College	680	Yes

$$IG(X) = H(Y) - H(Y|X)$$

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

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

1) List Possible Splits:

- Married, Education, Credit Score > t (t = 610, 700, 760)

2) Compute Information Gain for Each Possible Split:

- **Married**: (4 3) → No (2 2) Yes (2 1)

$$H(Y|X) = -\frac{4}{7}\left(\frac{2}{4}\log_2\frac{2}{4}+\frac{2}{4}\log_2\frac{2}{4}\right)-\frac{3}{7}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right) \approx 0.964$$

- **Education**: (4 3) → Highschool (1 1) College (2 0) Grad School (1 2)

$$H(Y|X) = -\frac{2}{7}\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right)-\frac{2}{7}\left(\frac{2}{2}\log_2\frac{2}{2}\right)-\frac{3}{7}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right) \approx 0.512$$

- **Credit Score > 610**: (4 3) → ≤610 (0 1) >610 (4 2)

$$H(Y|X) = -\frac{1}{7}\left(\frac{1}{1}\log_2\frac{1}{1}\right)-\frac{6}{7}\left(\frac{4}{6}\log_2\frac{4}{6}+\frac{2}{6}\log_2\frac{2}{6}\right) \approx 0.787$$

- **Credit Score > 700**: (4 3) → ≤700 (1 3) >700 (3 0)

$$H(Y|X) = -\frac{4}{7}\left(\frac{1}{4}\log_2\frac{1}{4}+\frac{3}{4}\log_2\frac{3}{4}\right)-\frac{3}{7}\left(\frac{3}{3}\log_2\frac{3}{3}\right) \approx 0.463$$

- **Credit Score > 760**: (4 3) → ≤760 (3 3) >760 (1 0)

$$H(Y|X) = -\frac{6}{7}\left(\frac{3}{6}\log_2\frac{3}{6}+\frac{3}{6}\log_2\frac{3}{6}\right)-\frac{1}{7}\left(\frac{1}{1}\log_2\frac{1}{1}\right) \approx 0.857$$

Married	Education	Credit Score	Approve Loan
No	Highschool	680	No
No	Grad School	540	No
Yes	Grad School	680	No
No	College	680	Yes

Married	Education	Credit Score	Approve Loan
Yes	Highschool	720	Yes
Yes	College	720	Yes
No	Grad School	800	Yes

$$IG(X) = H(Y) - H(Y|X)$$

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

1) List Possible Splits:

- Married, Education, Credit Score > t (t = 610, ~~700~~, ~~760~~)

2) Compute Information Gain for Each Possible Split:

- **Married**: (1 3) → No (1 2) Yes (0 1)

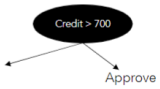
$$H(Y|X) = -\frac{3}{4}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right)-\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right)$$

- **Education**: (1 3) → Highschool (0 1) College (1 0) Grad School (0 2)

$$H(Y|X) = -\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right)-\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right)-\frac{2}{4}\left(\frac{2}{2}\log_2\frac{2}{2}\right)$$

- **Credit Score > 610**: (1 3) → ≤610 (0 1) >610 (1 2)

$$H(Y|X) = -\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right)-\frac{3}{4}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right)$$



Married	Education	Credit Score	Approve Loan
No	Highschool	680	No
No	Grad School	540	No
Yes	Grad School	680	No
No	College	680	Yes

Married	Education	Credit Score	Approve Loan
Yes	Highschool	720	Yes
Yes	College	720	Yes
No	Grad School	800	Yes

$$IG(X) = H(Y) - H(Y|X)$$

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

1) List Possible Splits:

- Married, Education, Credit Score > t (t = 610, ~~700~~, ~~760~~)

2) Compute Information Gain for Each Possible Split:

- **Married**: (1 3) → No (1 2) Yes (0 1)

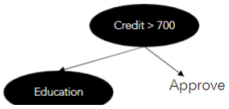
$$H(Y|X) = -\frac{3}{4}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right)-\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right) \approx 0.688$$

- **Education**: (1 3) → Highschool (0 1) College (1 0) Grad School (0 2)

$$H(Y|X) = -\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right)-\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right)-\frac{2}{4}\left(\frac{2}{2}\log_2\frac{2}{2}\right) \approx 0$$

- **Credit Score > 610**: (1 3) → ≤610 (0 1) >610 (1 2)

$$H(Y|X) = -\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right)-\frac{3}{4}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right) \approx 0.688$$



In Class Exercise: Learning a Decision Tree

Married	Education	Credit Score	Approve Loan
No	Highschool	680	No
No	Grad School	540	No
Yes	Grad School	680	No
No	College	680	Yes

Married	Education	Credit Score	Approve Loan
Yes	Highschool	720	Yes
Yes	College	720	Yes
No	Grad School	800	Yes

$$IG(X) = H(Y) - H(Y|X)$$

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

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

1) List Possible Splits:

- Married, Education, Credit Score > t (t = 610, ~~700~~, ~~760~~)

2) Compute Information Gain for Each Possible Split:

- **Married**: (1 3) → No (1 2) Yes (0 1)

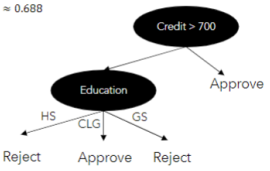
$$H(Y|X) = -\frac{3}{4}\left(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}\right) - \frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right) \approx 0.688$$

- **Education**: (1 3) → Highschool (0 1) College (1 0) Grad School (0 2)

$$H(Y|X) = -\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right) - \frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right) - \frac{2}{4}\left(\frac{2}{2}\log_2\frac{2}{2}\right) \approx 0$$

- **Credit Score > 610**: (1 3) → ≤610 (0 1) >610 (1 2)

$$H(Y|X) = -\frac{1}{4}\left(\frac{1}{1}\log_2\frac{1}{1}\right) - \frac{3}{4}\left(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}\right) \approx 0.688$$



CS 434

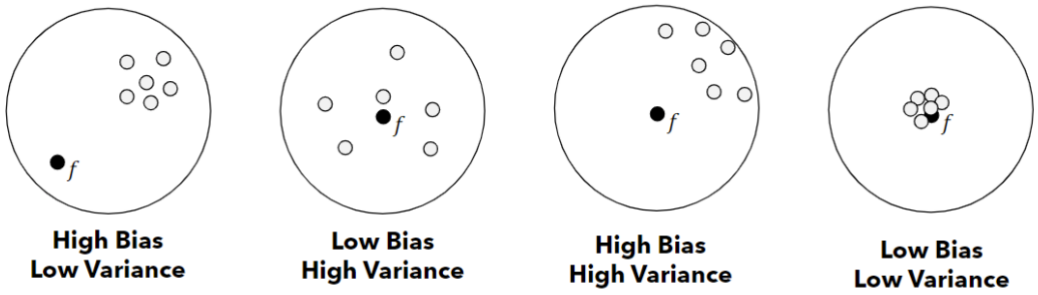
What is the bias-variance tradeoff?

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.

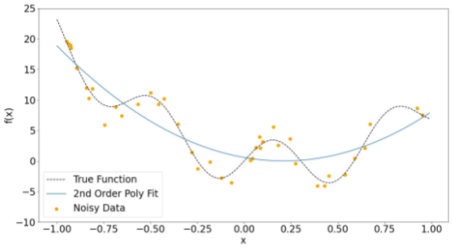


High Bias
Low Variance

Low Bias
High Variance

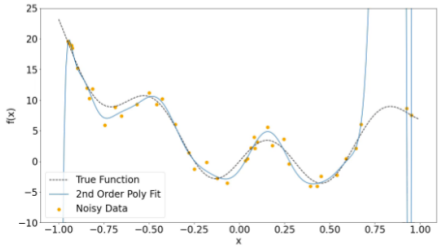
High Bias
High Variance

Low Bias
Low Variance



2nd Order Polynomial Fit:
Low variance / high bias

20th Order Polynomial Fit:
High variance / low bias



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!

What is an ensemble?

Ensemble method: combination of multiple classifiers outputs to produce a more accurate 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:

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

How do ensembles reduce error?

<u>If you combine decisions by models, you are more likely to get an accurate answer.</u>
Shown by the formula above that is essentially a binomial where:
M: Total number of models in the ensemble
k: Number of models making an error (in the range from ceil(M/2) to M)
P: Probability of a single model making an error
Q: Probability of a single model making a correct prediction (Q = 1 - P)
binom(M, k): Binomial coefficient, representing the number of ways to choose k models out of M

If ensembles reduce error so well, why doesn't everyone use them? **They do.**

Almost all challenges are won with ensemble methods.

Netflix Prize Challenge -- $1,000,000 for a 10% gain over Netflix's model for predicting user ratings.

"

Table 5: Best results of single approaches and their combinations

Method/Combination	RMSE
MF	0.9190
NB	0.9313
CL	0.9606
NB + CL	0.9275
MF + CL	0.9137
MF + NB	0.9089
MF + NB + CL	0.9089

-- Gravity (3)

"

What is bagging?

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.

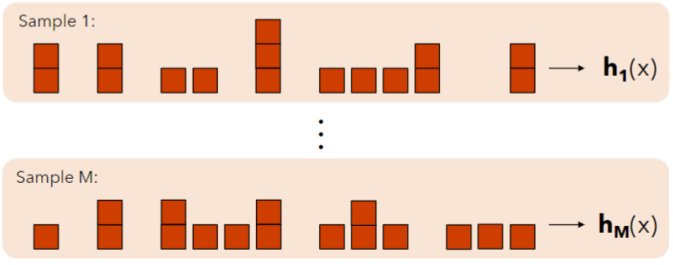
Bagging: Try to reduce variance in strong learners
- Uncorrelated errors → expected error goes down
- On average, do better than single classifier!

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!

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: 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 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

What are random forests?

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.

What is boosting?

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!

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?

Consider a regression setting where we are minimizing the squared error.

We are going to learn a sequence of models $h_1, h_2, \dots, h_M$ and will combine them with weights $\alpha_1, \alpha_2, \dots, \alpha_M$ such that our final model can be expressed as:

$$\hat{y} = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \cdots + \alpha_M h_M(x)$$

We would like to find models and weights that minimize error here:

$$\underset{\substack{h_1,\dots,h_M \\ \alpha_1,\dots,\alpha_M}}{\operatorname{argmin}} \sum_i (y_i - \alpha_1 h_1(x_i) - \alpha_2 h_2(x_i) - \cdots - \alpha_M h_M(x_i))^2$$

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:

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

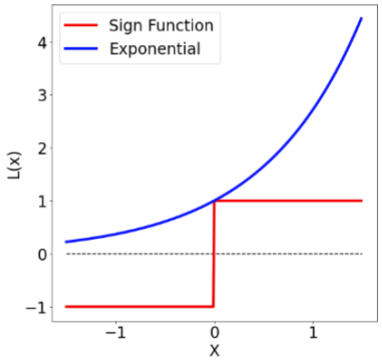
Easy to implement, just replacing y_i with $e_i^{(m)}$ in your favorite regression algorithm (e.g. regression trees).

Example with ADA boost

Consider a classification setting where we are minimizing errors (e.g., perceptron) where the class labels are {-1, 1}. We will do this by minimizing the exponential loss.

$$\underset{\alpha_1 \ldots, \alpha_M, h_1, \ldots, h_M}{argmin} \sum_i e^{-y_i(\alpha_1 h_1(x_i) + \cdots + \alpha_M h_M(x_i))}$$

Exponential loss forms an upper bound on the sign function. Datapoint is an error if $-y_i(\alpha_1 h_1(x_i) + \cdots + \alpha_M h_M(x_i)) > \mathbf{0}$, so we are minimizing errors by minimizing this loss.



Example: AdaBoost (Classification)

Let's assume we've trained the first model $h_1(x)$. How can we train the next model to improve over this?

$$\alpha_2, h_2 = \underset{\alpha, h}{argmin} \sum_i e^{-y_i(\alpha_1 h_1(x_i) + \alpha h(x_i))}$$
$$= \underset{\alpha, h}{argmin} \sum_i e^{-y_i \alpha_1 h_1(x_i)} * e^{-y_i \alpha h(x_i)}$$

Let $w_i^{(k)} = e^{-y_i(\alpha_1 h_1(x_i) + \ldots + a_k h_k(x_i))}$

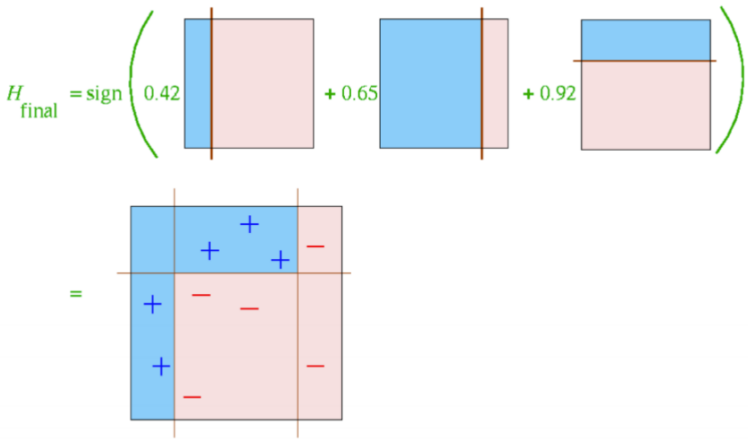
$$\alpha_m, h_m = argmin \sum w_i^{(m-1)} e^{-y_i \alpha h(x_i)}$$

Example: AdaBoost (Classification)

Adaboot Algorithm (Training):

1. Initialize all importance weights a $w_i^{(0)} = 1$
2. For m in range(M):
    1. Train classifier h_m to minimize weighted exponential loss $\sum_i w_i^{(m-1)} e^{-y_i h_m(x_i)}$
    2. Compute the weighted error rate of the classifier as: $\epsilon_m = \frac{\sum_i w_i^{(m-1)} I[y_i h_m(x_i) > 0]}{\sum_j w_j^{(m-1)}}$
    3. Compute classifier quality as $\alpha_m = \frac{1}{2} \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
    4. Update weights $w_i^{(m)} = w_i^{(m-1)} * e^{-y_i \alpha_m h_m(x_i)}$

The blocks are all different Hm where they classify weights. You combine them all at the end.



Loss Name	Loss Formula	Boosting Name
Regression: Squared Loss	$(y - f(x))^2$	L2Boosting
Regression: Absolute Loss	$\lvert y - f(x) \rvert$	Gradient Boosting
Classification: Exponential Loss	$e^{-yf(x)}$	AdaBoost
Classification: Log/Logistic Loss	$\log\left(1 + e^{-yf(x)}\right)$	LogitBoost

enerally, will assume you can train a model on a weighted dataset.

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.

Weekly Concept Quiz:

Decision trees have a tendency to overfit training data. What are options for reducing overfitting?

- [x] Removing nodes based on validation performance from the tree after training has finished
- [x] Limiting the depth of the tree
- [x] Monitor the validation accuracy and perform early stopping

All three are valid ways to control the complexity of a decision tree and can reduce overfitting.

Explain how decision trees can deal with continuous valued attributes.

Your Answer:

Split values based on ranges

Even though continuous valued attributes have an infinite number of possible thresholds, only a finite set of thresholds need to be considered during training because thresholds occurring between the same datapoints will have identical purity measures like Information Gain. As such, we can just consider thresholds at the midpoint between sequential datapoints for an attribute.

Decision tree learning can terminate whenever all attributes result in zero information gain.

- (•) True
- ( ) False

False. We discussed this as a potential base case; however, many functions (like XOR) may not show any information gain in initial variables but be able to be usefully split later only after multiple variables have been considered.

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

Variance -> Error due to sensitivity to changes in the dataset (aka estimation and optimization error)

Weak Learner -> Models that tend to have high bias but low variance.

Strong Learner -> Models that tend to have low bias but high variance.

Given a training dataset {a, b, c, d, e}, show three example bootstrap training sets that could be used for training bagged models.

Your Answer:

idk

Each training dataset for bagging is a sample with replacement from the original dataset with the same number of instances. So I drew a random number between 1 and 5 five times to make each dataset below:

{d, d, b, a, d}

{b, d, a, d, b}

{a, c, e, b, a}

Bagging is an ensemble method for reducing the variance of weak learners by combining multiple models trained on random samples of the dataset.

- True
- False

False. Bagging is used to reduce variance in **strong learners**. Weak learners already have low variance, but high bias.

Boosting is a method to make an ensemble of weak learner stronger -- i.e., a way to reduce bias in weak learners. Boosting typically trains a sequence of models with each model focusing on the errors of the previous models.

- True
- False

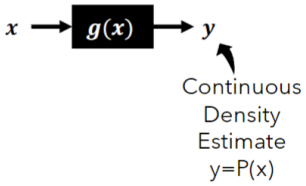
True! We talked about two versions of this -- L2 boosting and Adaboost

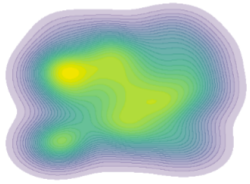
L9.1 - Unsupervised Learning I

What is unsupervised learning and how does it differ form supervised learning?

Unsupervised Learning
Only given a set of input instances (x)

Density Estimation: Estimate the underlying distribution generating our data



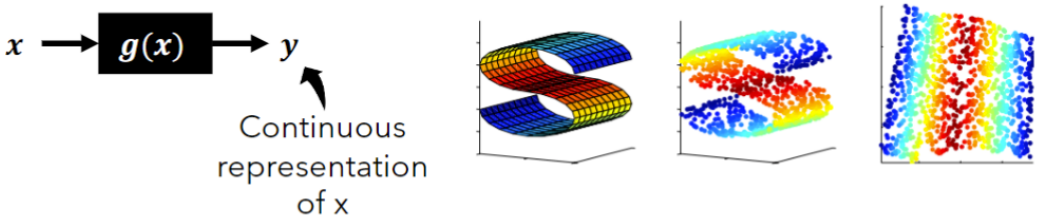


Example: Estimate how likely is a given house with these properties

It differs from supervised learning because there are no input labels.

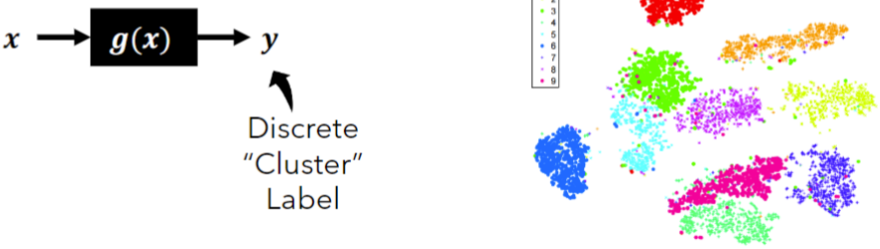
Unsupervised Learning
Only given a set of input instances (x)

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



Unsupervised Learning
Only given a set of input instances (x)

Clustering: Discover self-similar groups within the data



Example: Find groups of houses

What is clustering?

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

How does k-means clustering work?

Conceptual Overview of k-Means:

- Start with random initial points to represent the center of the k clusters.
- Form initial clusters based on these random starting points.
- Iteratively refine these clusters and stop when no longer making changes.

Hyperparameters:

- k – the number of clusters
- Some distance function, we'll assume Euclidean (aka L2) for now

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

Formalization:

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 $\boldsymbol{c}_j$:

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

Overall computational complexity is therefor O(mknd)

- linear in relevant inputs (for fixed iteration count)

What is k-means optimizing?

Claim: k-Means is minimizing the sum of squared error between points and their associated cluster centroid. That is, the optimal centroids and assignments are:

$$C^*, Z^* = \underset{C,Z}{\text{argmin}} \; SSE(X, C, Z) = \underset{C,Z}{\text{argmin}} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{c}_{z_i}\|_2^2$$

This is an optimization over two sets of variable – assignments and centroids – and mixes discrete / continuous variables. Very hard in general.

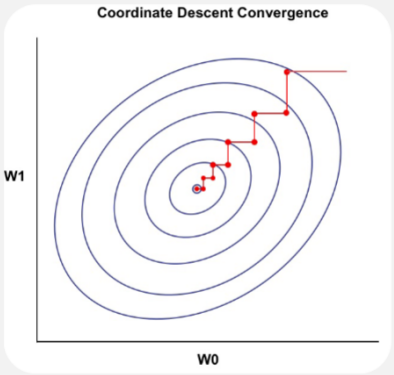
How do we get from this problem to the 2-step process we defined before?

Suppose we have some function $f(w_1, w_2)$ we want to minimize.

Coordinate Descent would alternative between the following steps:

1. Fix w_2 as a constant and optimize w_1
2. Fix w_1 as a constant and optimize w_2

Why do this? Sometimes functions have closed-form / easy solutions in some variables if the others are fixed.



Does it converge to an optima? Yes for convex, differentiable f. Not generally.

What is a coordinate descent algorithm?

Coordinate Descent on SSE. Will alternate between solving for C and Z that minimize SSE with the other fixed.

$$SSE(X, C, Z) = \sum_{i=1}^{n} \| \boldsymbol{x}_i - \boldsymbol{c}_{z_i} \|_2^2 = \sum_{i=1}^{n} \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t}\right)^T \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t}\right)$$

Minimize objective with assignments Z^t fixed from previous round:

$$C^{t+1} = \underset{C}{\text{argmin}} \sum_{i=1}^{n} \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t}\right)^T \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t}\right)$$

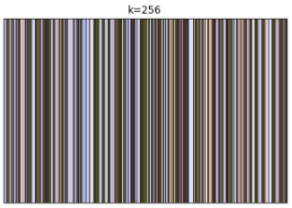
Minimize objective with centroids C^t fixed from previous round:

$$Z^{t+1} = \underset{Z}{\text{argmin}} \sum_{i=1}^{n} \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i}^{t+1}\right)^T \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i}^{t+1}\right)$$

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction

Only store id of centroid at each pixel. Store the centroids as well.

K initialization for k-means clustering:

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.

Algorithm to reduce outlier sensitivity:

Properties: Sensitive to Outliers

Rough Idea of k-medoids Algorithm: Instead of taking the mean in the centroid update step, take the median – i.e., the data point that is closest to the other points in the cluster.

$$c_j = \frac{1}{\sum_{i=1}^{n} \mathbb{I}[z_i = j]} \sum_{i=1}^{n} \mathbb{I}[z_i = j]\ x_i \quad \longrightarrow \quad c_j = \underset{z \in M_j}{\operatorname{argmin}} \sum_{x_i \in M_j} \|x_i - z\|_2^2$$

K-Medoids is computationally more expensive but more robust to outliers.

What is a Gaussian Mixture Model?

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.

Given: a dataset $D = \{\boldsymbol{x}_i\}_{i=1}^n$ where $x \in \mathbb{R}^d$

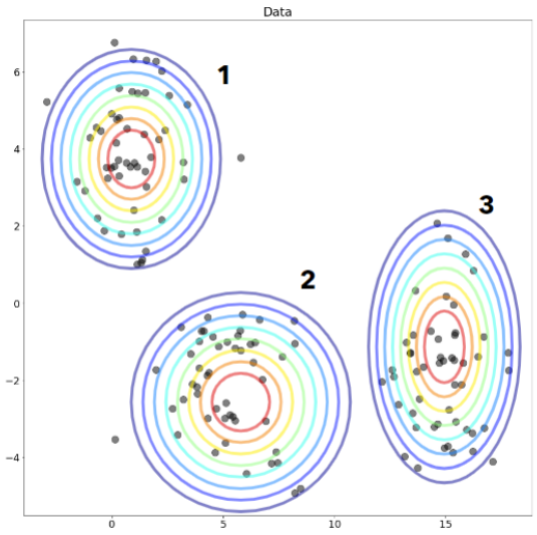
Goal: Fit the distribution $P(\boldsymbol{x})$

But this seems like a weird distribution… what if I approximate it as multiple Gaussians?

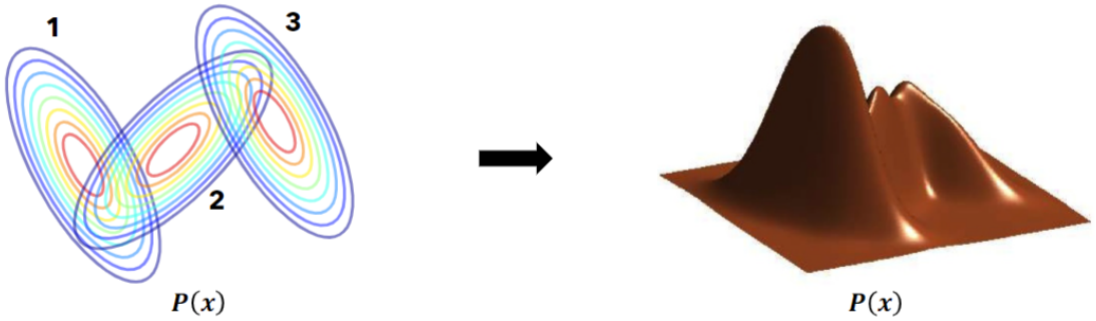
An Idea: Assume the following generative story of how this data was created.

Assume each point was generated by:

1. Sampling one of the three clusters, then
2. Sampling a point from a Gaussian defining that cluster



$$P(\boldsymbol{x}) = 0.5 * \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_1}, \Sigma_1) + 0.25 * \mathcal{N}(\mathbf{x}; \boldsymbol{\mu_2}, \Sigma_2) + 0.25 * \mathcal{N}(\mathbf{x}; \boldsymbol{\mu_3}, \Sigma_3)$$



Gaussian Mixture Models (GMM)

Given: a dataset $D = \{\boldsymbol{x}_i\}_{i=1}^{n}$ where $x \in \mathbb{R}^d$

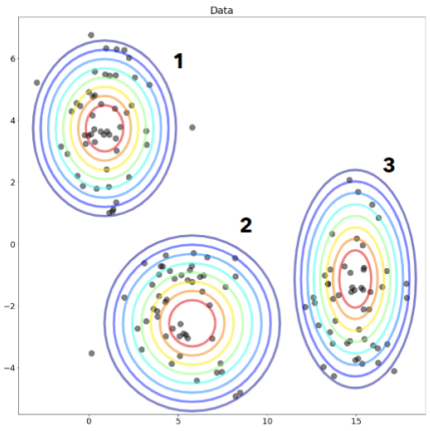
Goal: Fit the distribution $P(\boldsymbol{x})$ as a Gaussian Mixture Model with k components.

Formalizing our generative story: Assume each point x_i was generated by the process:

$$z_i \sim P(z)$$

$$x_i \sim \mathcal{N}(x; \mu_{z_i}, \Sigma_{z_i})$$

Call the z's "latent or hidden variables" as they are not observed from our data and must be inferred.



Need to learn:

- The means and covariances of the k Gaussians.
- The Categorical distribution P(z) over the k Gaussians.

The General Case – EM for GMM

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 $\theta_1, \ldots, \theta_k$ all to 1/k
means of the Gaussians $\mu_1, \ldots, \mu_k$ to random points
covariances of the Gaussians $\Sigma_1, \ldots, \Sigma_k$ 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

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

Fraction of mass assigned to c

Weighted mean of fractional points assigned to c

Weighted covariance of fractional points assigned to c

What can this tell us about k-Means?

How to get k-Means back from GMMs:

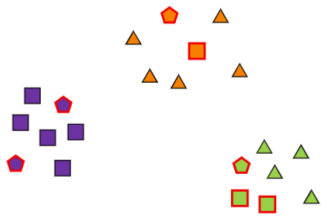
- Assume hard-assignment rather than fractional.
- Assume all Gaussians have the same isotropic covariance.

How to get k-Means back from GMMs:

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

How do we evaluate clustering?

- **Purity** – Fraction of points that would be correctly classified by a "majority vote" per cluster where all points get the label of the majority.



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

L9.2: Unsupervised Learning II

What is Hierarchical Agglomerative Clustering(HAC)?

Two Categories of Clustering Algorithms

Partition Algorithms ("Flat" Clusterings)
- k-Means / k-Medians
- Gaussian Mixture Models

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

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?

Summarizing Hierarchical Agglomerative Clustering

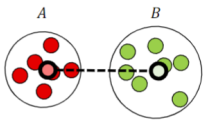
- HAC is a convenient tool that often provides interesting views of a dataset
- Primarily HAC can be viewed as an intuitively appealing clustering procedure for data analysis/exploration
- We can create clusterings of different granularity by stopping at different levels of the dendrogram
- HAC often used together with visualization of the dendrogram to decide how many clusters exist in the data
- Different linkage methods (single, complete and average) often lead to different solutions

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 $d(x, y)$:

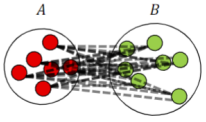
Centroid
- The distance between the cluster means

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



Average-link
- Average distance between all cross-cluster pairs

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



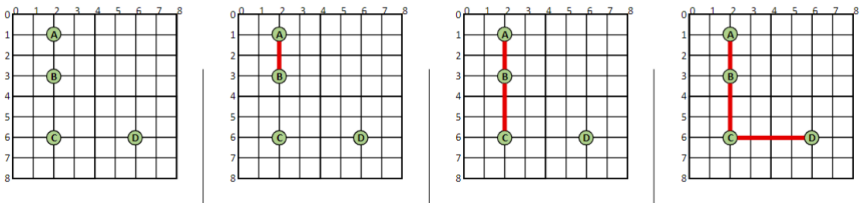
- **Most robust and most commonly used**

What are different linking strategies?

Single-Link Method Example (L1 distance for simplicity)

Single-link
- The distance between the nearest points

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

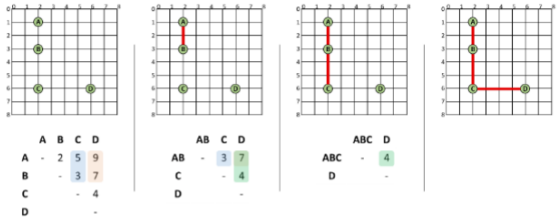


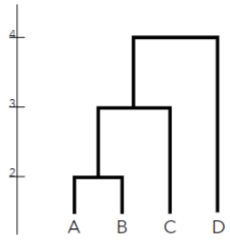
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-

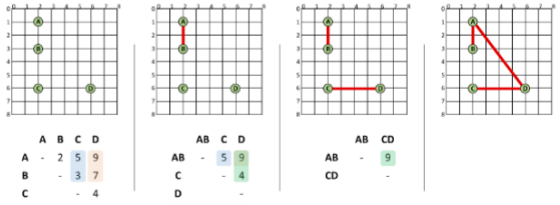
	AB	C	D
AB	-	3	7
C		-	4
D			-

	ABC	D
ABC	-	4
D		-

Visualizing Hierarchical Clusterings: Dendrogram



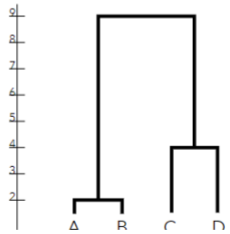


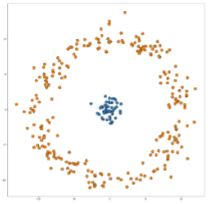


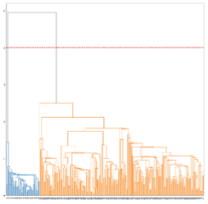
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-

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

	AB	CD
AB	-	9
CD		-







Single-link has a "chaining effect"

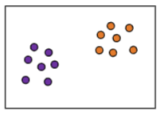
- Famous for its ability to gradually add more and more examples to a cluster
- Creates long, straggling cluster that are super evident in the dendrogram

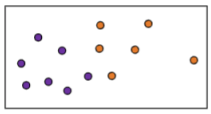
How can we evaluate clustering?

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.

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.

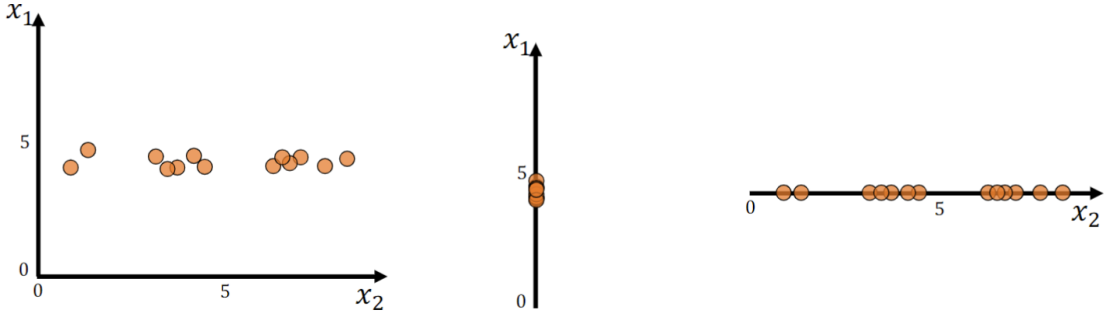
What is dimensionality reduction?

New Topic: Dimensionality Reduction

Unsupervised Learning
Only given a set of input instances (x)

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

This is an example of how you could split 2-d to 1-d based on the two axis



What is principal component analysis (PCA) and how does it work?

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

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!

HOW TO IMPLEMENT:

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 $\frac{1}{n} X^T X$
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

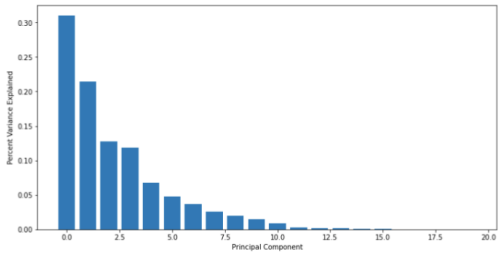
After this class is over: Just call a PCA API, they do basically the same thing.

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:

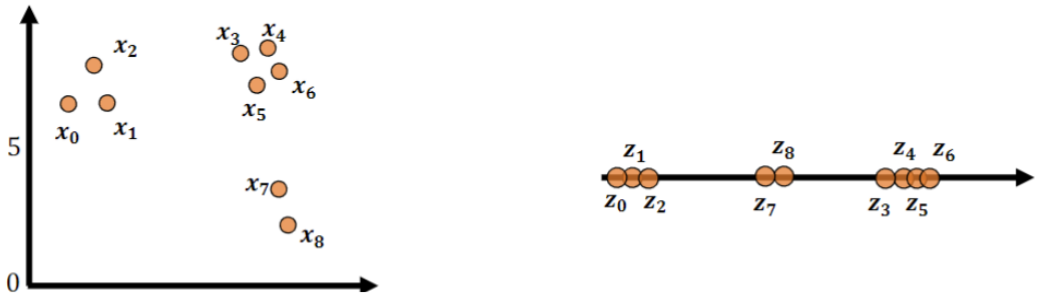


What are stochastic neighbor embeddings (SNE) and how do they work?

Non-Linear Dimensionality Reduction: t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Idea: For each **d-dimensional** input vector 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.



Use KL-Divergence to measure differences between these distributions:

$$KL(p_i \,||\, q_i) = \sum_j p_{i|j} log_2 \left(\frac{p_{i|j}}{q_{i|j}}\right)$$

Let our loss be the sum over all these distributions:

$$loss = \sum_i KL(p_i \,||\, q_i) = \sum_i \sum_j p_{i|j} log_2 \left(\frac{p_{i|j}}{q_{i|j}}\right)$$

Optimization time! How do we find low-dimensional vectors $z_0, z_1, \ldots, z_n$ that minimize this loss.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Hyperpamameters:

- The number of lower dimensions **k**
- The sigma's for each datapoint:
  - No good way to pick a global sigma for all points to use
    - Potentially different density of points in different regions of the input space
  - t-SNE introduces a "**perplexity**" hyperparameter to implicitly control sigma's
    - Bigger perplexity leads to bigger sigma's and more neighbors considered.
    - Each point's sigma is independently optimized to match the perplexity.
- Distance functions can also be changed

$$p_{i|j} = \frac{e^{\frac{\|x_i - x_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{\frac{\|x_i - x_k\|^2}{2\sigma_i^2}}}$$

WEEK 9 Questions:

Which of the following are tasks in unsupervised learning?

- [x] Clustering
- [ ] Density Estimation
- [x] Dimensionality Reduction
- [x] Classification
- [ ] Regression

There are two main categories of clustering algorithms -- hierarchical and partitional. Partitional clustering algorithms like k-means assign each datapoint to one of a finite set of clusters. Hierarchical algorithms like Hierarchical Agglomerative Clustering (HAC) iteratively merge or divide clusters in order to produce a hierarchy where each point belongs to a set of clusters.

- (•) True
- ( ) False

Centroid initialization is not important to k-means because k-means always converges to the global minima of sum-of-squared error.

- (•) True
- ( ) False

False. The k-mean algorithm is not guaranteed to reach the global minima of SSE, just a local one. As such, k-means is sensitive to which initialization.

What is the stopping criteria for the k-means algorithm?

Your Answer:

idk

k-means is guaranteed to converge within a finite number of steps. Convergence occurs when the assignments don't change across an iteration.

Note that many implementations also consider a maximum number of iterations or the a percentage-assignments-changed criteria.

The k-means algorithm is derived by attempting to minimize the sum-of-squared error between centroids and assigned datapoints using coordinate descent.

- (•) True
- ( ) False

True. See the derivation in the class slides.

Why is it a bad idea to choose *k* that minimizes SSE on a dataset?

Your Answer:

idk

SSE decreases with larger *k* and thus doesn't tell us much about the quality of the clustering.

k-Means - A partitional clustering algorithm that assigns each point to one cluster by iteratively updated centroids.

Gaussian Mixture Model (GMM) - A probabilistic model of data that assumes the data was generated by a set of Gaussian distributions. Parameters for these distributions are found via MLE using the Expectation-Maximization algorithm.

Hierarchical Agglomerative Clustering (HAC) - A clustering algorithm that builds a hierarchy of clusters by iteratively merging the closest clusters according to some "link function" that measures cluster similarity.

Principal Component Analysis (PCA) - A dimensionality reduction that finds linear subspaces that capture the variance of high dimensional data.

False Statements:

PCA ensures points belonging to different classes are well-separated in the projection. -- PCA is an unsupervised approach and does not consider class labels.

PCA works well for data with non-linear subspaces. -- Standard PCA only considers linear subspaces.

Which of the following are true statements about Principal Component Analysis (PCA)?

- [ ] PCA is derived by maximizing the variance of data after being projected to a linear subspace.
- [x] The solution to the PCA objective reduces to an Eigenvector decomposition.
- [ ] PCA works well for data with non-linear subspaces.
- [x] PCA transforms high dimensional data into low dimensional representations.
- [x] PCA ensures points belonging to different classes are well-separated in the projection.

Complete-link HAC is able to generate very long clusters because it only considers the minimum distance between points in two clusters for merging.

- (•) True
- ( ) False

False. Complete-link considers the maximum distance between points in two clusters. The question would describe Single-Link.

The height of a joint in a Dendrogram reflects _____,

clustering

The height of a joint is the distance between two merged clusters.

L10.1 - KDE and Reinforcement Learning.pdf

What is Kernel Density Estimation?

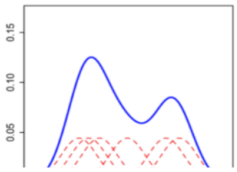
KDE is one approach to estimating a probability density given some data.

- Recall, a density function describes the likelihood of sampling points from given parts of space.

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)



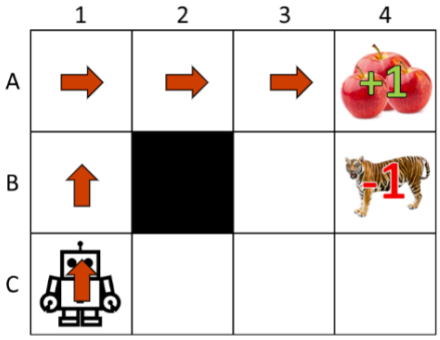
In this 1D example:

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

What is a sequential decision problem?

In **sequential decision making tasks**, our algorithms will need to make a **sequence** of correct predictions in order to be successful. Further – inputs our algorithms receive will change based (in part) on the sequence of actions that have been taken thus far!

Let's start thinking about sequential decisions using this simple grid world.
(The same ideas apply to more abstract things than motion like "when to raise or lower control rods in a nuclear reactor")



- Assume a fully-observable, deterministic environment shown to the left.
- Each grid cell is a **state**
- At each time step, agent can take **actions**: Up, Down, Left, Right
- Positive **reward** for getting apples, negative for entering the tiger cell.

What is a Markov Decision Process?

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

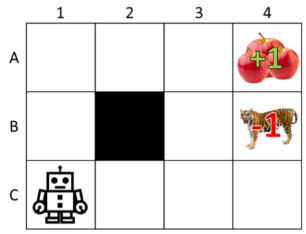
- $\mathcal{S}$ Set of possible **states**
- $\mathcal{A}$ Set of possible **actions**
- R: $\mathcal{S} \rightarrow \mathbb{R}$ **Reward function** - mapping between states and rewards
- T: $\mathcal{S} \times \mathcal{A} \rightarrow \Omega_{\mathcal{S}}$ **Transition** function - mapping between state-action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

"Markov" assumption says transitions depend only on current state

$$P(s_{t+1}|a, s_0, \dots, s_t) = P(s_{t+1}|a, s_t)$$

If the agent takes the "Right" action at s_2, the probability of arriving at $s_3 = A2$ only depends on the fact that the agent is at s_2 and not the entire history $\{s_0, s_1, s_2\}$

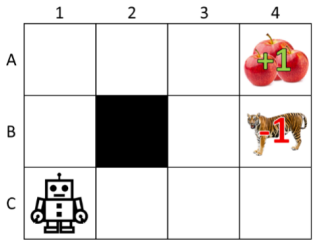
What is a state?



States $\mathcal{S}$:

$\mathcal{S} = \{A1, A2, A3, A4, B1, B3, B4, C1, C2, C3, C4\}$

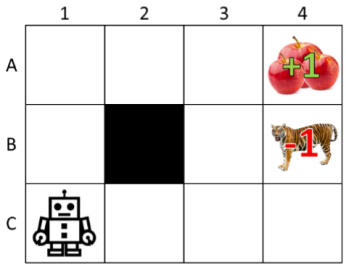
What is a reward?



Reward Function $R: \mathcal{S} \rightarrow \mathbb{R}$:

$R = \{A4 = 1, B4 = -1, Others = -0.0001\}$

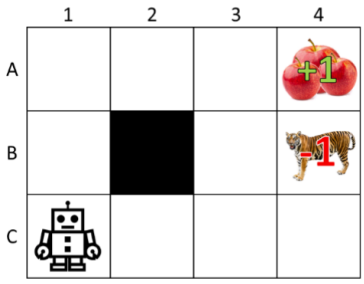
What is an action?



Actions $\mathcal{A}$:

$\mathcal{A} = \{Up, Down, Left, Right\}$

What is a transition function?



Transition Function $T: \mathcal{S} \times \mathcal{A} \rightarrow \Omega_{\mathcal{S}}$ **:**

$T(s, a, s')$ is probability of arriving at state s' after performing action a in state s

$$T(C1, Up, C2) = 0.1$$
$$T(C1, Up, B1) = 0.8$$
$$T(C1, Up, A2) = 0$$
$$\vdots$$

Random stuff about Markov (idk)

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.

More about **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.

- Describes preference for current reward over future reward
  - **Biology:** Compensates for uncertainty in available time (model mortality)
  - **Financial:** Money gained today can be invested and make more money

2. What do we mean by policy "performs well"?

How much reward can we *expect* a policy to collect on average if we run it many times?

- Each time we run the policy, will get a trajectory of states. However, which precise trajectory is random due to the environment (and/or the policy if it is stochastic).
  - Can write a state sequence s that comes from running π as $s \sim \pi$

Write expected discounted reward as:

$$\mathbb{E}_{s\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

Optimal policy π^* is the policy that maximizes expected discounted reward:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{s\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

1. **What is a behavior policy?**

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

2. **What do we mean by "performs well"?**

   Expected discounted reward of running the policy in the environment.

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

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 the above techniques to find best policy.

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

What reinforcement learning generally?

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

Problems:

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 2.08×10170 legal board configurations!

What are policy gradient methods?

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.

REINFORCE Algorithm

(Williams, 1992)

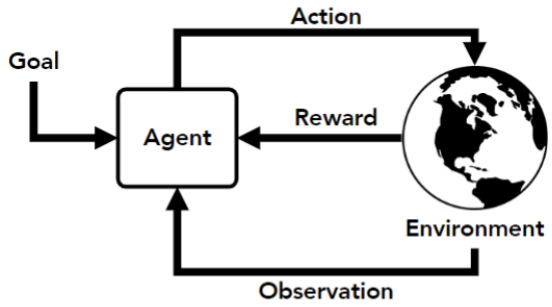
While not converged:

1. Run policy to collect trajectories $\tau_j = (s_{j0}, a_{j0}, s_{j1}, a_{j1}, s_{j2}, \dots)$ and reward $r(\tau_j)$
2. Compute gradient estimate $\nabla_\theta J(\theta) \approx \sum_j \sum_{s_{ji}, a_{ji} \in \tau_j} \nabla_\theta \log \pi(a_{ji}|s_{ji}; \theta)\, r(\tau_j)$
3. Update policy parameters $\theta' = \theta + \alpha \nabla_\theta J(\theta)$

What is imitation learning?

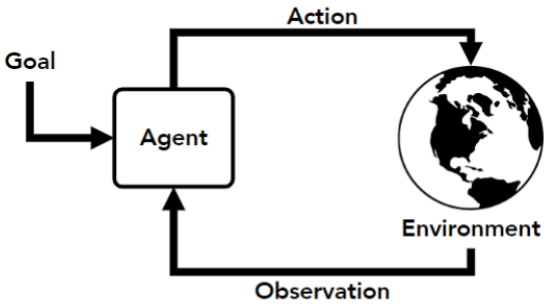
Reinforcement Learning

- Environment provides feedback
- No examples of optimal policy



Imitation Learning

- Have expert demonstrations (possibly interactive)



What is behavior cloning?

Behavior Cloning

- Given dataset of trajectories $D = \{ (s_0, a_0, s_1, a_1, \dots, s_T, a_T)_i \}_{i=1}^N$ from an expert demonstration policy π^*
- Break things down to individual state-action pairs s_t, a_t and directly train a policy $\hat{a}_t = \pi(s_t)$ using supervised learning:

$$\theta^* = argmin_\theta \mathbb{E}_{s \sim \pi^*}[\ L(\pi_\theta(s), \pi^*(s))\ ]$$

$$\theta^* = argmin_\theta \sum_i L(\pi_\theta(a_t|s_t), \pi^*(a_t|s_t))$$

- Interpretations:
  - Assuming perfect imitation so far, learn to continue imitating perfectly
  - Minimize 1-step deviation for states the expert visits

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.

How are sequential decision problems different from the standard supervised learning setting's we've discussed before?

Your Answer:

i

In sequential decision problems, our model's predictions affect the next observations our model receives and the long-term outcome of a sequence of decisions. In the standard IID supervised learning setting, this is not the case because each prediction is independent.

States - The set of possible configurations of the world

Actions - The set of things an agent can do

Transition Function - A mapping from state-action pairs to the resulting next state

Reward Function - A mapping from states to positive or negative feedback signals

In model-based reinforcement learning, our task is to learn _____. In model-free reinforcement learning, we want to learn _____.

transition fucntion, leanr policy to maximize reward

A behavior policy is a mapping from states to actions (or distributions over actions). It is essentially a plan for what to do in any given state. This can be a simple as a table that exhaustively lists every state and the corresponding action to take; or as complex as a deep neural network that maps a state representation to a distribution over possible actions.

transition and reward functions; behavior policies directly

What are two major problems in reinforcement learning we discussed in class?

Your Answer:

credit assignment, explore exploit

Credit assignment can be difficult because reward may come only after many actions and it is not clear which actions contributed most to the reward.

Balancing exploring new strategies vs. exploiting strategies that work well is another common challenge.