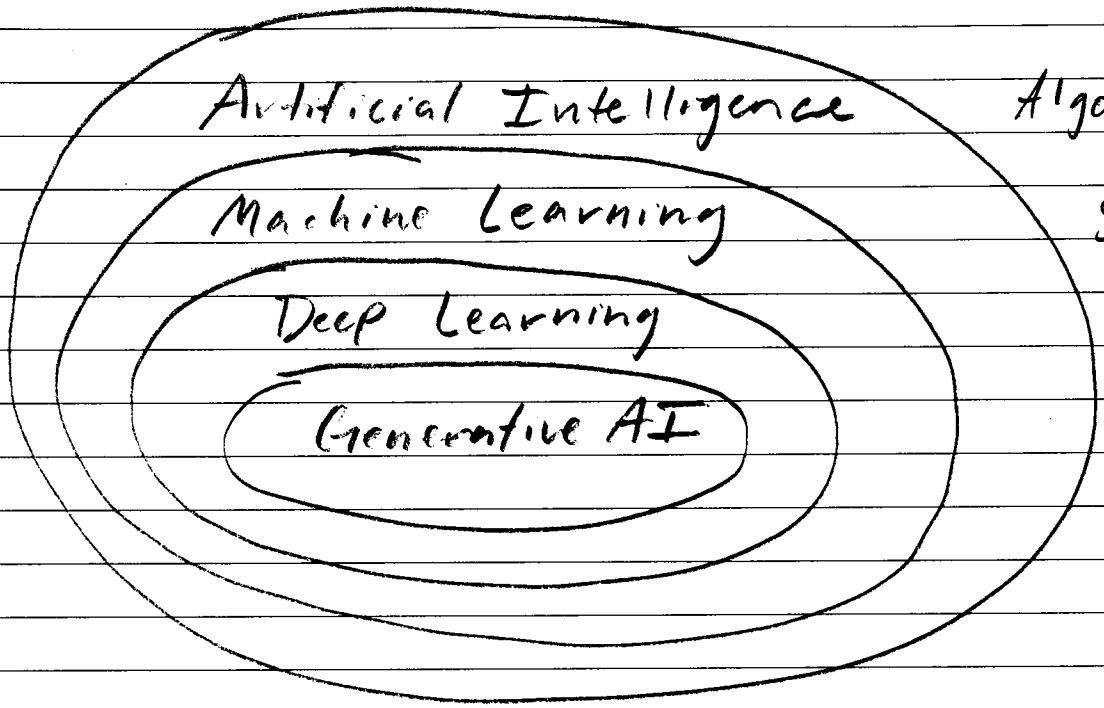
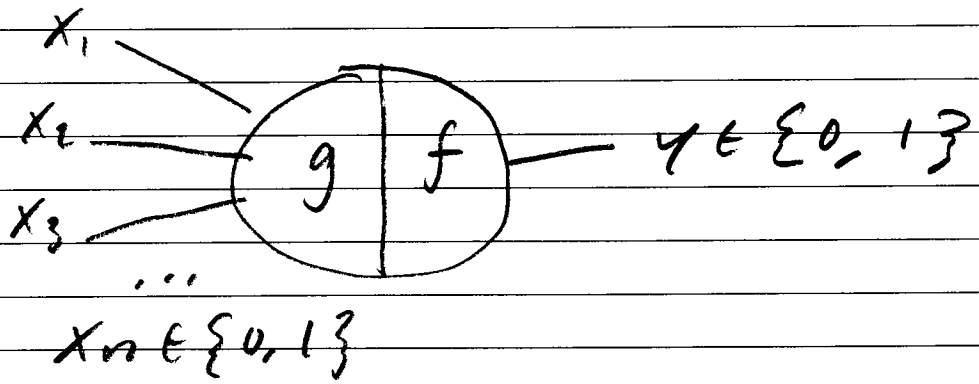
Intro to Artificial Intelligence

Date. 8.25.25
No. 1



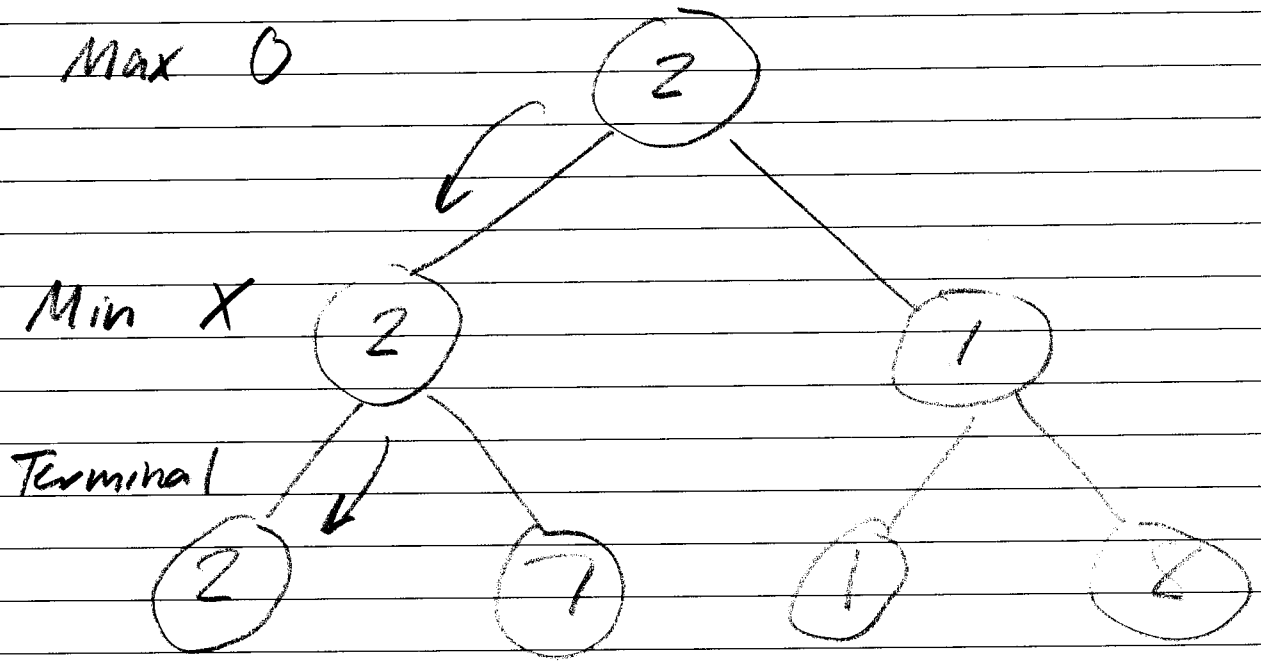
Artificial Neuron:



Heuristics: rule-based narrow approach to solving a problem

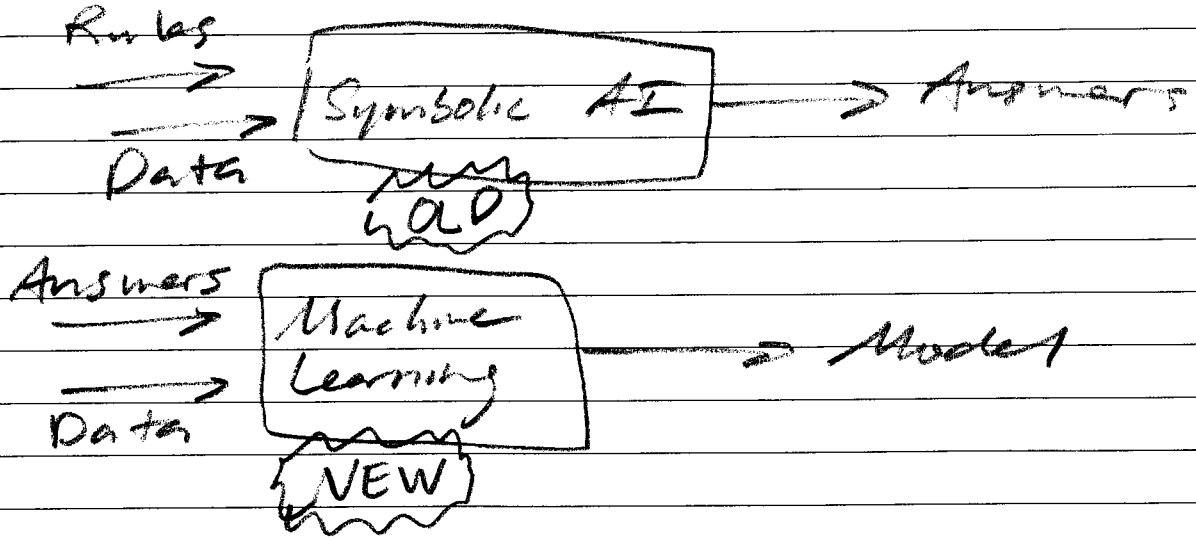
Date.
No.

The Minimax Algorithm:



* numbers are assigned to game states using heuristics

GOFAI: Good Old Fashioned AI

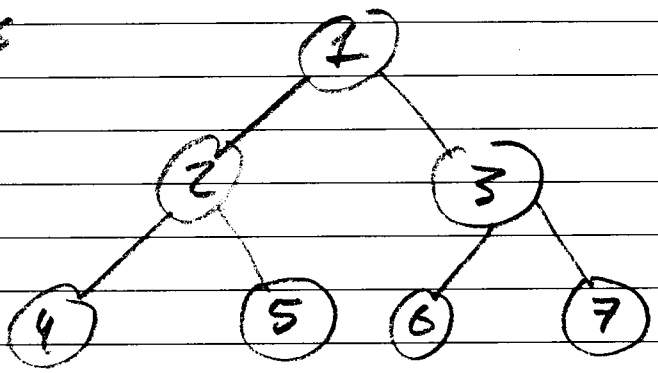


Search and Agents

Date. 8.27.25
No. 2

Traversal: Systematic visiting of each element in a data structure

Ex. BFS



BFS: Queue for frontier (FIFO)
DFS: Stack for frontier (LIFO)

BFS Pseudo-Code:

```
bfs_traversal(g, v):
    q = Queue()
    q.push(v)

    while q:
        n = q.pop()

        q.add(n.left)
        q.add(n.right)
```

Traversal vs. Search:

- Traversal accesses **every** node in DS
- Search looks for a **specific** node in DS

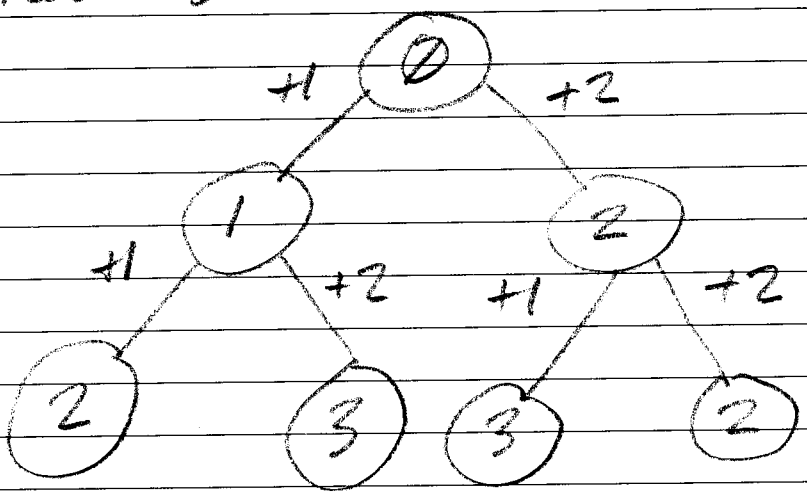
Date.
No.

Search in AI: (early AI)

- Reasoning step by step
- Symbol manipulation
- Computational limitations

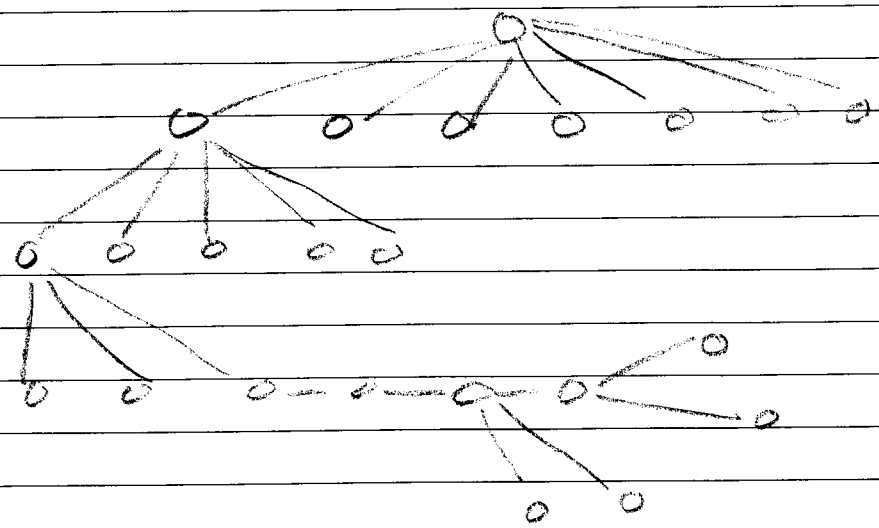
very explainable, unlike neural networks

Ex. Start at 0, can add either 1 or 2, try to reach 3



DFS is faster here than BFS here

Ex.



but here BFS would be better

Date.
No.

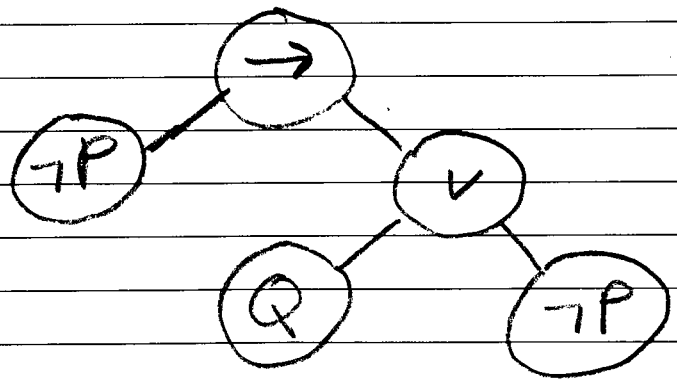
... but BFS takes up way more memory than DFS — more than people had in the 50s ...

Aside: DFS Pseudo-Code (search for a)

```
dfs-search(g, v, a):
    if v == a:
        return True
    if not v:
        return False

    return dfs-search(g, v.left, a)
    return dfs-search(g, v.right, a)
```

The Logical Theorist: Early AI that solved proofs in the Principia Mathematica using search algs

Ex. $\neg P \rightarrow Q \vee \neg P$



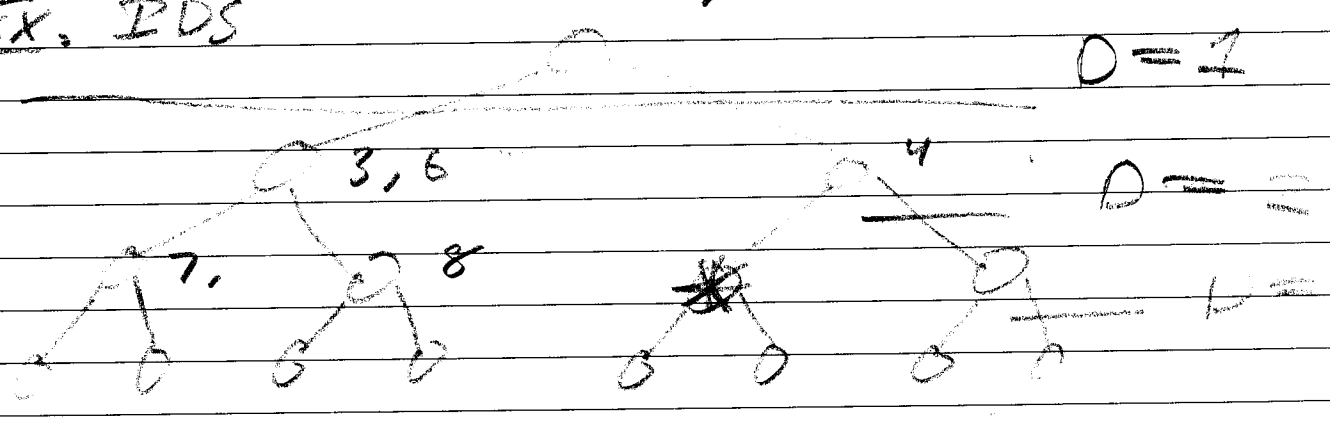
Date.
No.

Iterative-Deepening Search (IDS):

IDS combines DFS's space efficiency and BFS's fast search (for nodes closer to root)

IDDFS calls DFS for different depths starting from an initial value. In every call DFS is stopped at a certain depth

Ex. IDS



IDS Pseudo-Code: frontier still a stack

```
ids_search(g, v, a):
    for i in range(1, tree.height):
        if dfs_search(i, v, a):
            return True

    return False
```

Combinatorial Issues: This can't solve any real-world problems b/c of combinatorial explosion

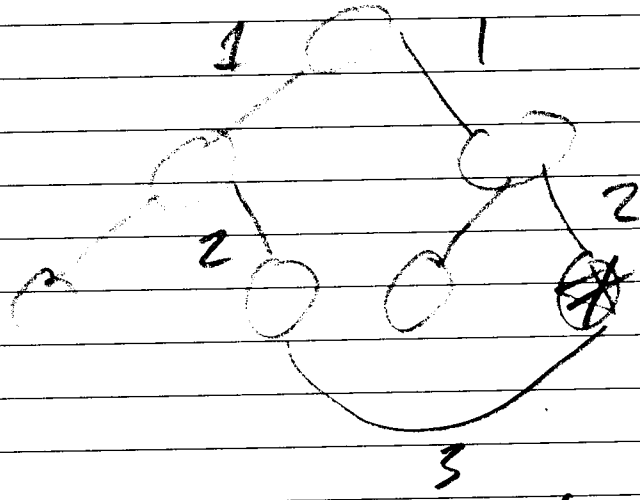
i.e. too many game states to enumerate

Informed Search and A*

Date. 9.7.25
No. 3

Pruning Branches: can remove branches of the tree that don't need to be explored which reduces the search space

Pathfinding: with search we don't care about how we find an answer. In pathfinding we do



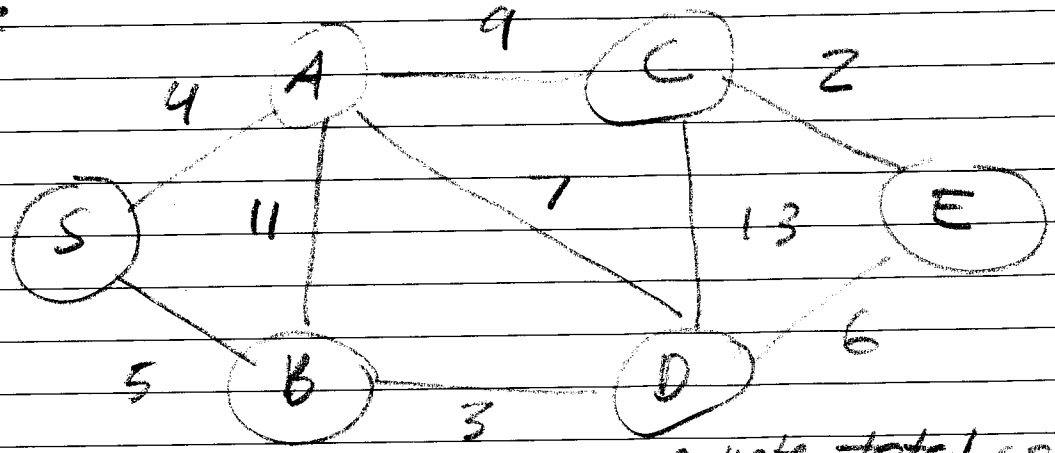
we want to find our target in a minimum amount of edges visited

Dijkstra's: Single-source shortest path algo

- determines the shortest path between two vertices.
- remove vertex from frontier like BFS/DFS
- to choose next vertex, select the "closest" node

Date.
No.

Ex.



note. total cost!

Steps:

~~S → A : 4~~ S → A → C : 13 S → B → A : 16
~~S → B : 5~~ S → A → D : 11 S → B → D : 8
S → A → B : 15

... and so on ...

Frontier Review:

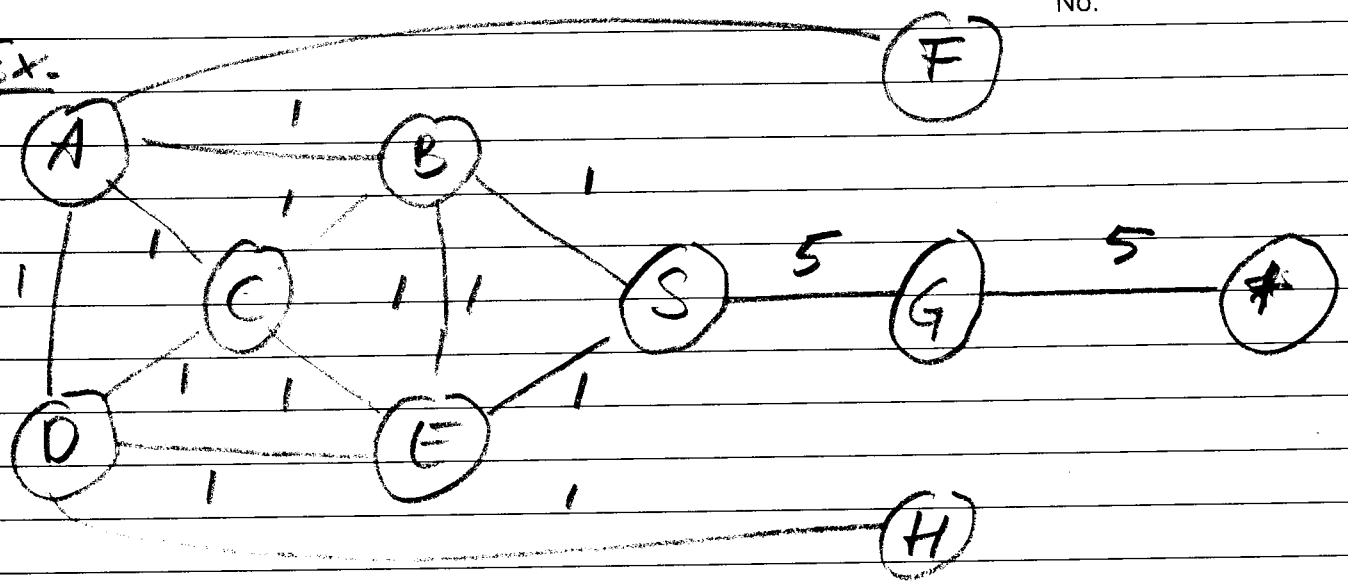
BFS → Queue
DFS → Stack
IDS → Stack
Dijkstra's → Min-heap (Priority queue)

Pseudo-Code:

```
Dijkstras(g, v):                          (n.weight, n.curr, n.prev)
    frontier = min_heap()
    frontier.push((0, v, v))
    visited = {}
    while frontier:                              if n = goal:
        n = frontier.pop()                           return n
        if n in visited:
            continue
        visited[n.curr] = n.prev
        for neighbor in g[n]:
            frontier.push(n.weight + edge_weight,
                          neighbor, n)
```

Date.
No.

Ex.



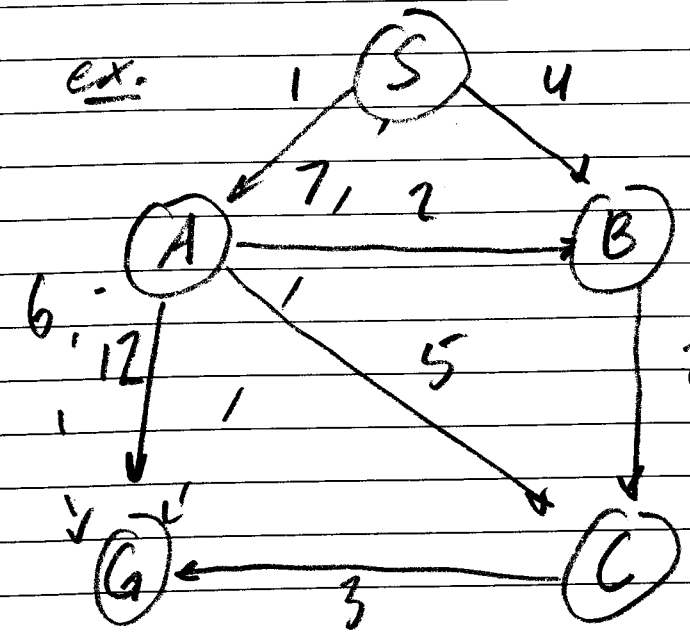
need a Heuristic! Dijkstra's would do really bad here.

Heuristic: a function that ranks alternatives in search algorithms at each branching step based on extra available information to decide which branch to follow

The A* Algorithm: graph traversal/pathfinding alg initially used today

uses heuristic information alongside dijkstras

ex.



N	h(N)
S	7
A	6
B	2
C	1
G	0

"best-guess from N to G"

Date.
No.

Ex cont. btw prev exam question

frontier. still a min-heap

Steps: → same as dijkstra's! → local only!

0) $N \to N' = d(N, N') + h(N', G)$
1) $S \to A = 1 + 6 = 7$ ✗ (2)
2) $S \to B = 4 + 2 = 6$ ✗ (1)
3) $S \to B \to C = 6 + 1 = 7$
4) $S \to A \to B = 3 + 2 = 5$ ✗ (3)
5) $S \to A \to C = 6 + 1 = 7$
6) $S \to A \to G = 13 + 0 = 13$
7) $S \to A \to B \to C: 5 + 1 = 6$ ✗ (4)
8) $S \to A \to B \to C \to G = 8$

Heuristic Qualifiers: needed for helpful heuristics

Admissible: heuristic cost never overestimates the actual cost

$\forall h(n) \leq d(n, G)$

Consistent: the heuristic cost from a node is less than or equal to the cost of reaching a neighbor and that neighbors heuristic cost

$\forall h(n) \leq d(n, n') + h(n', G)$

think triangle inequality!

A* needs admissable, optimal with consistent

Constraint Satisfaction Problems

Date. 9.03.25
No. 04

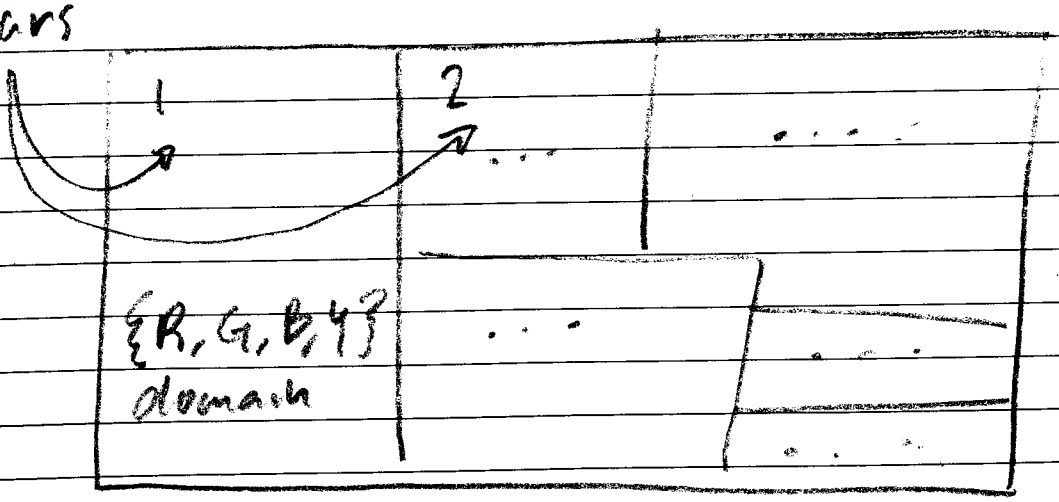
Constraint Satisfaction Problem (CSP): find a solution that satisfies all constraints

- These can be considered as a class of optimization / search problem

Applications:
- scheduling flights
- sudoku puzzles
- placing circuit board components
- murder mystery

Items of a CSP:

- Variables: something that can have an assignment
- Values: something that can be assigned to a variable
- Domains: The set of possible values a variable can hold
- Constraints: Limits on the values of (often pairs of) variables



Values: {R, G, B, Y}

Constraint: 1 ≠ 2, ...

Date.
No.

Arcs. more specific instantiation of constraints

Arc-Consistency: when enforcing arc-consistency, we look for values that can be removed when constraints are violated

If we've removed all such values, then everything is arc consistent

Ex. CSP for Sudoku

Variable: d_n - empty cell
Value: 1-9
Domain: 1-9 per cell
Constraints:
ALL-DIFF(Row)
ALL_DIFF(Col)
ALL-DIFF(3x3)

Outcomes of AC3:

1) at least one var has an empty domain
→ no solution exists

2) All variables have at least one value in their domains
→ a solution may exist

3) All variables have exactly one value in their domains
→ the solution exists and is found

Date.
No.

Arc-Consistency with AC3:

AC-3 is a pre-processing step we can perform prior to running a backtrack solution search for a CSP to further prune the search space

enforces arc-consistency among all vars

Backtracking Search:

Pseudocode.

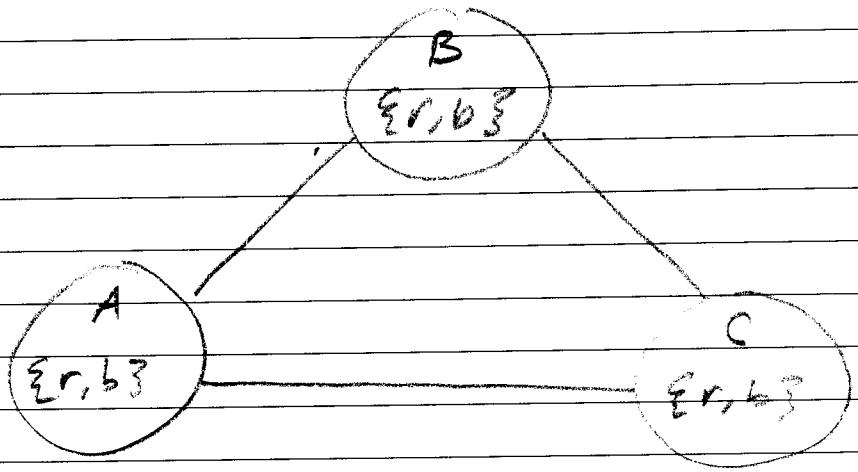
frontier : stack

```
backtrack(assignment):
    if assignment is_complete():
        return assignment

    variable = get_unassigned()
    for value in values:
        if valid((value, variable)):
            assignment += (value, variable)

            if backtrack(assignment):
                return True

            assignment -= (value, variable)
    return False
```

Date.
No.

Local vs. Global Consistency: Ex

Constraints:
$A \neq B$
$B \neq C$
$A \neq C$



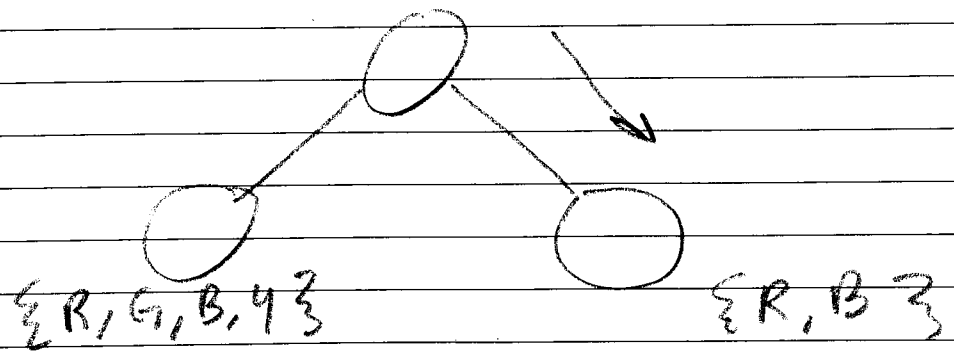
1) What does AC3 result in?
2) What does backtracking result in?

A. 1) wouldn't eliminate anything (maybe not)
2) would say there's no solution

Heuristics in CSPs:

Recall. Heuristic is extra human knowledge to help code

MRV Heuristic: Minimum Remaining Values



Date.
No.

Forward Checking:

Can improve backtracking through forward checking, where we can prune neighboring domains if their values would violate constraints

Maintaining Arc Consistency (MAC):

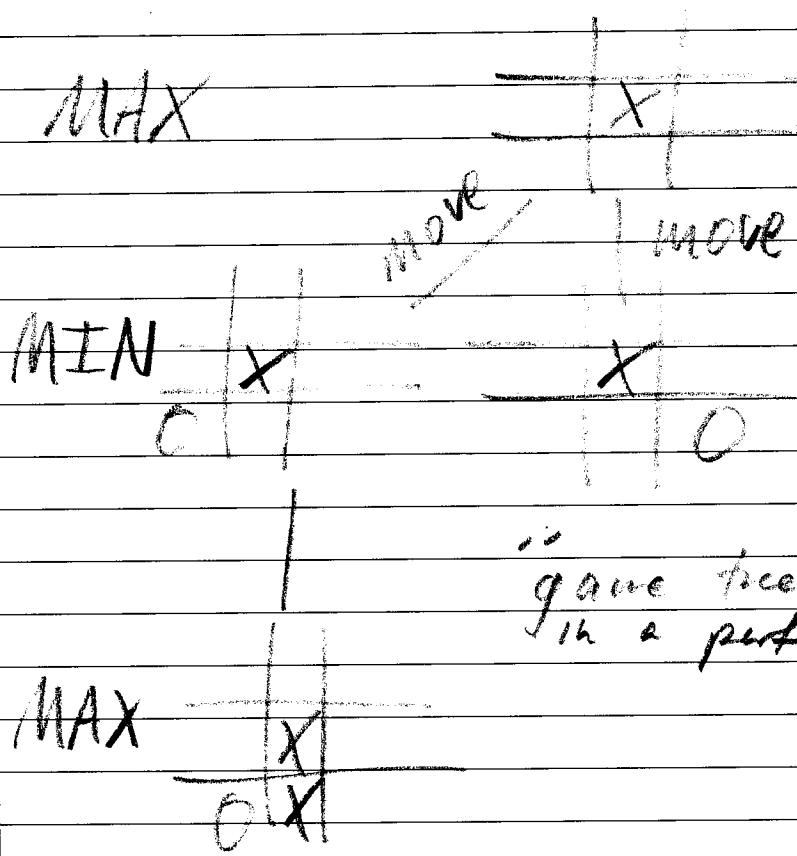
Constraint Propagation: prune all domains if they'd violate constraints

Forward Checking Steps

1. Consider Nothing
2. Check Assignment — Backtracking
3. Check Neighbors — Forward Checking
4. Check vars with 1 value
5. Check vars with reduced domain
6. Check everything

Date. 9/08/25
No. 05

Adversarial Search



∴ game tree as two opposing players in a perfect-information game

Adversarial Search:

search in which two or more players with conflicting goals are trying to explore the same search space for the solution

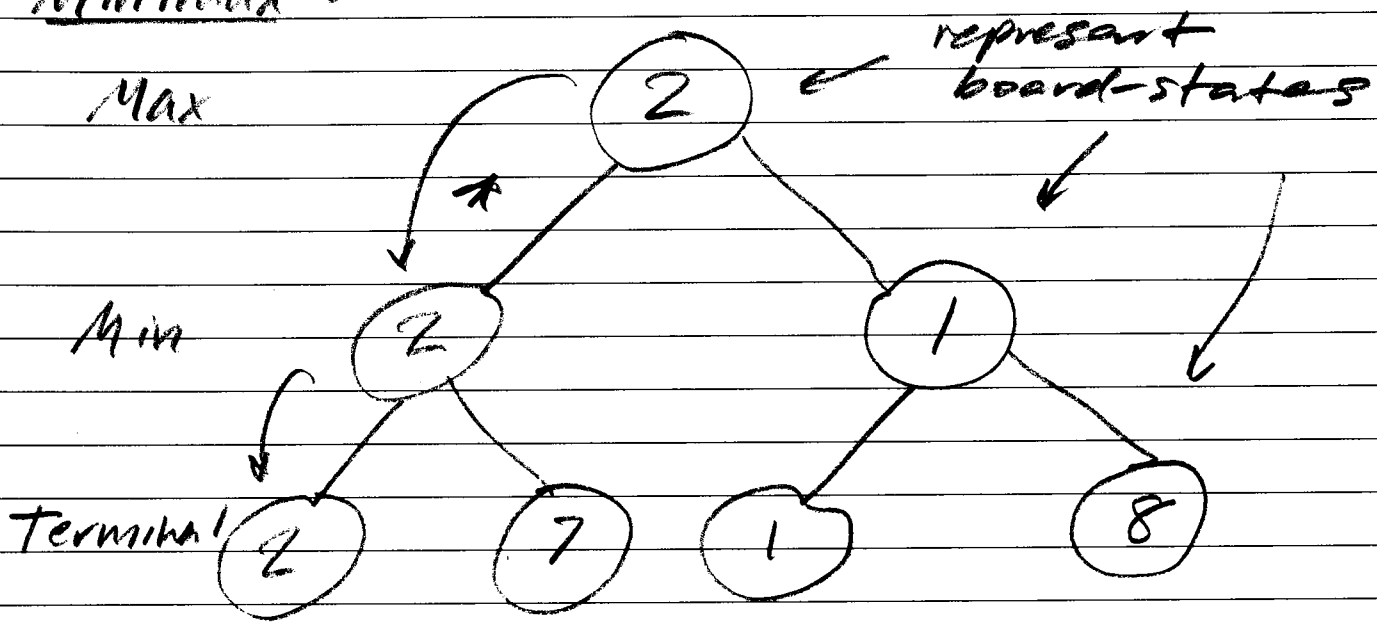
The Minimax Algorithm:

decision rule used in AI, decision theory, game theory, statistics, and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario

In two-player zero-sum games, this is same as the Nash equilibrium

Date.
No.

Minimax:



Where do the numbers come from?

$$f\left(\frac{\text{board}}{\text{game state}}\right) = N, 4$$

N = blocked X and 2 in a row

To create a score for a board state, we need a heuristic function to assess the board

4 Levels of these trees are called Plies

The case of ties:

just pick one for some reason it doesn't matter

note, in multiplayer minimax, each player just maximizes their score

Date.
No.

The Complexity of Chess:

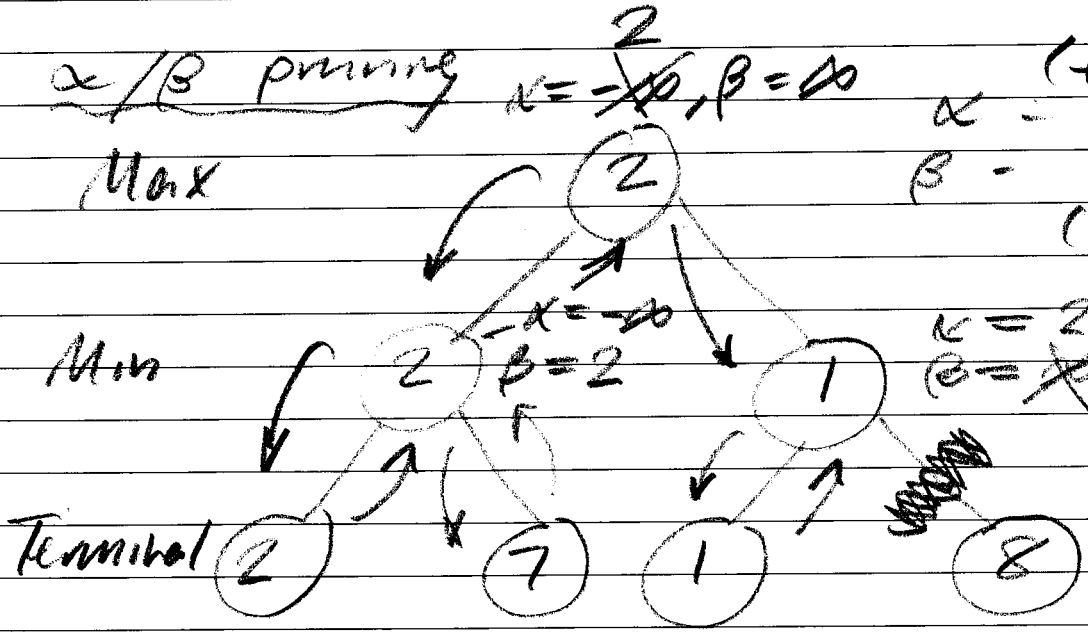
Chess Terminal States: 10^{120}
ie. really large

Alpha/Beta Pruning: improving minimax

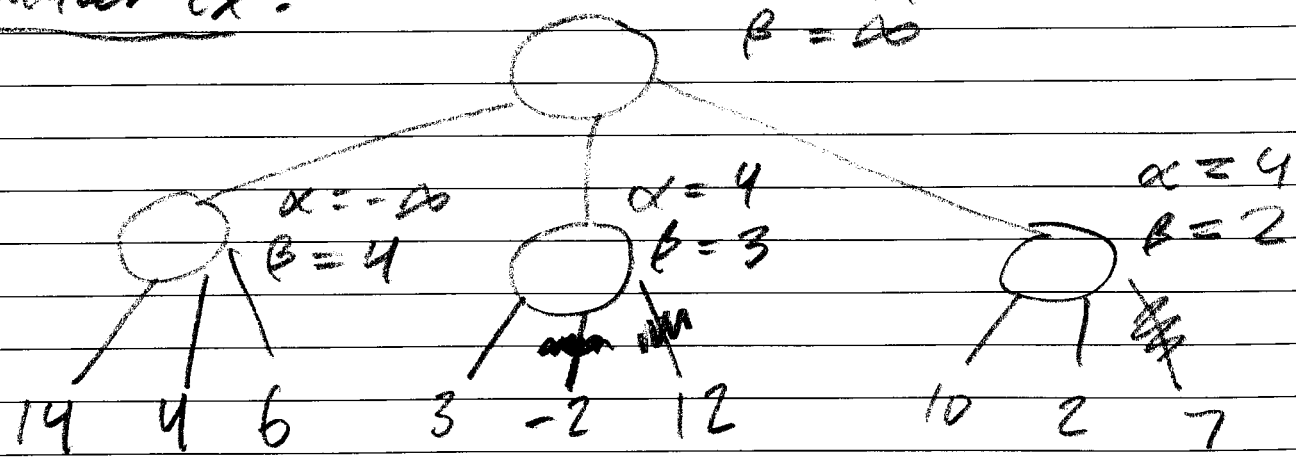
search algorithm ~~that~~ decreases the number of nodes that are evaluated in minimax

α/β pruning $\alpha = -\infty, \beta = \infty$

(floor)
α - "at least"
β - "at most"
(ceiling)



Another Ex:



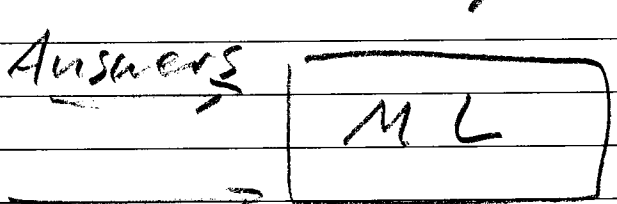
best score: 4
cut at branches: -2, 12, 7

Decision Trees

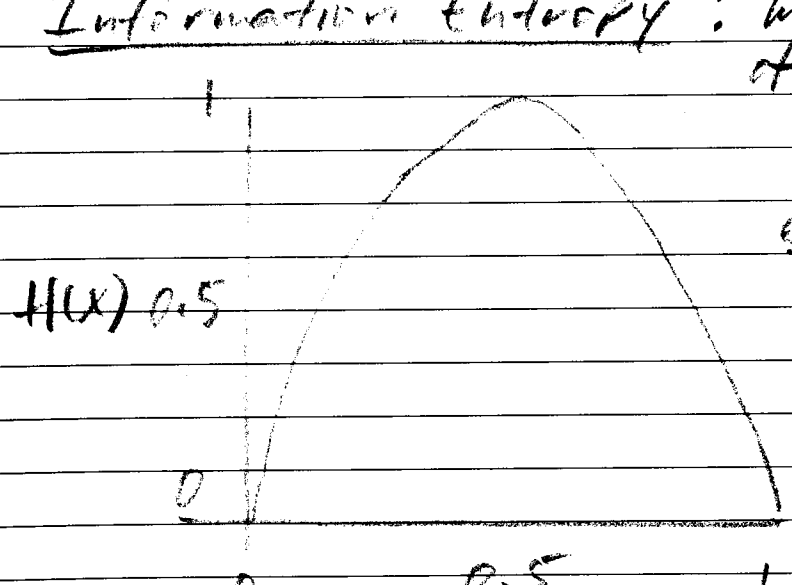
Date. 9.15.25
No. 67

Unit 02: Machine Learning (ML)

subset of AI concerned with the development of statistical algorithms that can learn from data and generalize to unseen data



Information Entropy: measuring the uncertainty of random variables



ex. flipping a coin

we can use this to measure the expected amount of information to convey the outcome

$$H(x) = -P(x) \log_2 P(x)$$

Date.
No.

Function Approximation:

$\exists f(x)$ that produces the data we see.
In ML, we try to train a model to approximate $f(x)$

Ex. Cubes: 11/20
Coins: 2/20
Ships: 2/20
Meeple: 1/20
Soldiers: 3/20
Pawn: 1/2

* want to split where the entropy reduction is the largest *

information gain = $\frac{d}{dx}$ information entropy

$$-(0.55\log_2 0.55 + 0.10\log_2 0.10 + 0.10\log_2 0.10 + 0.15\log_2 0.15 + 0.05\log_2 0.05 + 0.05\log_2 0.05)$$

$= 1.965$ bits

ie. in 2 bits, we can find out what's in the bag (ON AVERAGE)

cube → 0
soldier → 1
coin → 00

Date.
No.

$H(x) = 1.98$

IG

$IG(x, A) = H(x) - H(x|A)$

$A = is\_cube$

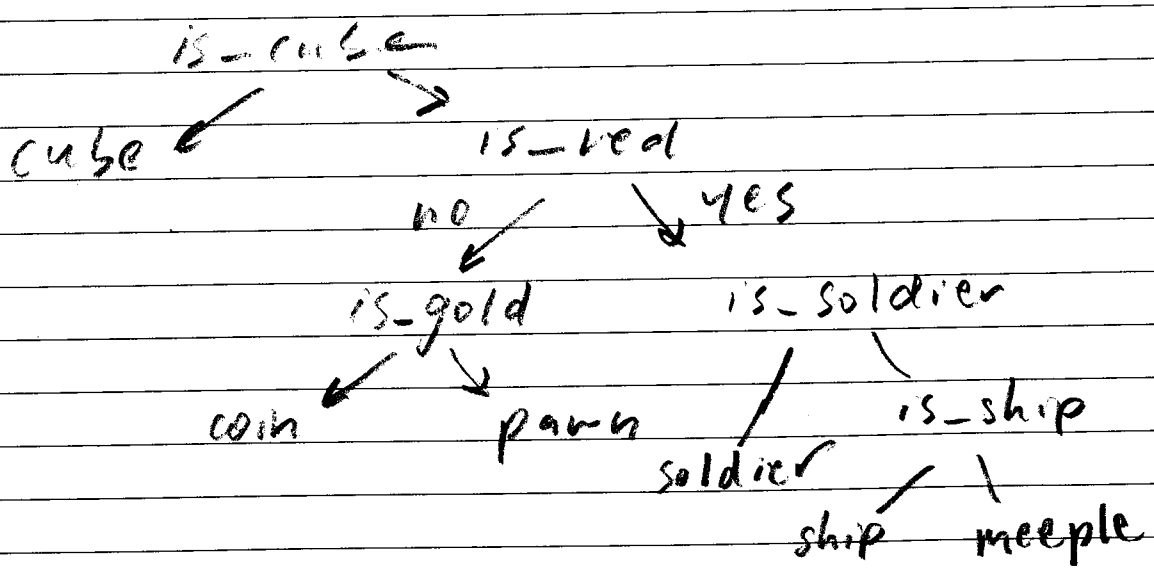
$H(x|A) = -\sum P(x) \log_2 P(x)$

$= -\left[\frac{2}{9} \log_2 \frac{2}{9} + \frac{2}{9} \log_2 \frac{2}{9} + \dots \right]$

$= 0.99$

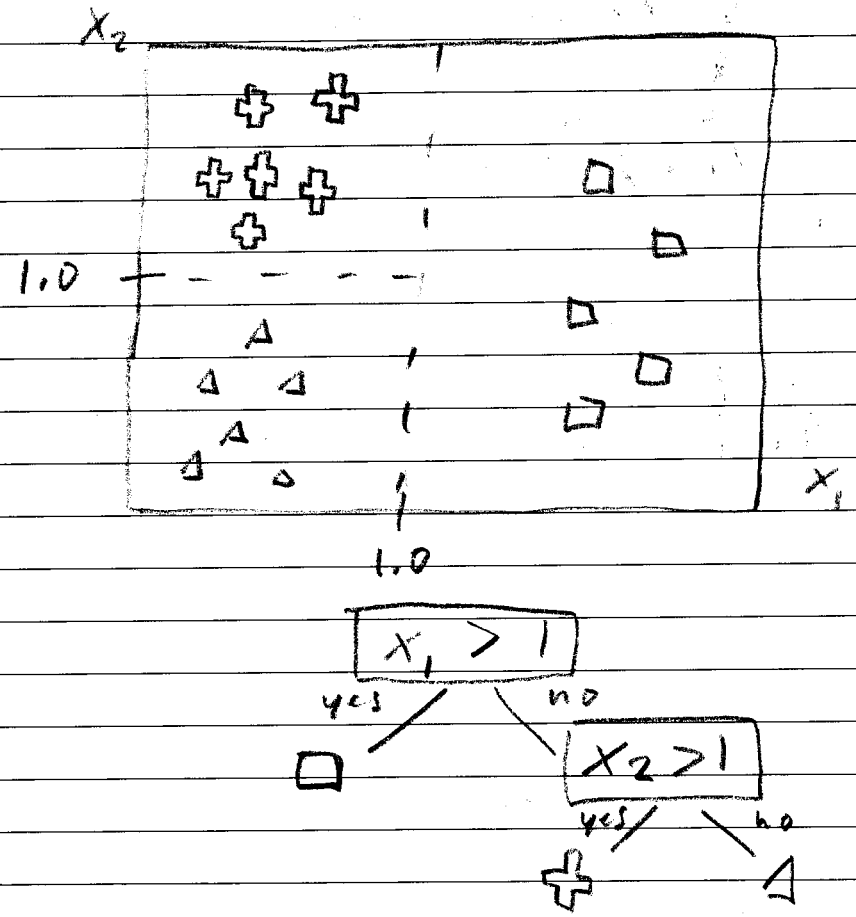
$\Rightarrow IG = 1.98 - 0.98 = 0.998$

... some calculation for all questions ...
... and pick the best one ...
(the one that reduces the entropy the most)



Date.
No.

Decision Trees:



Splitting Data:

we search among all the possible ways to split the data and choose, in a greedy way, the split that reduces entropy the most:

1. Information Gain
2. Gini Impurity
3. Variance

Search will continue to show up throughout the course, though the spaces we search in change

Date.
No.

Ground Truth: Supervised learning and a classification task

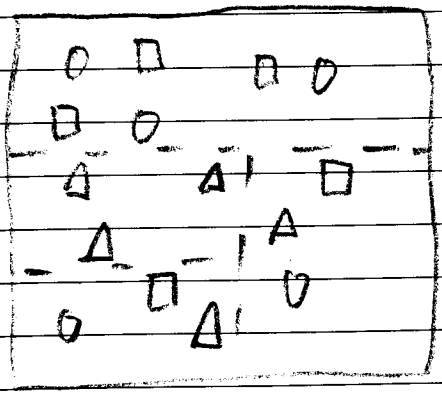
give model labeled data and answers to use while making decision tree

Sklearn: the python package for machine learning

# fitting the model
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion = 'entropy')
clf = clf.fit(x, y)

# the predictions are stored in x_pred
x_pred = clf.predict(x)

Leaf Classification Strategies:



Strategies:

1. Vote
2. Percentage
3. Deepen Tree

Under and Overfitting Data:

underfitting: model is not complicated enough

overfitting: model is too fit to training data

Date. 9.10.25
No. 06

Knowledge Representation

Ground-Truth Data: information that is known to be true
- needed in machine learning to train and test

Supervised Learning: when we have ground-truth "something to compare to", we call the learning process supervised

Feature Engineering: preprocessing step in supervised ML / statistics that transforms raw data into a more effective sets of inputs

Categorical Variables: represent types of data which can be divided into groups

ex. education levels, age bracket, humidity

Input Space vs. Feature Space:

Day	Outlook	Temp	Humid	Wind	Play

Sample #1:

< 1, Sunny, Hot, High, Weak, [No] >

ground truth

↓

out_sun	out_over	out_rain	...	Play
1	0	0	...	1

Date.
No.

Continuous Data Spaces:

continuous data can take on any value within a defined range and is often measured on a continuous scale

ex. weight, height, temperature

categorical data consists of discrete values that fall into distinct categories

↑ features as Axes, samples as Points ↑

Machine Learning Tasks:

Regression: given input → predict value for a sample

Classification: given input → predict label for a sample

other tasks: retrieval, clustering

Next 3 Lectures:

- Decision Trees
- Markov Models
- Bayesian Statistics

↑ All 3 can be constructed as state diagrams

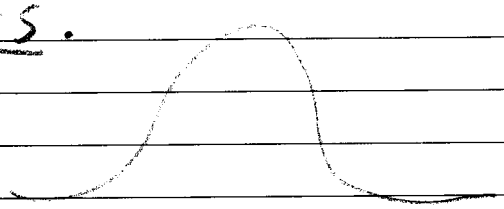
Date. 9.17.25
No. 68

Markov Models

Statistical Distribution: describes the likelihood of different outcomes

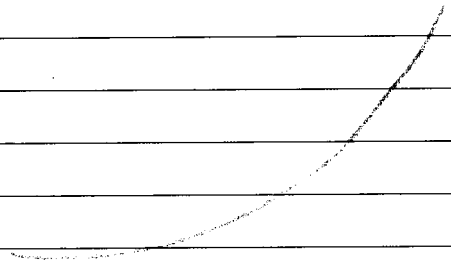
note. AI is trying to learn distributions

Exs.



"normal"

"uniform"

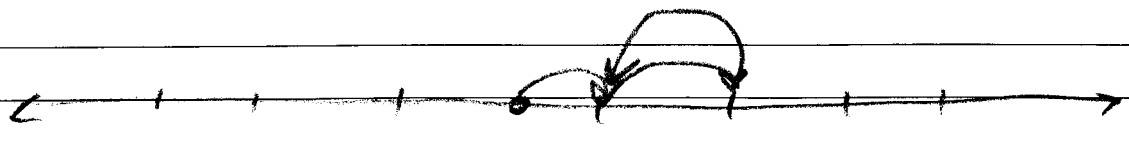


"exponential"

Introduction to Markov Processes:

a systematic method for generating a sequence of random variables where selecting the next variable is dependent only on the value of the previous variable

heads = +1, tails = -1



Date.
No.

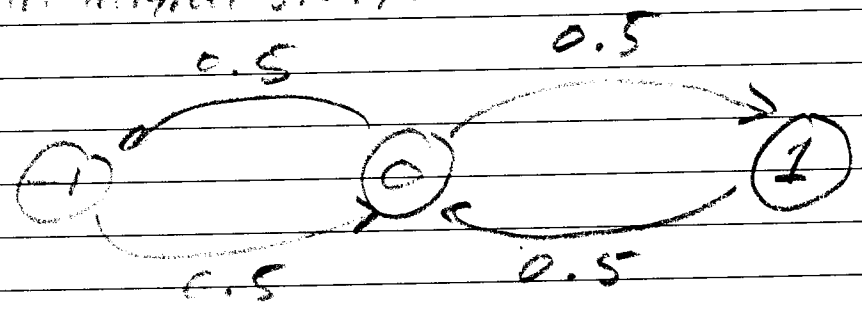
Markovian Terms:

the changes of state are called transitions and the probabilities associated are called transition probabilities

the process is characterized by:

1. a state space
2. a transition matrix
3. an initial state

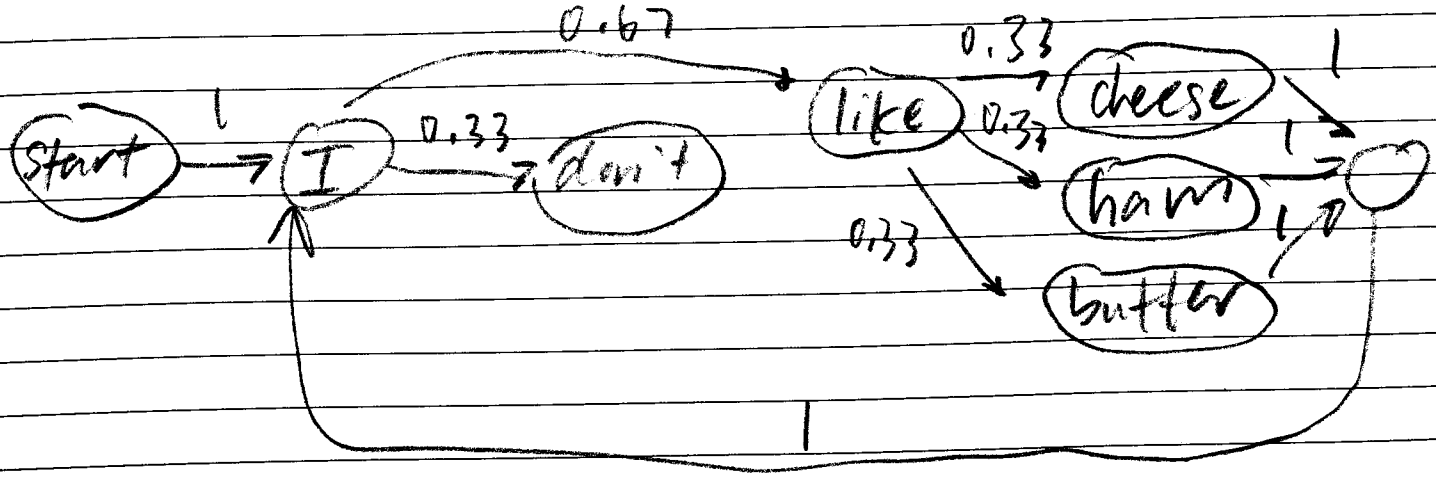
Ex.



Markov Language Model:

How could you use a Markov chain to model and generate english sentences

Markov Babblers: from a training corpus, we could build a transition matrix for the english language and then sample from the learned distribution



Date.
No.

note, N-Grams:

to improve a Markov Babbler we could consider more words at each state

word level character level

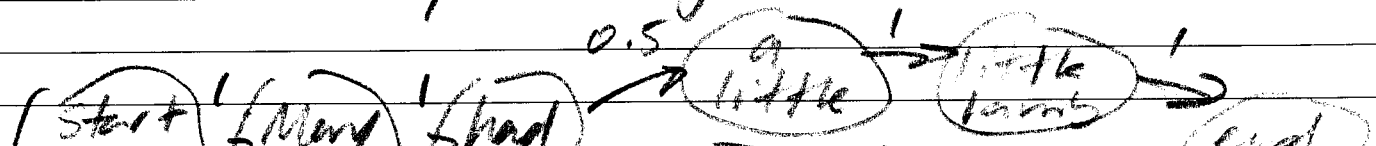
I hate | professor Bill

bigram bigram

also called a "tokenizer"

Bigram State Space: draw the bigram state space for the following sentences

Ex. • Mary had a little lamb
• Mary had a giant crab



has "context" of prior word

Hidden Markov Models:

Markov Model: has directly observable states and associated transition probabilities

Hidden MM: involves a sequence of hidden (unobservable) states that generate observable outcomes

Date.
No.

The Viterbi Algorithm: a dynamic programming algorithm used to find the "Viterbi Path", or the path returning the maximum a posteriori probability estimate

Pseudocode:

```
def viterbi(states, init, trans, emit, obs):
    prob = T x S matrix of 0s
    prev = T x S empty matrix

    for state in states:
        prob[0][state] = init[state] * emit[state][obs[0]]

    for t in range(1, len(obs)-1):
        for curr in states
            for prev in states
                new_prob = prob[t-1][prev] *
                           trans[prev][curr] *
                           emit[curr][obs[t]]

                if new_prob > prob[t][curr]
                    prob[t][curr] = new_prob
                    prev[t][curr] = prev

def reconstruct(prob, prev):
    path = []
    path[t-1] = max prob final state (prob[t-1][s])

    for t in range(t-2, 0):
        path[t] = prev[t+1][path[t+1]]

    return path
```

HMM in sklearn: HMM are not in sklearn ☹ must do it from scratch

Date. 9.22.25
No. 09

Bayesian Reasoning

Random Variables: has a domain of values

Ex. Dice: {1, 2, 3, 4, 5, 6}
Coin: {H, T}

when sampled, random variables will select a value from its domain

Unconditional Probability:

P(A): probability of event A

Ex. $P(5) = 1/6$

Independent Event: knowledge of one event doesn't influence the probability of a second event

$$P(A \cap B) = P(A)P(B)$$

Ex. $P(6)P(5) = 1/6 \cdot 1/6 = 1/36$

Conditional Probability:

P(A|B) = probability of A "given" B

B = "evidence" or "observation"

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Date.
No.

Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayesian Network: probabilistic graphical model that represents a set of variables and their conditional dependencies via a DAG

Bayesian Inference: if we wake up and observe wet grass, what is the probability that it rained?

$$\text{Ex. } P(\text{Rain} \mid \text{Wet Grass}) = \frac{P(\text{Wet Grass} \mid \text{Rain})P(\text{Rain})}{P(\text{Wet Grass})}$$

$P(\text{Rain}) = 0.2$ # network in slides #

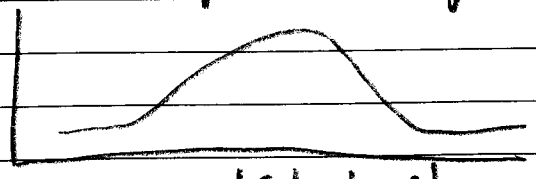
$$P(\text{Rain} \mid \text{Wet Grass}) = \frac{\sum_{x \in \{T,F\}} P(G=T, S=x, R=T)}{\sum_{x,y \in \{T,F\}} P(G=T, S=x, R=y)}$$

$$= \ldots = 0.3577$$

Ex. can describe every flower with 4 values or features, every flower has a **feature vector** of size 4 or be a point in 4D space

table of values in slides

What will petal length look like

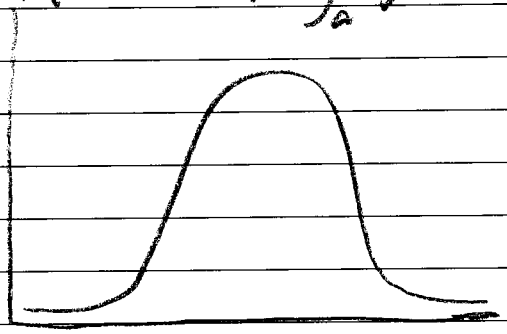


Date.
No.

Probability Density Functions:

$$P(a \leq x \leq b) = \int_a^b f(x)\,dx$$

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

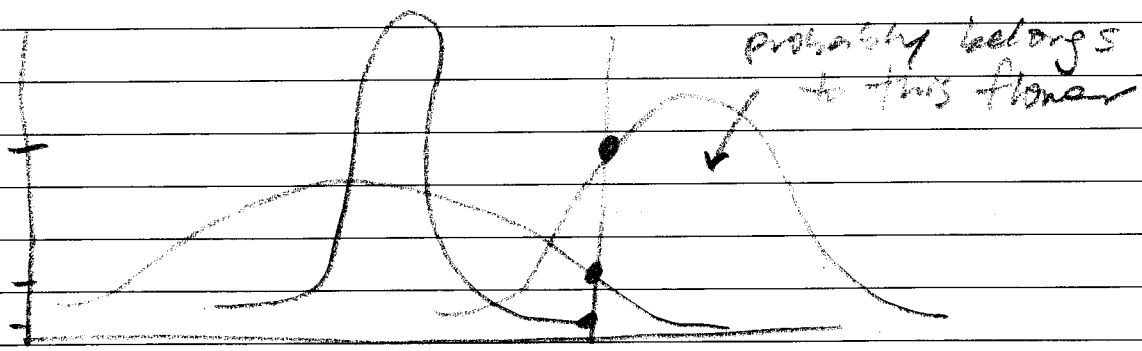


μ = mean

σ = std. dev

provides a relative likelihood that a variable would come from a model

Ex. if I had an unknown iris with a pedal length, how could I make a prediction of which species it is

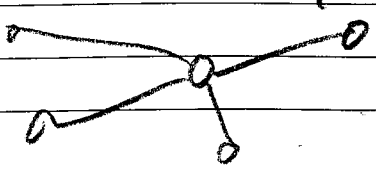


pedal length

Naive Bayes Network: Bayesian Networks with an assumption of conditional independence amongst features

* this assumption is what makes them naive

* also means DAGs only have straight lines out! *



Date.
No.

Maximum Likelihood Estimation (MLE):

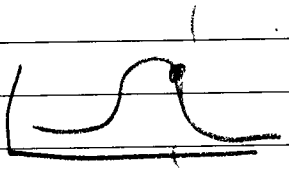
method of estimating the parameters of an assumed probability distribution given some observed data

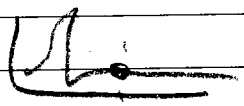
Ex. classifying a flower

$S_i = \{x_1 : 5.1, x_2 : 3.0, x_3 : 4.2, x_4 : 1.5\}$

x_1: feature

for each feature

Setosa: $\mu = 5.01$, $\sigma = 0.35$ → 

Versicolor: $\mu = 5.94$, $\sigma = 0.51$ → 

Virginica: $\mu = 6.59$, $\sigma = 0.63$ → 

*look at the curves for each feature and multiply them all together *

Date. 9.24.25
No. 10

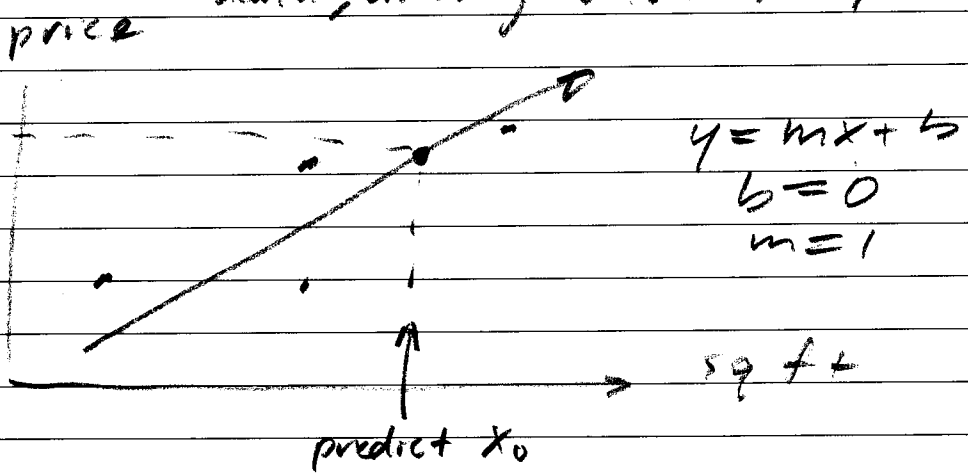
Linear Regression

Classification Task: one in which we try to assign a discrete label to a sample

Regression Task: we try to assign a continuous number to each sample

A regression is another form of supervised learning

Linear Regression: technique to fit a line to our data, allowing us to make predictions



Regression-specific Vocab:

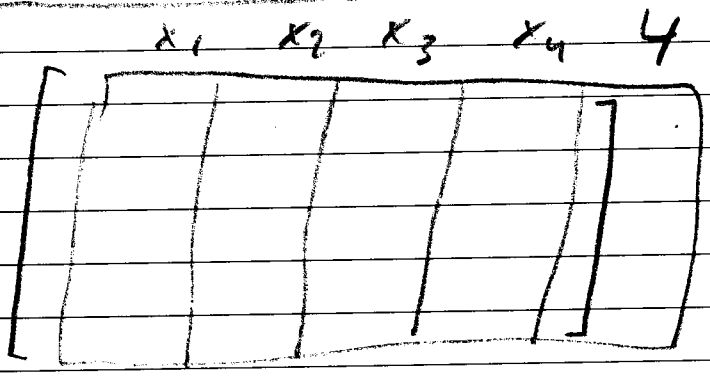
$$y = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \beta_0$$

β: regression coefficient (or w for weight)
x_i: independent, explanatory, covariate variable (i.e. feature)
y: dependent, target variable (prediction)

Date.
No.

Points in Space: features are dimensions, and samples are points in [len_feature] dimensional space

Matrix Notation:



features can be represented as matrices, and can do matrix operations to predict values

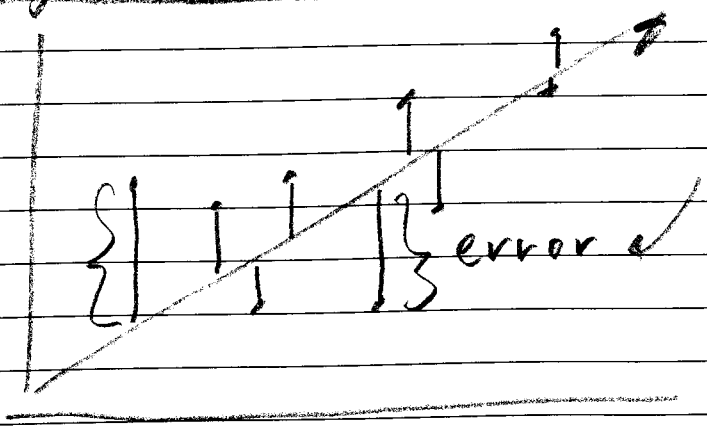
$(n \times 4) \cdot (4 \times 1) = (n \times 1)$ or y^T

Multiple Linear Regression: linear regression performed on many features

$$y' = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_0$$

Hyperplanes: "lines" in higher dimensions are really hyperplanes

Fitting Our Line: how do we find the line that "fits"



Date.
No.

Optimization Problems:

Cost (Loss) function: way to measure how well our model fits the data

* for Lin Reg, we often use SSE, or sum of squared errors

$$SSE = \sum_{i=1}^{n} (y - x_i^T w)^2$$

ground truth ↑ ↑ prediction

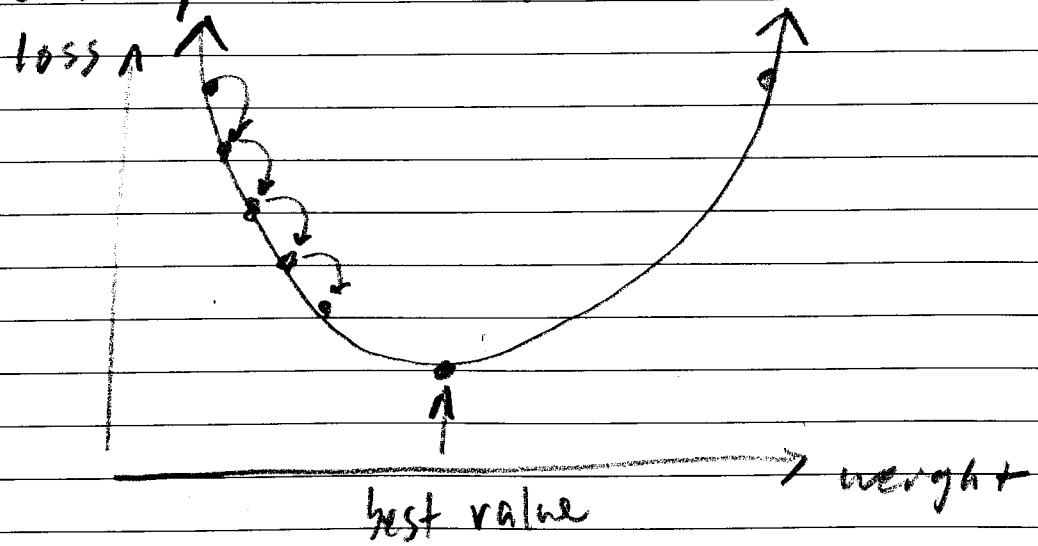
Finding our weights:

given a loss function, our goal is to minimize it

when would it be minimized? when the gradient is 0

Visualizing Loss:

we can plot the loss function with respect to w



note. perfectly convex → singular optimal point

Date.
No.

We can solve for the gradient equaling 0 using Ordinary Least Squares (OLS):

$$w = (X^T X)^{-1} X^T y$$

note. taking the inverse of a matrix is expensive, so this is not often used in practice

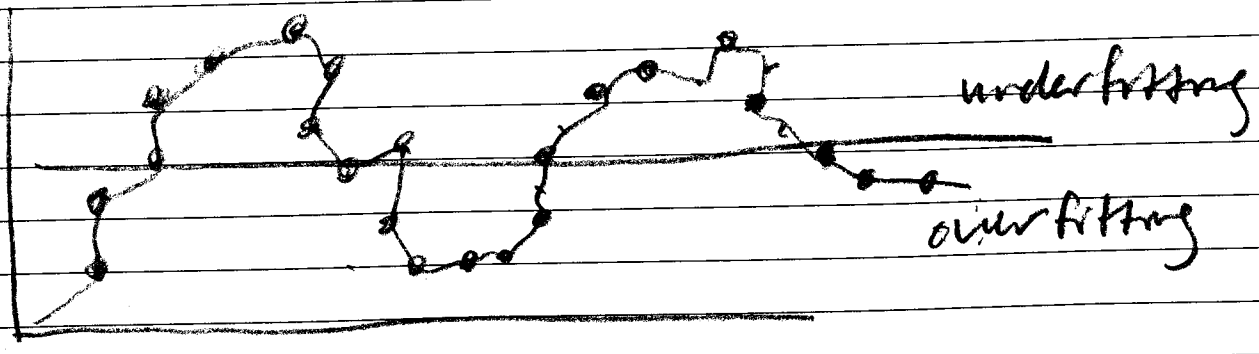
instead, Gradient Descent is used!

Maximum Likelihood Estimation (MLE):

method of estimating the parameters of an assumed probability distribution given some observed data

"choose parameters that make observed data most likely"

Overfitting: Linear Regression (a form of MLE) is prone to overfitting



Date.
No.

Regularization : L_1 and L_2

modifications to the loss function to encourage models to learn the broader patterns within data without memorizing it

L_2 Regularization: Ridge regularization

encourages smaller, evenly distributed weights

$$f(w) + \lambda \|w\|_2^2$$

L_1 Regularization: Lasso Regularization

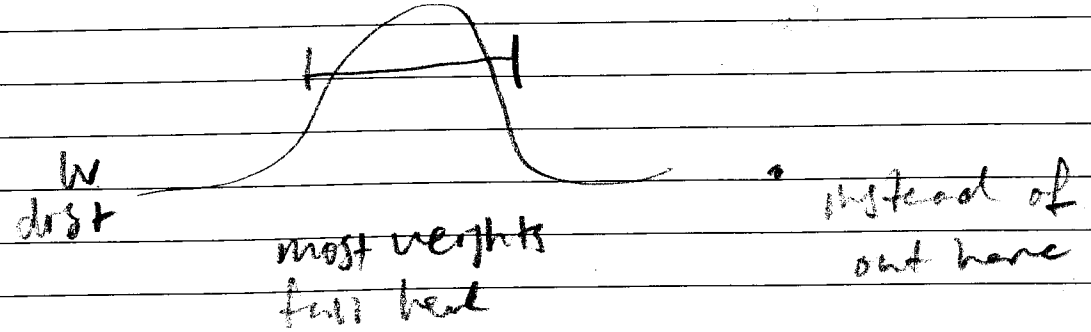
leads to weights being set to 0 and sparse models ("feature selection")

$$f(w) + \lambda \|w\|_1$$

MAP Estimation: L_1 and L_2 regularization are forms of this

MAP: Maximum a posteriori

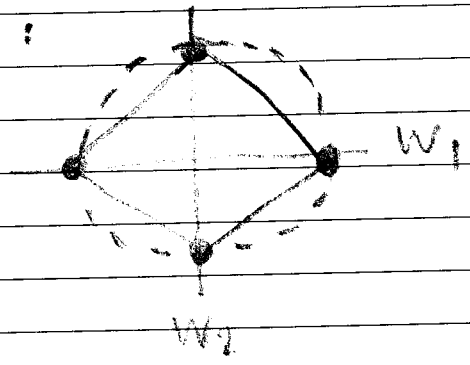
i.e. Bayesian approach for apply a prior to our weights



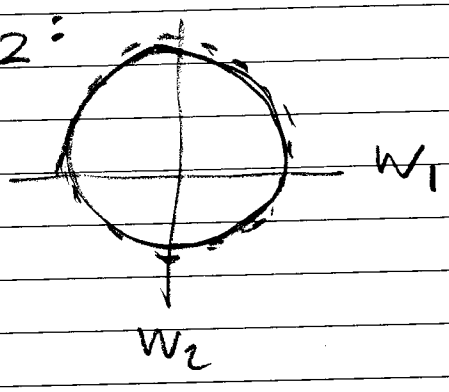
Date.
No.

L_1 and L_2 MAP Estimation:

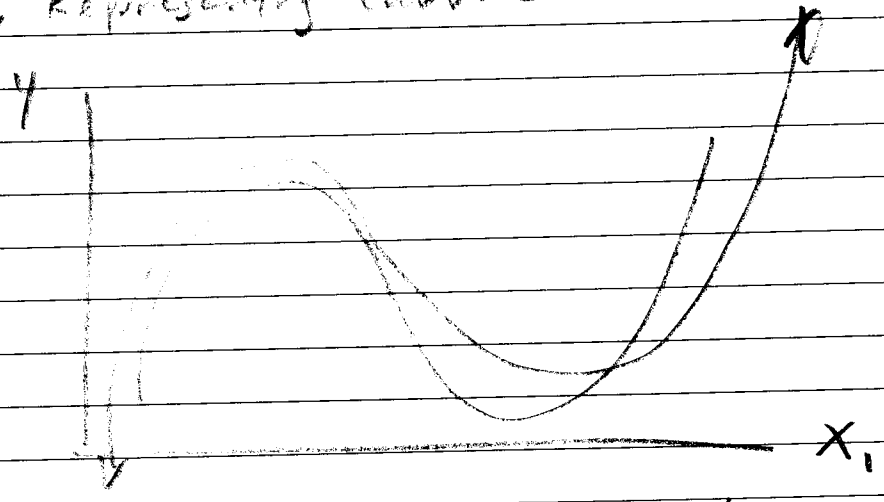
L_1:



L_2:



Ex. Representing Curves



linear regression is BAD here!

use polynomial regression:

$$y = w_2 x_1^2 + w_1 x_1 + b$$

$$y = w_3 x_1^3 + w_2 x_1^2 + w_1 x_1 + b$$

...

$$y = w_n x_1^n + \dots + w_1 x_1 + b$$

Date. 9.29.25
No. 11

Perceptrons

Linear Transformations: can predict the output of a new point with the dot product

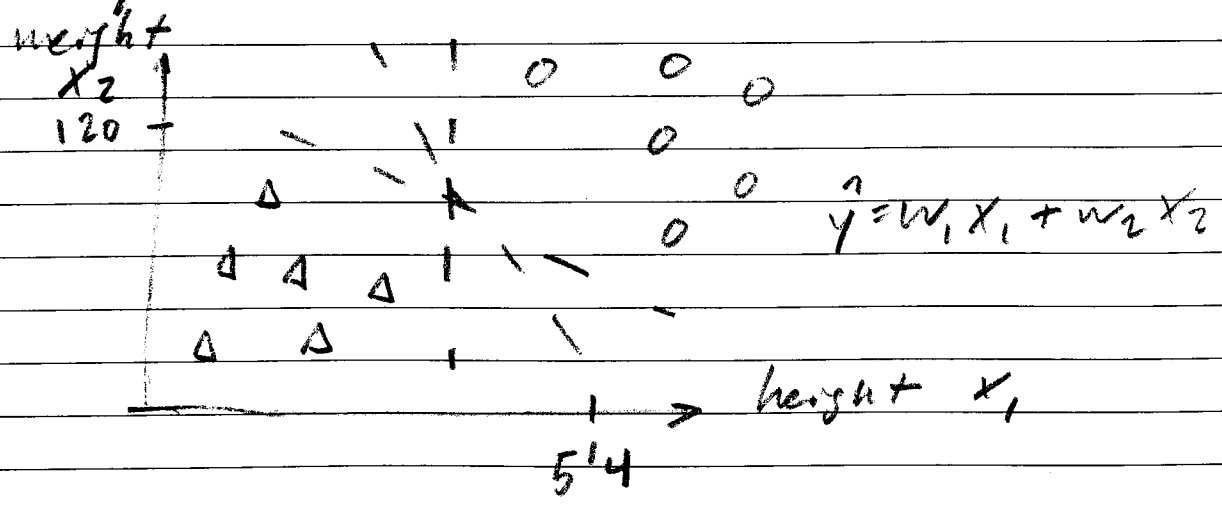
$$\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} \cdot \begin{bmatrix} x_0 & x_1 & \dots & x_n \end{bmatrix} = \hat{y}$$

weights features value

can also predict the output of multiple points with matrix multiplication

$$\begin{bmatrix} x_{01} & x_{11} \\ x_{02} & x_{12} \\ x_{03} & x_{13} \end{bmatrix} \times \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix}$$

Classification via Regression: how could you modify a linear regression to perform a binary classification



can set a threshold:

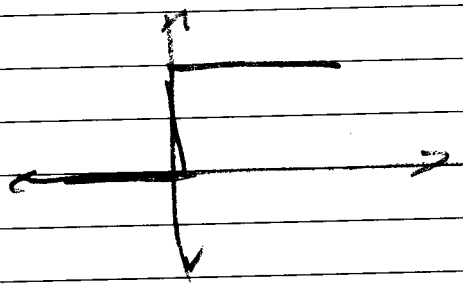
$$f(x) = \begin{cases} x > 0 = 1 \\ x < 0 = 0 \end{cases}$$

Date.
No.

Binary Classification:

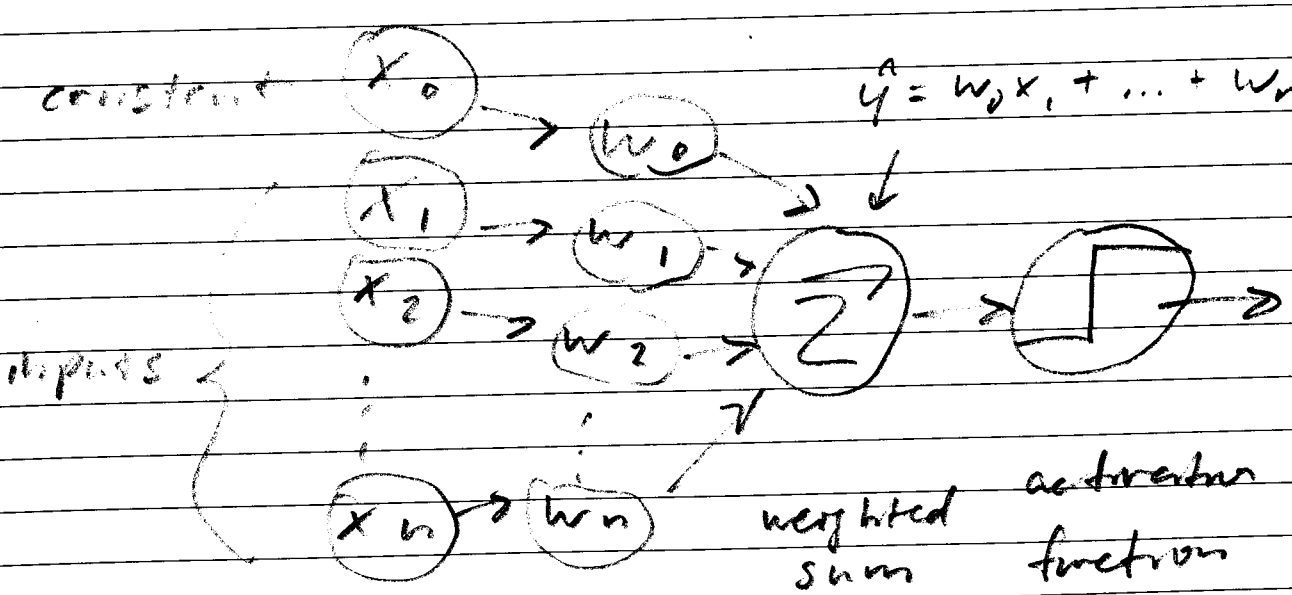
we can add an activation function that will transform the output from our linear transformation into one of two classes

$$f(x) = \begin{cases} x \geq 0 = 1 \\ x < 0 = 0 \end{cases}$$



$$\hat{y} = w_0 x_0 + w_1 x_1 + w_2 x_2$$

The Perceptron:



features weights

OLS for Perceptron:

for lin reg we set the gradient of the loss function with respect to the weights to 0

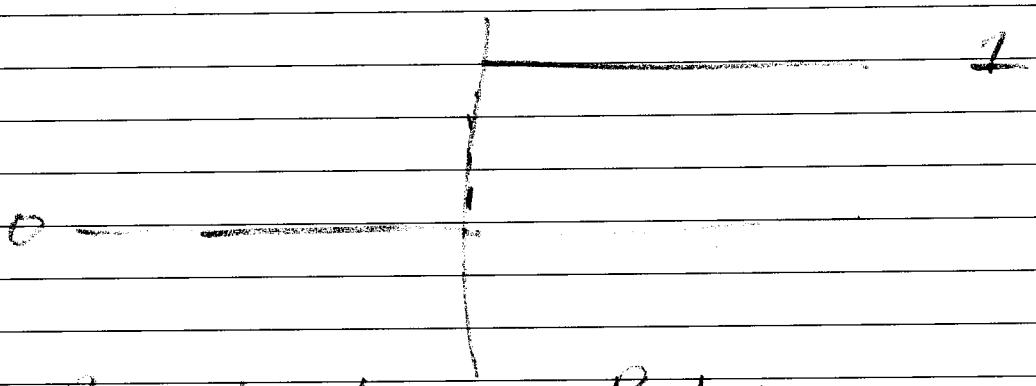
$$OLS = (x^T x)^{-1} x^T y$$

we CANNOT do this for perceptrons

Date.
No.

Implications of the Step Function:

due to the non-linearity of the step function we can't calculate its gradient



The Perceptron Learning Rule:

there is unfortunately a lot of these:

$w = w + \alpha y_i x_i$
$w = w + \alpha(y_i - \hat{y}_i) x_i$ } UIUC

$w = w + r(d_j - y_j) x_i$ } wikipedia

$w = w + \eta(y_i - \hat{y}_i) x_i$ } geeks 4 geeks

$w = w + \mu y_i x_i$ } ND

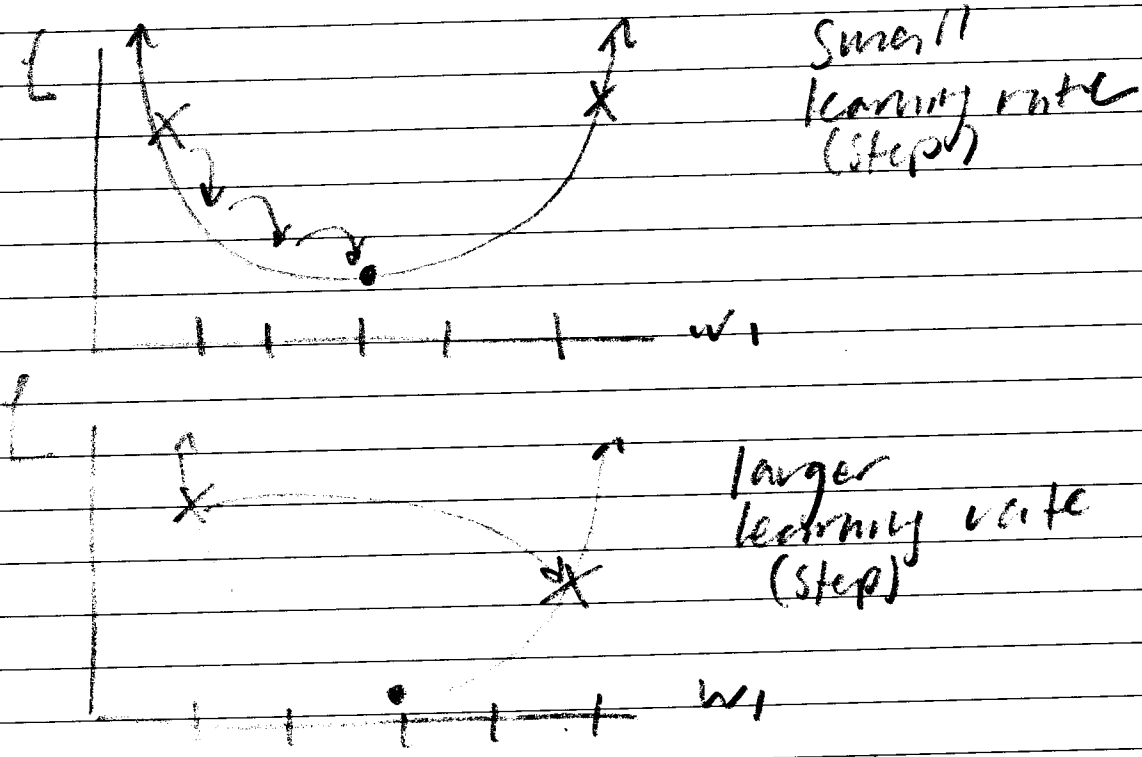
The Learning Rate: the size of the "step" we take when updating our weights

Large Learning Rate: quicker updates
may overshoot

Small Learning Rate: slower updates
probably won't overshoot

Date.
No.

Learning on Loss:



note. number of steps is called "epochs"

E.x. $w_1 = (1.2, 0.7, 0.2, 0.5)$ $\eta = 0.2$
(perceptron (weights) LR

Samples	x_1	x_2	x_3	y
S_1	2	3	-1	1
S_2	1	1	1	1
S_3	-2	-4	3	-1

activation fuction = $\{-1, 1\}$

Classify:

S_1: $1.2(1) + 0.7(2) + 0.2(3) + 0.5(-1) = 2.7 \rightarrow 1 = 1$ ✓
S_2: $1.2(1) + 0.7(1) + 0.2(1) + 0.5(1) = 2.6 \rightarrow 1 = 1$ ✓
S_3: $1.2(1) + 0.7(-2) + 0.2(-4) + 0.5(3) = 0.5 \rightarrow 1 \neq -1$ X

Weight Updates:

$W_2 = 0.2 + (0.2)(-4)(-1) = 1.0$
... $w_2 + LR \cdot x_2 \cdot 4$ (ground truth)

Date. 10.1.25
No. 12

Review 01

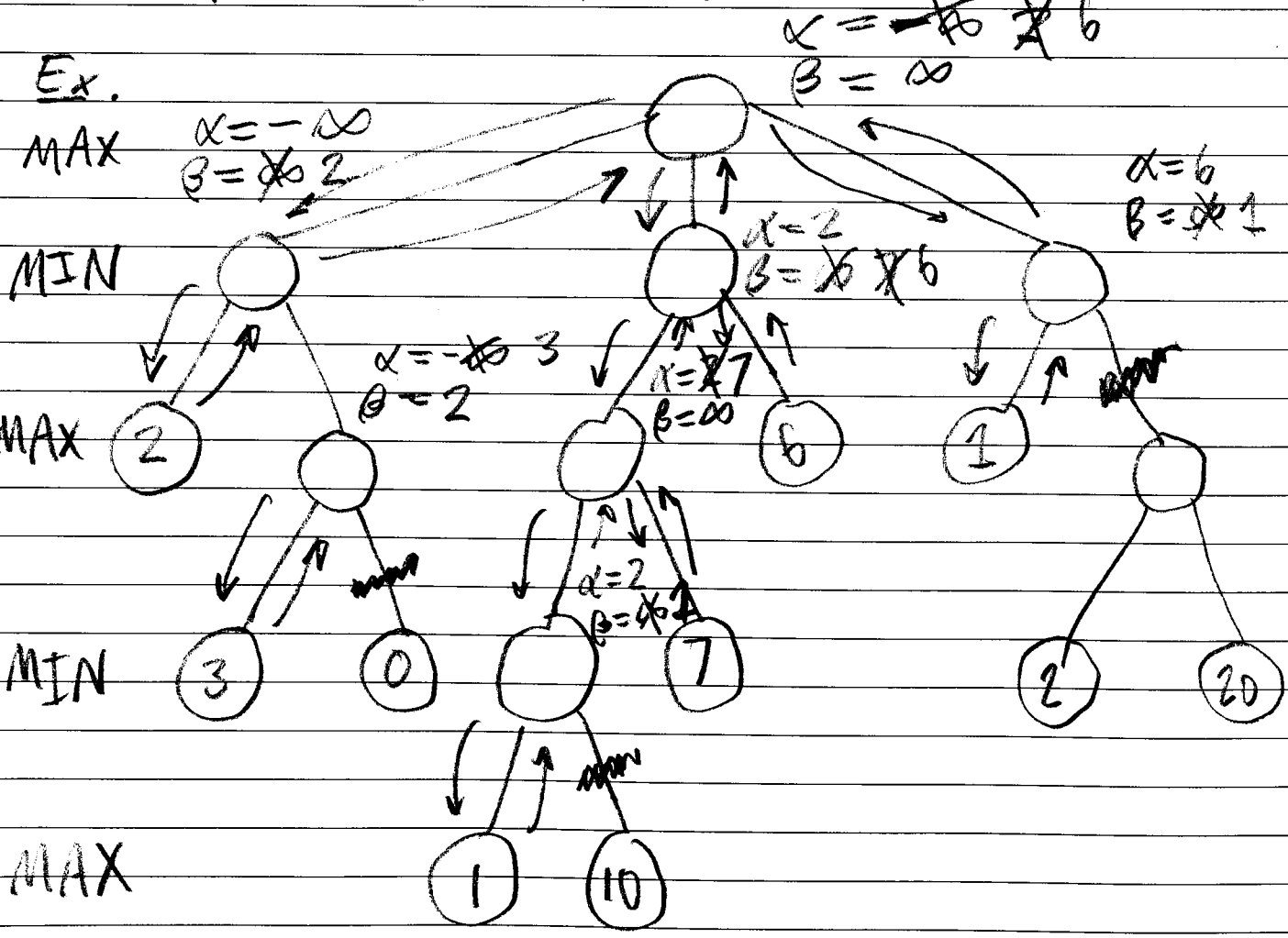
Alpha / Beta Pruning:

Alpha: "At Least" / "Floor"
- managed by MAX nodes
- starts at $-\infty$

Beta: "At Most" / "Ceiling"
- managed by MIN nodes
- starts at ∞

* Still run on top of minimax
* Alpha > Beta = Prune

Ex.



* only want α to go up
* only want β to go down

Date.
No.

Probability:

Random Variable: something we can sample from that will produce a result

ex. coin, dice, weather

Probability of Event: probability of one event when sampling a random var

ex. $P(H) = \frac{1}{2}$

Joint Probability: prob of 2 events from RV

ex. $P(A \cap B)$

Conditional Prob: prob of 1 event given another already happened

ex. $P(A|B)$

Bayesian Reasoning:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Naive Bayes:

- used for classification
- assumes independence of features (naive)
- assume distribution of features (gaussian) (MLE)

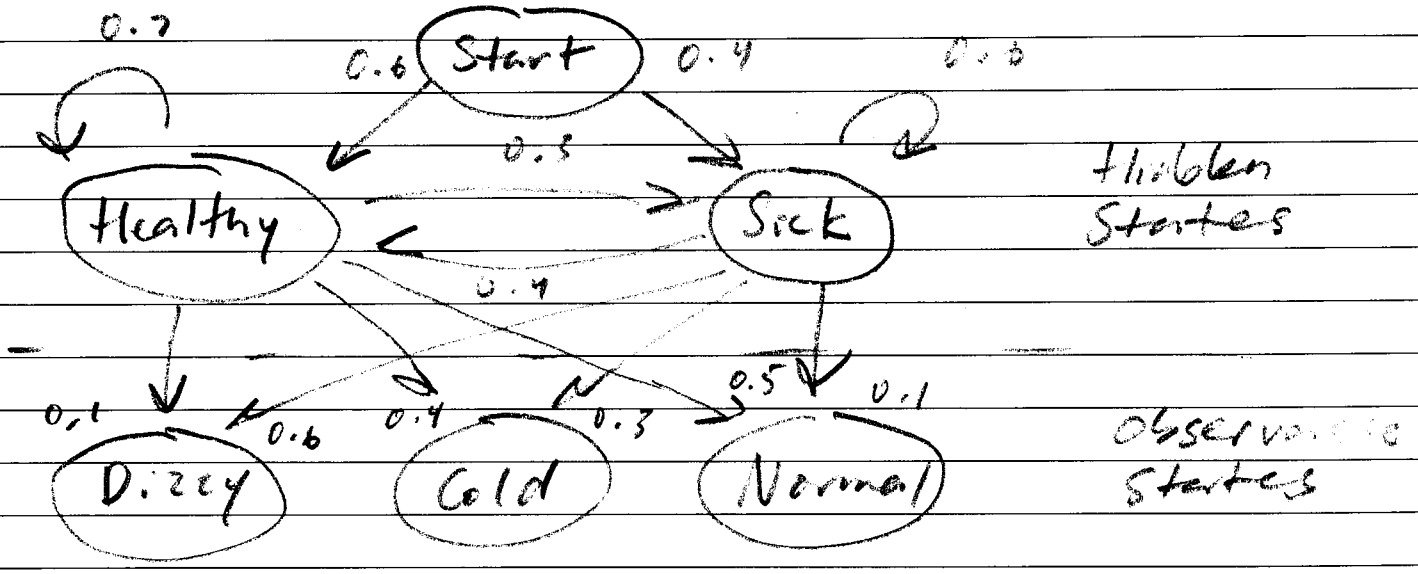
* pick distribution that fits data points
↑ calculate μ, σ that makes those data points most likely

Date.
No.

Markov Models:

- state machine with probabilistic transitions between states
- each state is only dependent on previous states

Hidden Markov Models:



Patient reports: Normal → cold → Dizzy

Day	1	2	3
	Normal	Cold	Dizzy
Healthy	A 0.3 →	C 0.084	
Sick	B 0.04 ↗		

pick greater of 2 since viterbi alg

A: P(Normal, Healthy) = 0.6 × 0.5 = 0.3
B = P(Normal, Sick) = 0.4 × 0.1 = 0.04

* 2 possible "paths" to reach Healthy and Cold on Day 2

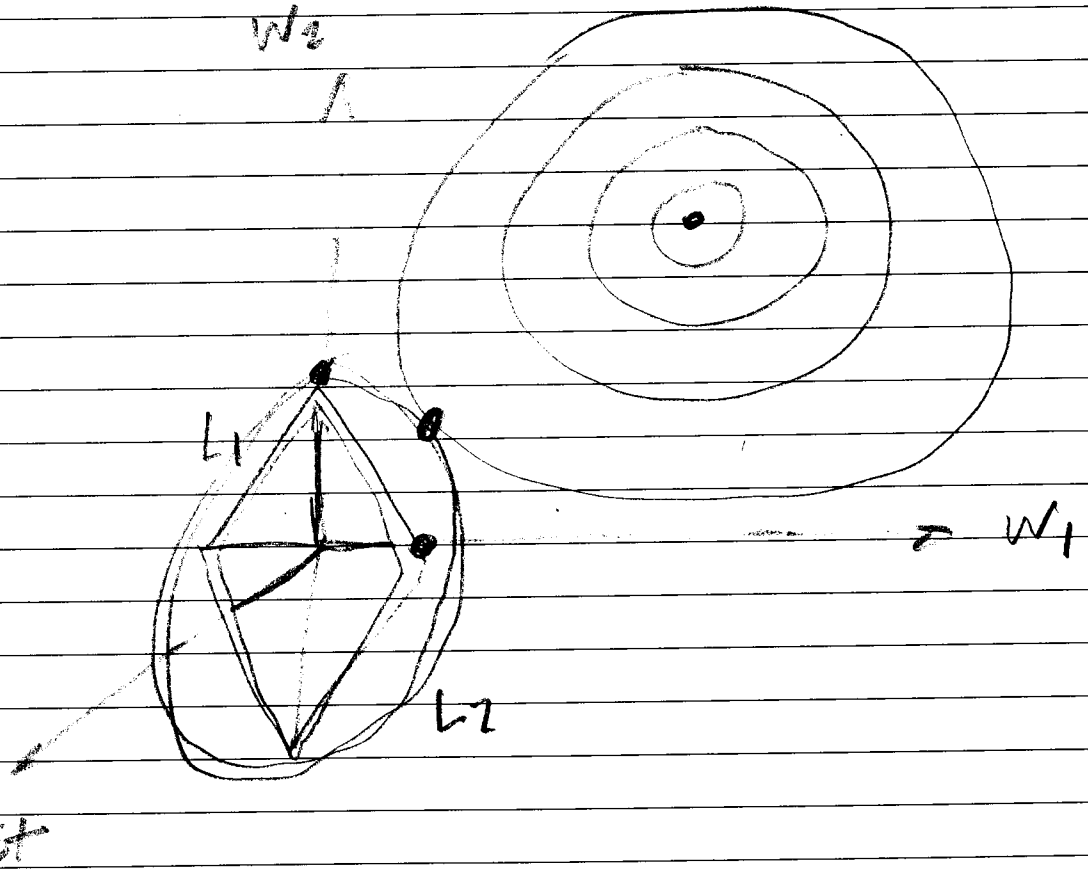
P[Healthy, Cold, A] = 0.3 × 0.7 × 0.4 = 0.084
P[Healthy, Cold, B] = 0.04 × 0.4 × 0.4 = 0.0064

* keep track of the most likely path—NOT total probability

Date.
No.

Regularization

- way to help overfitting
- L_1 - LASSO - sets most weights to 0
- L_2 - RIDGE - evenly distributed weights



- intersection between shapes is where weights must lie

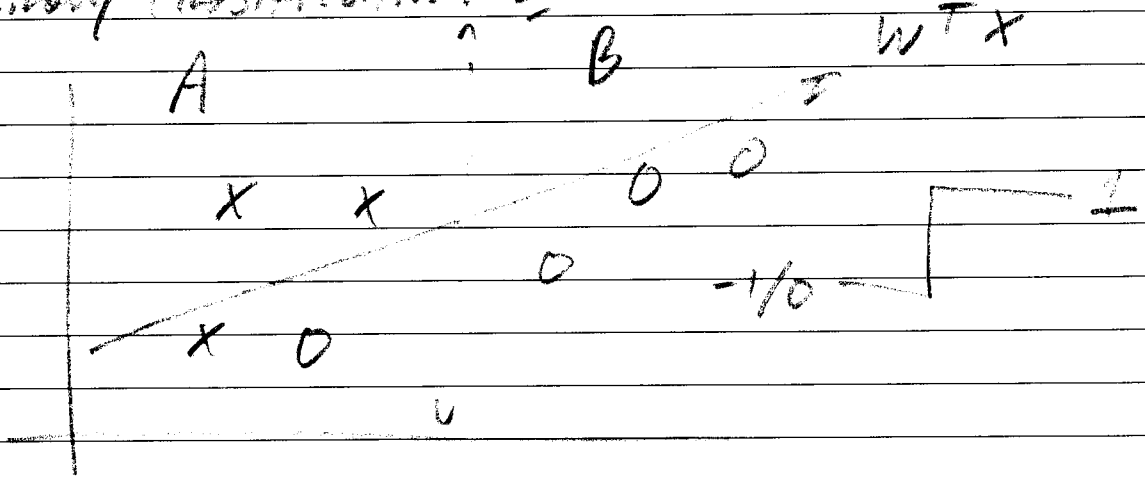
L_1: diamond
L_2: sphere

Date. 10.6.25
No. 14

Logistic Regression
(and Gradient Descent)

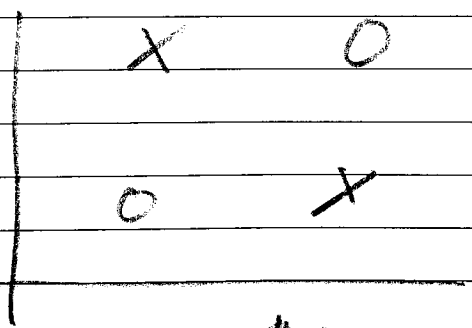
* Perceptrons need to have a perfect boundary between its binary decisions or it will train forever looking for it *

Classification vs Regression: how can you modify a linear regression to perform a binary classification?



Binary Classification: we can add an activation function that will transform our regression to one of two values

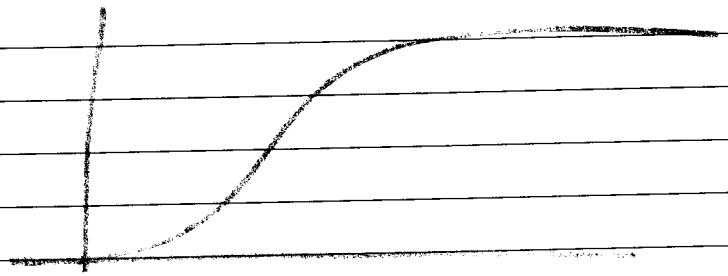
The XOR Problem: how could you classify this data with a perceptron?



You CAN'T (*), need a multi-layer perceptron

Date.
No.

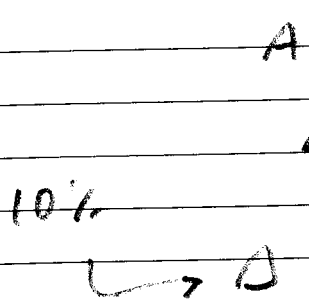
Fixing the Step Function:



use sigmoid instead of step function

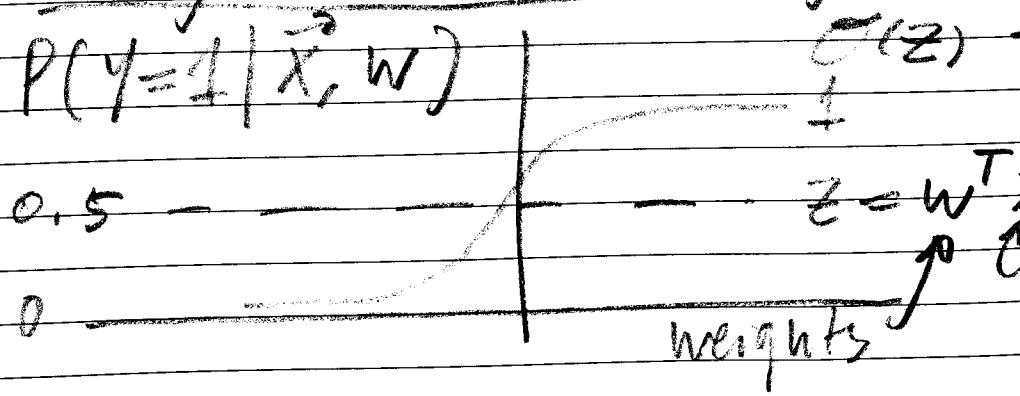
Issue with Perceptrons:
- step functions are non-differentiable (no gradient descent)
- no indication of how extreme a point is in a class (just 0 or 1)

Logistic Regression: binary classifier similar to Perceptron, but gives us the probability of a sample belonging to a class



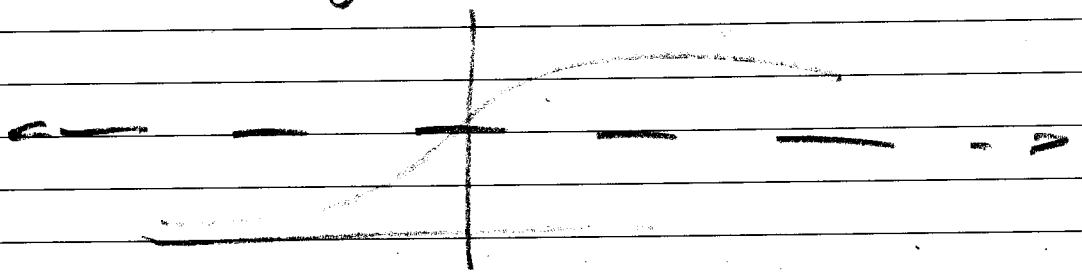
The Sigmoid Function: Sigmoid = σ

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



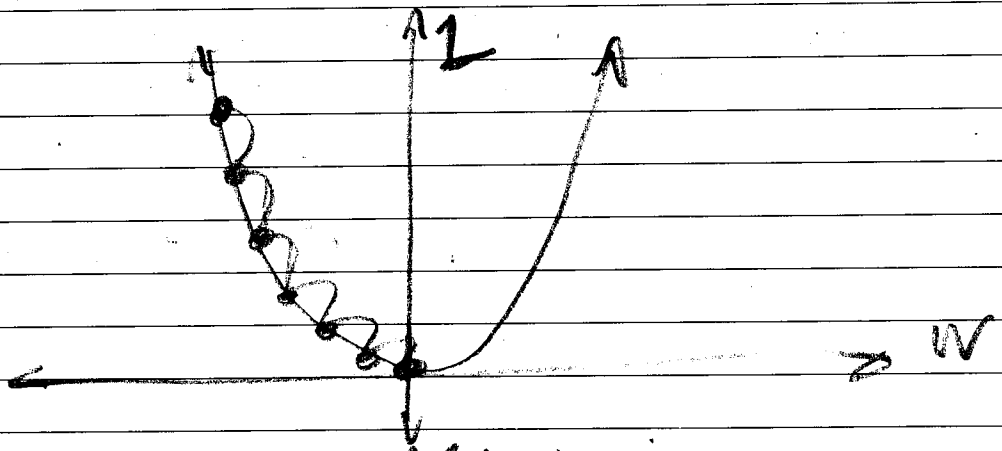
Date.
No.

Sigmoid Decision Boundary: can make our classification by setting a threshold on our Sigmoid function



Inventing Gradient Descent:

how can we minimize our loss function?



$$w \leftarrow w - \alpha \frac{dL(w)}{dw} \quad \alpha = LR$$

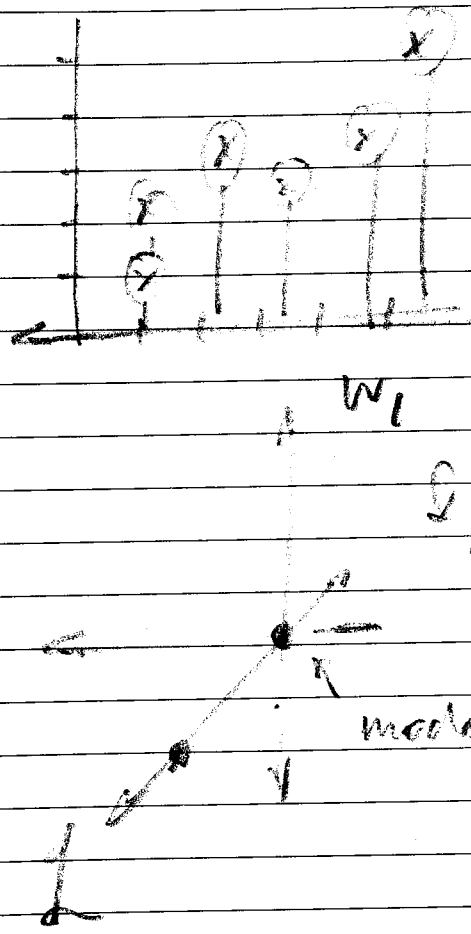
Perceptron Learning to Gradient Descent

Perceptron: $w \leftarrow w + \eta y_i x_i$

Gradient Descent: $w \leftarrow w + \alpha \frac{\partial L(w)}{\partial w}$

Date.
No.

Ex.



$SSE = \sum (y - \hat{y})^2$

$\hat{y} = w_1 x + w_0$

Model = (w_0, w_1)

$L = \sum (y - (w_1 x + w_0))^2$

$\{$ optimal model $\} = \left\langle \frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1} \right\rangle$

1. find the gradient
2. descend

Picturing the Chain Rule: MNIST

[784] $\hookrightarrow$ 28 × 28 image $\rightarrow$ [0 0 1 0 0 0 0 0 0 0] (10)

$(1, 784)(784, 10) \rightarrow (1, 10)$

image · weight matrix → output

but it's useful to make some steps along the way...

$[784] \xrightarrow{\theta} [512] \xrightarrow{\theta} [256] \xrightarrow{\theta} [64] \xrightarrow{\theta} [10]$

$(1, 784)(784, 512)(512, 256)(256, 64)(64, 10)$

Date.
No.

a series of linear transformations is the same as one big linear transformation, so most activation functions here ...

$x^0 = [28, 28] \rightarrow [784]$

L_1
- $z_1 = w^T x$
- $A_1 = \sigma(z_1)$

$L = y - A_2$

$L = y - \sigma(w^T \sigma(w^T x))$

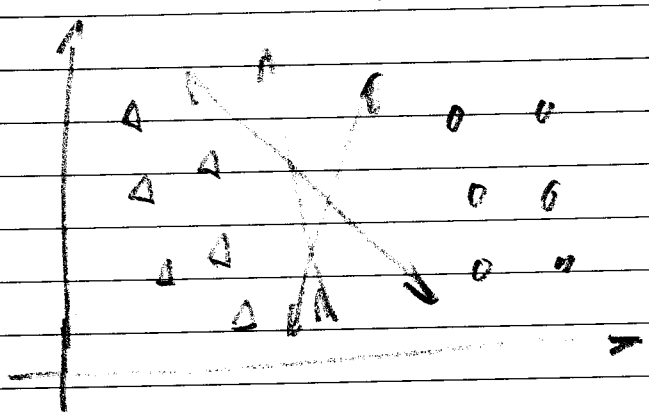
L_2
- $z_2 = w^T A_1$
- $A_2 = \sigma(z_2)$

$\frac{\partial L}{\partial w} = \ldots$ CHAIN RULE

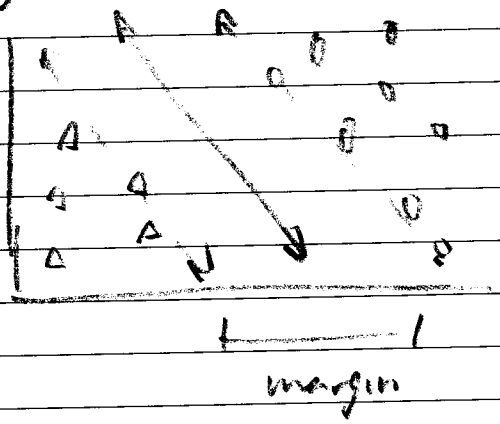
SVMs and KNNs

Date. 10.13.25
No. 15

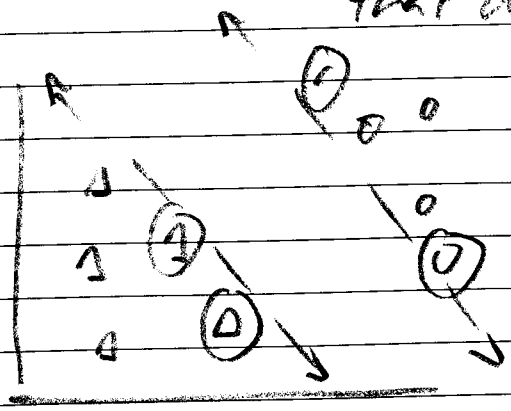
Linear Classifiers: given the dataset below, what would a Perceptron draw a separating line?



Margin Classifiers: find the best such line



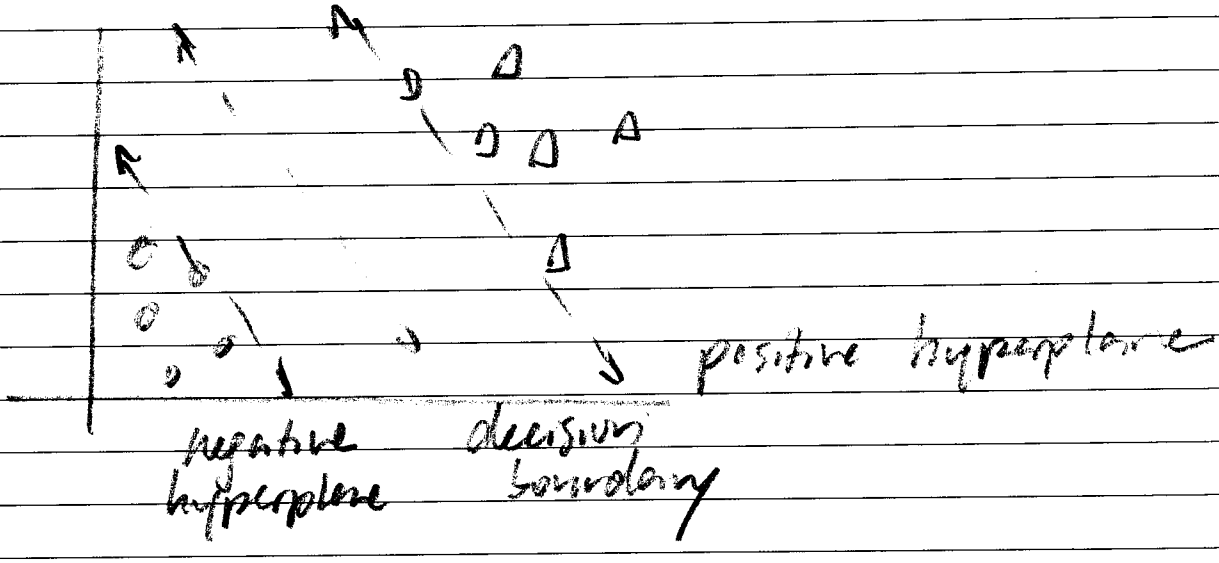
Support Vectors: the points on or in the margin that defines the decision boundary



Date.
No.

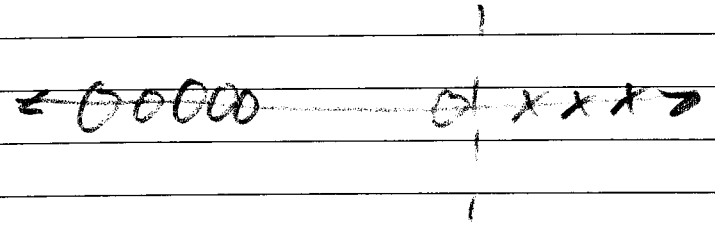
Support Vector Machines:

a Binary classifier that focuses on the concept of a margin to make more robust and efficient classifier

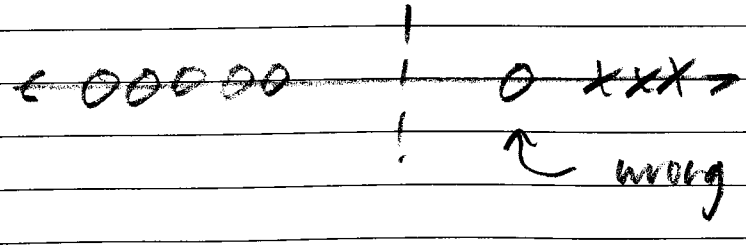


Hard-Margin vs. Soft-Margin SVMs:

Hard-Margin SVMs allow no misclassifications, much like perceptrons



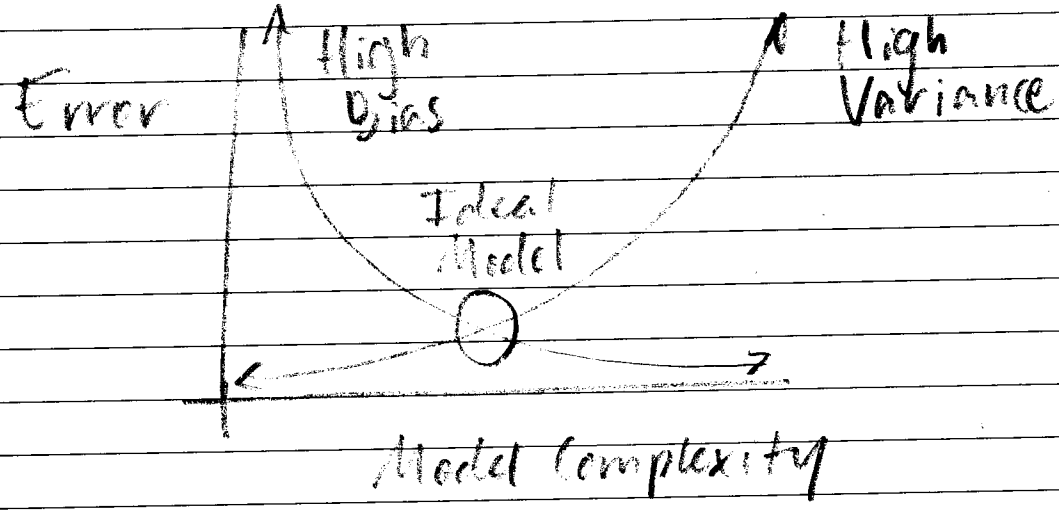
Soft-Margin SVMs allows for some misclassifications to allow for a more robust decision boundary



Date.
No.

The bias-variance tradeoff:

Underfitting and Overfitting are symptoms of having a high bias or high variance



High Bias (Low Variance)
- underfitting
- dumb model
- High error

High Variance (Low Bias)
- Overfitting
- Too complex
- Memorizes errors
- High error

Hard-Margin Classifier Definition:

$$\min \frac{1}{2}\|w\|^2 \text{ st } y(w^T x - b) \geq 1$$

- $\min \frac{1}{2}\|w\|^2$: maximize our margin
- $y(w^T x - b) \geq 1$: $y \cdot \hat{y}$ ie all points are correctly classified
- ≥ 1: all points are on or outside of the margin

y	$\hat{y}$	pred
-1	-1	1
1	-1	-1

Date.
No.

Lagrangian Multipliers:

a strategy for finding the local maxima and minima of a function subject to constraints

$$\min \frac{1}{2}\|w\|^2 \text{ st } y(w^T x - b) \geq 1$$

(goal) (constraint)

allows us to convert st. into a loss function of

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 + \sum \alpha (1 - y(w^T x + b))$$

Hard-Margin SVM Optimization:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 + \sum \alpha (1 - y(w^T x + b))$$

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum \alpha y x \qquad \frac{\partial \mathcal{L}}{\partial b} = \sum \alpha y = 0$$

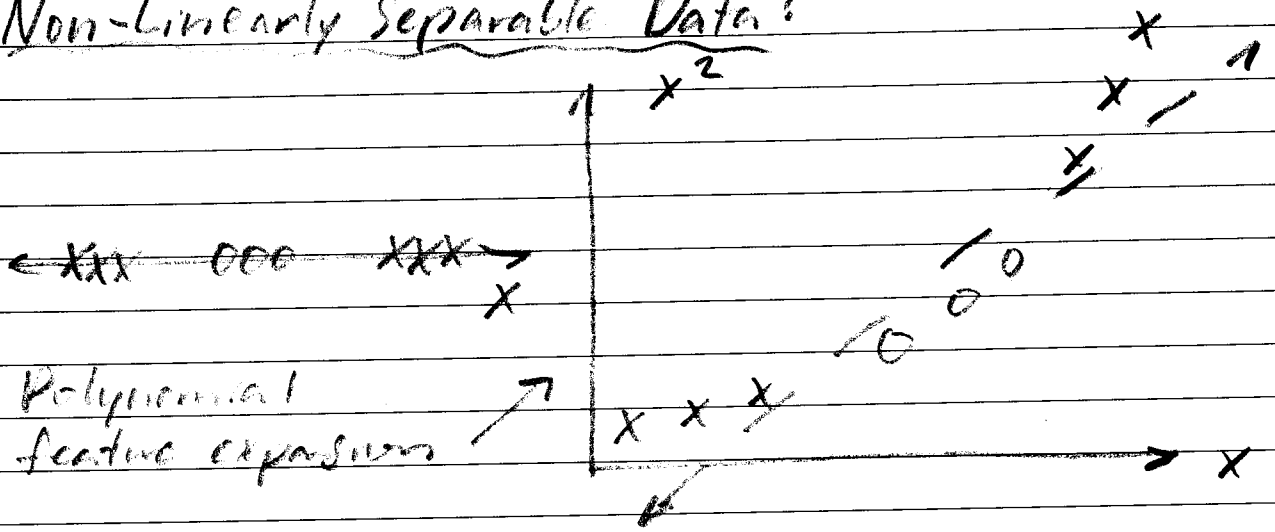
$$\boxed{w = \sum \alpha y x}$$

note. α : lagrangian multiplier

$$\max \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \boxed{(x_i^T x_j)}$$

Date.
No.

Non-Linearly Separable Data:



classifying your space is very powerful!

The Dual-Form Equations:

$$\max \vec{1}^T \alpha - \frac{1}{2} \vec{1}^T \vec{1}^T \alpha \alpha y y \boxed{(x^T x)}$$

only dependent on dot prod between two vectors
→ no more weights!

Kernel Function:

$$k(x, x) = \phi(x)\phi(x)$$

The Kernel Trick:

what makes SVMs so powerful is their ability to implicitly project points to a higher dimension

Date.
No.

Common Kernels: There are 3 common kernels

- Linear: $x_i^T x_j$
- Polynomial: $(x_i^T x_j)^d$

* Radial Basis Function: $\exp(-\gamma \|x_i - x_j\|^2)$
↑ infinite dimensions

Aside: The MNIST database

"the fizzbuzz of computer vision"

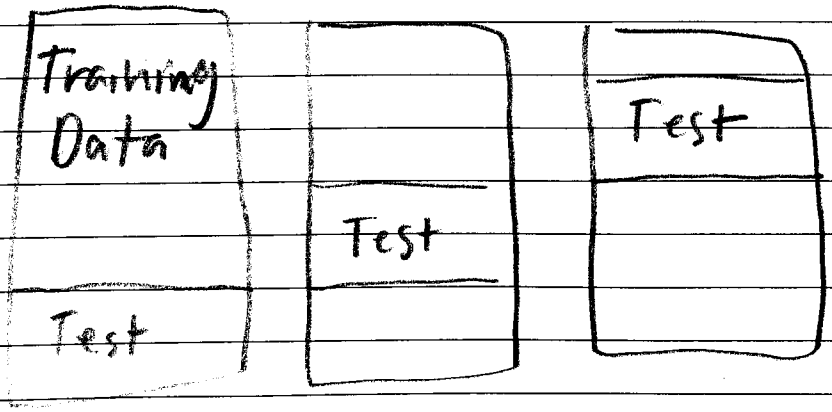
SVMs for Multiclass Classification:

MNIST is a multiclass classification problem. There are 2 common methods for handling this:

One-vs-Rest (OvR): one SVM per class vs all other classes (10 for MNIST)

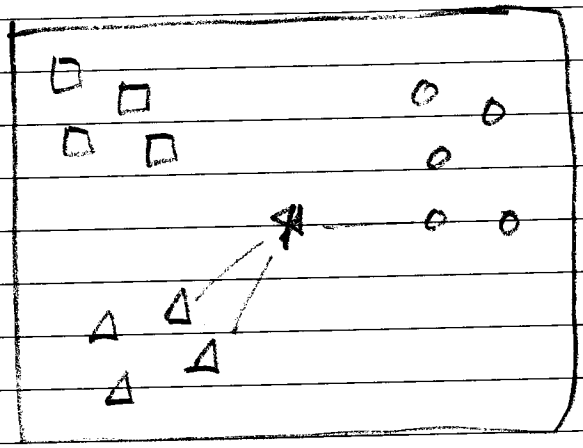
One-vs-One (OvO): one SVM per pair of classes (45 for MNIST)

K-fold Cross Validation: strategy for testing your model to help calculate more accurate results



Date.
No.

k-Nearest Neighbors : kNNs



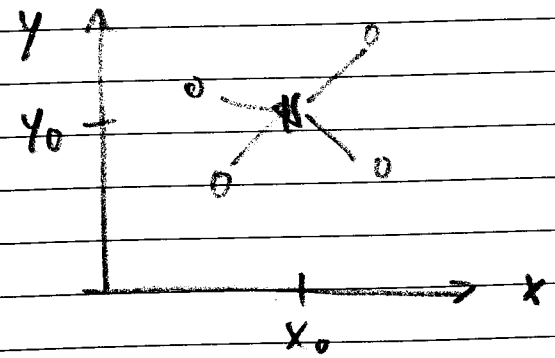
k=1 : * is nearest other class (○)
k=3 : * is nearest 3 other classes (△△○)
(picks the majority)

Voting Methods in kNN :

1) Majority vote
2) Distance Weighted Vote *
3) Rank Weighted
4) Probability

* close neighbors get more weight, using an inverse distance function

kNN Regression :



* if algorithm has no weights → non-parametric
if weights → parametric

Date.
No.

Parametric vs Non-Parametric

Parametric Models assume a fixed structure with a limited number of parameters (weights)

Ex. Linear Regression
Naive Bayes
Neural Networks

Non-parametric: " " no weights!

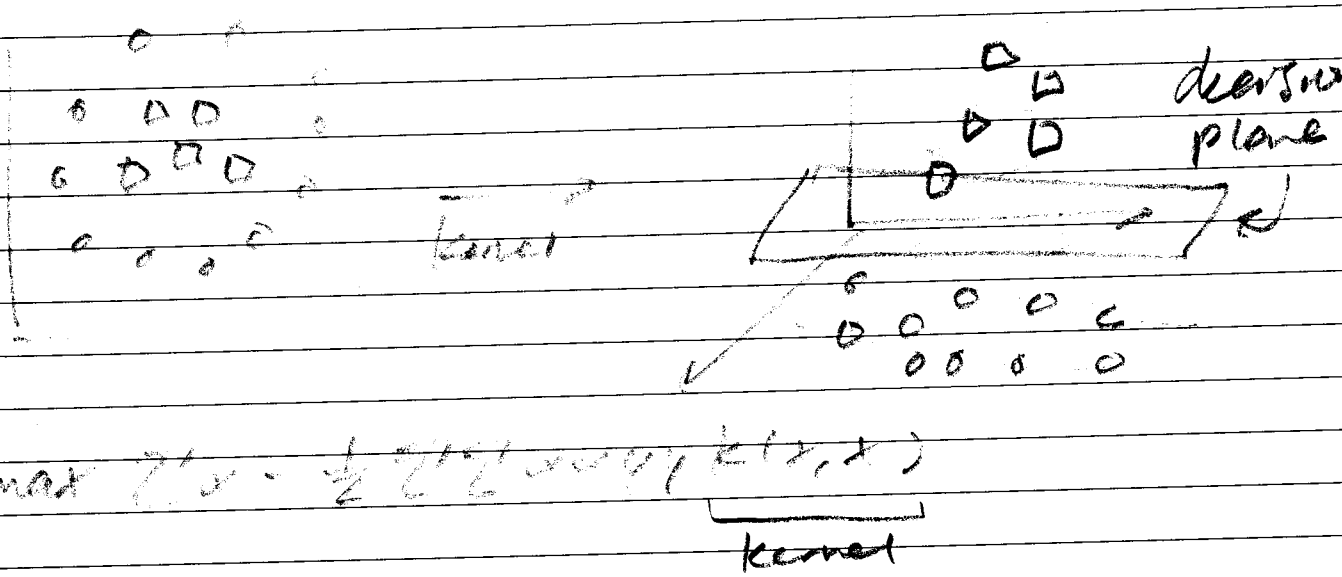
Exs: kNN
decision trees
some SVM kernels

The Curse of Dimensionality: as the number of features (dimensions) grows, the amount of data we need to generalize accurately grows exponentially

Unsupervised Learning and Clustering

Date. 10.15.25
No. 16

Support Vector Machines:



max $\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$

kernel

quadratic complexity: runs on GPUs

Recall: k-Nearest Neighbors (kNN)

Supervised Learning: is when we have ground truth from all of our sample vectors

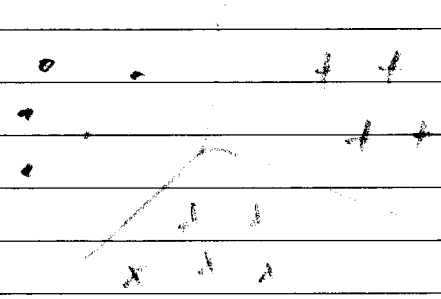
i.e. we learn by comparing our ground-truth output to our predicted output and try to bring the two closer by minimizing our cost function

Unsupervised Learning: no more ground truth

instead we discover patterns by comparing points against each other

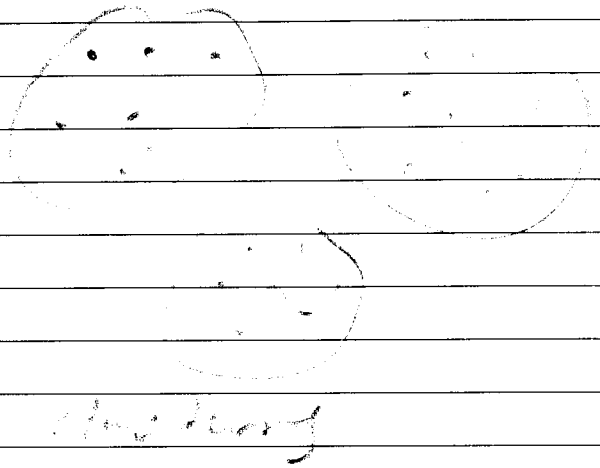
Date.
No.

Supervised



classification

Unsupervised

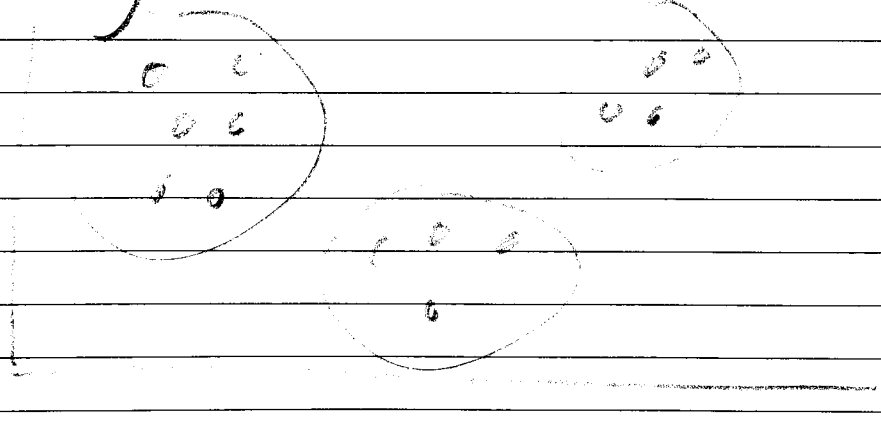


clustering

Goal of Unsupervised Learning: We want to discover hidden patterns in our data to answer:

1. Is there an interesting way to visualize our data?
2. Are there subgroups within our data? - clustering

Clustering: most common unsupervised learning task



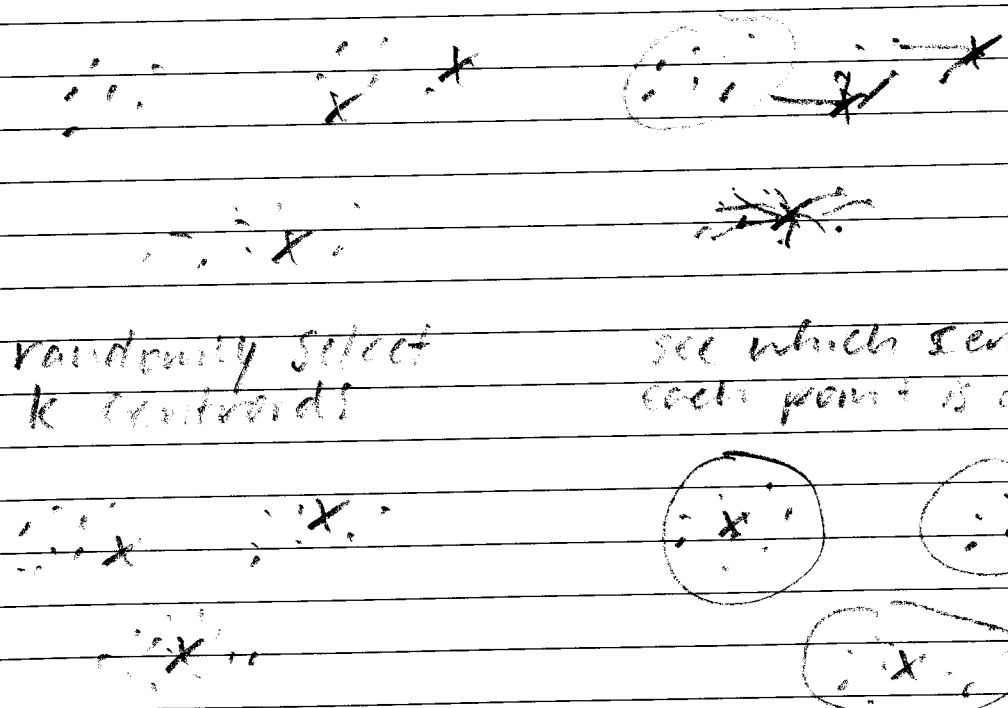
Date.
No.

K-means Clustering: "most common" clustering algorithm

Assumptions:
1. You know how many clusters you want
2. Inner Groups are Similar

But even it makes these assumptions, it always comes up with an answer!

Visualizing k-means:



* k-means algorithm in slides *

note, no way to really tell whether your clusters are good

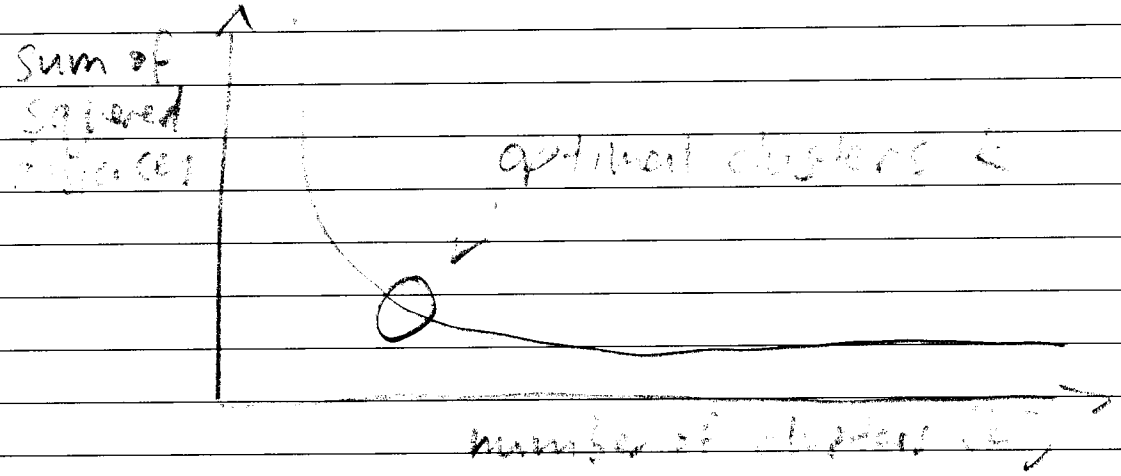
Date.
No.

Evaluating Unsupervised Learning:

Because k-means is unsupervised, we don't have ground-truth

So it's tricky to assess how good our clusters are

Elbow Plots:



k-means:

Pros:
- intuitive and easy to implement
- guaranteed to work

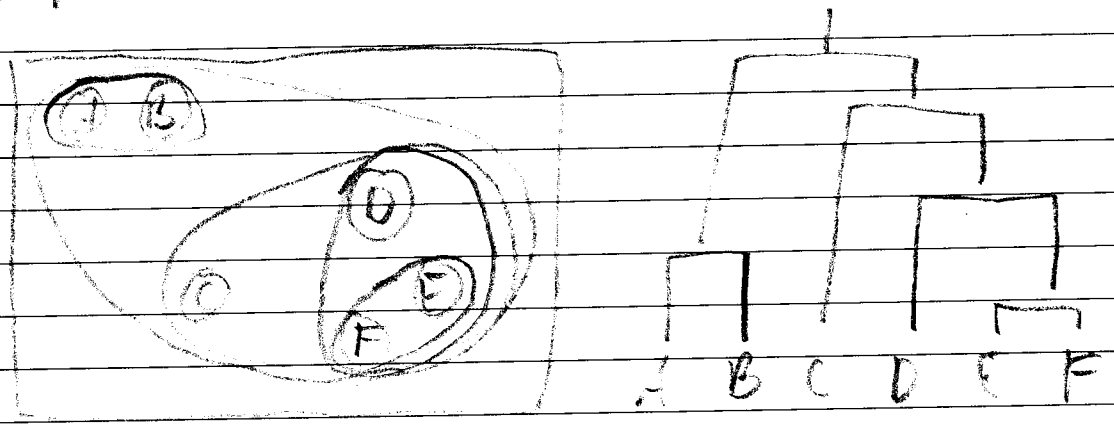
Cons:
- Number of clusters is a hyperparameter
- May converge to local minima

* kmeans is also in sklearn - on slides *

Date.
No.

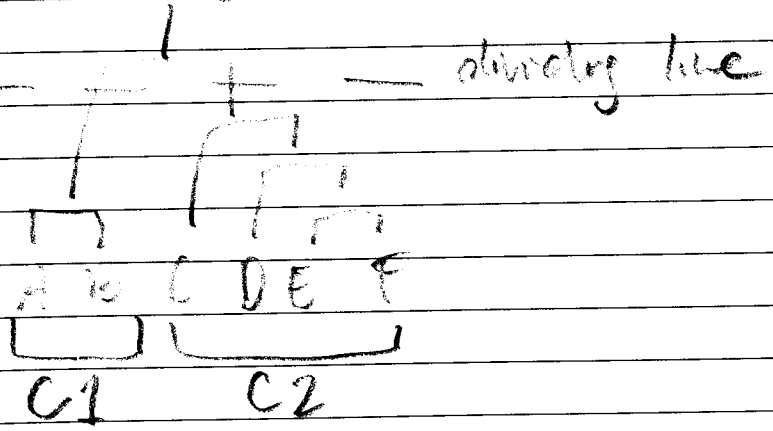
Hierarchical Clustering:

We build a dendrogram, a tree like representation of our samples



Getting clusters from dendrogram

Any points joined below the dividing line is a cluster



Methods for Building Dendrograms:

Agglomerative: Start small, end big
i.e. bottom up

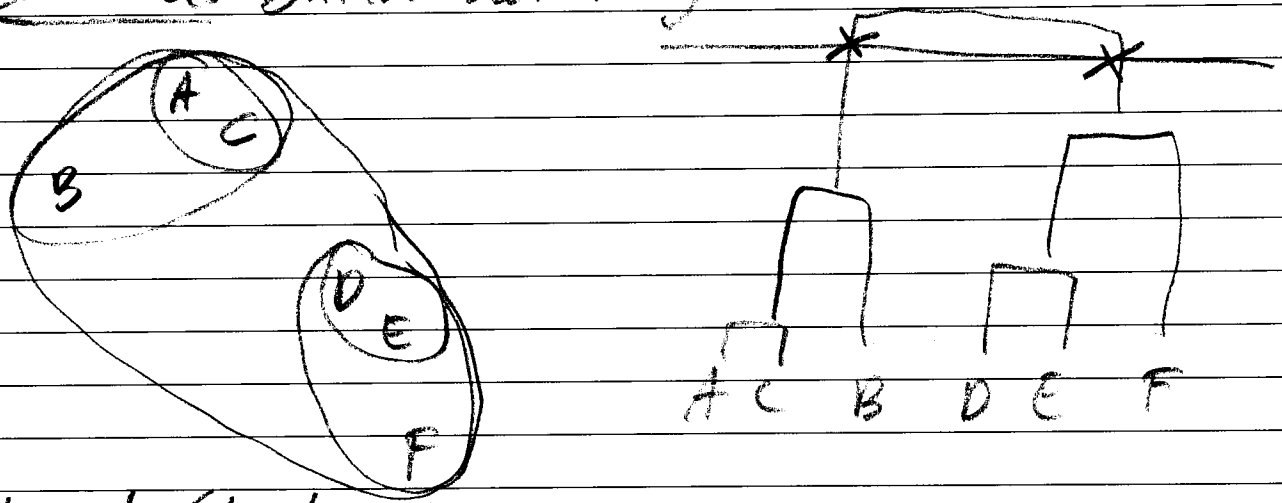
Divisive: Start big, end small
i.e. top down

Date.
No.

Agglomerative Hierarchical Clustering

1. Compute the proximity matrix
2. Set each point to its own cluster
3. Merge the two closest clusters and update the proximity matrix
4. Repeat (3) until only one cluster

Exam Q. Build dendrogram



Closest Clusters:

Single Linkage: minimum distance between the closest elements in clusters

Complete Linkage: maximum distance between closest elements " "

Average Linkage: average " "

Centroid Method: centroid " "

Ward Method: average of the squared distances of all pairs of nodes

Date.
No.

Summary:

Pros.
- # of clusters not a hyperparameter
- Dendrogram is easily visualizable
- Deterministic

Cons
- High computational complexity
- Sensitive to outliers
- No clear objective function

lots of other clustering methods — # clusters

Clustering for Image segmentation: process of separating the pixels in an image into different objects or regions

2 clusters → background and foreground clusters (based on background and foreground colors)

Ground truth for image segmentation → binary mask (can be used to evaluate image segmentation)

Compute intersection over union, or Jaccard Index

$$IoU = \frac{\text{area of overlap}}{\text{Area of Union}}$$

Date. 10.27.25
No. 17

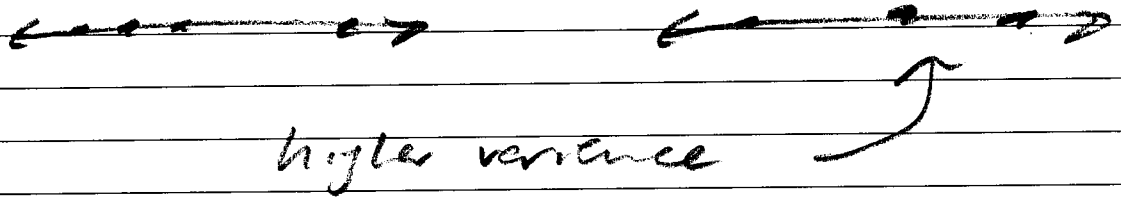
Dimensionality Reduction

Principle Component Analysis (PCA):

unsupervised learning algorithm that finds a new set of axes for our data in which our features have no relation (0 covariance)

What separates data? variance!

data is only useful if samples are different



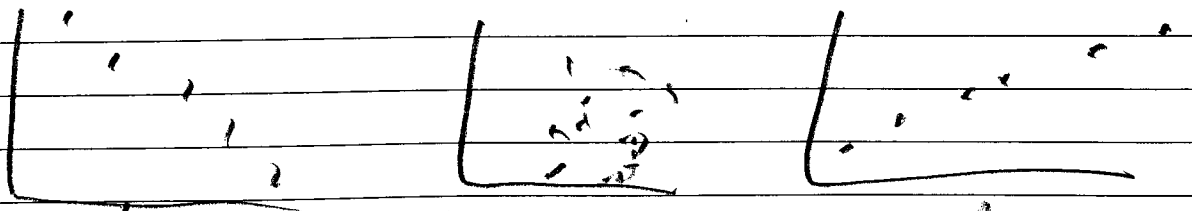
higher variance

Total variance of a Data Set:

$$s^2 = \frac{\sum (x - \bar{x})^2}{N - 1}$$

Higher the variance of a feature, the more information it can give us!

Variance between features:



negative covariance | no covariance | positive covariance

Date.
No.

Covariance:

the more two features overlap, the more redundant information they carry.

helps us determine whether we need all features

PCA tries to find a more informative "view" of the data that minimizes covariance

Finding the Principle Components

1. Standardize our data (0 mean and unit variance)
2. Calculate the covariance matrix
3. Compute the eigenvalues and eigenvectors
4. Sort the eigenvalues/vectors
5. Choose principal components

ex of this in slides

2) $cov = \begin{bmatrix} var(x_1) & x_1 \cdot x_2 \\ x_1 \cdot x_2 & var(x_2) \end{bmatrix}$

3) $det(C - \lambda I) = 0$

eigenvalues: amount of variance contained by each principle component

eigenvector: direction along which the principle components point

Date.
No.

Singular Value Decomposition (SVD):

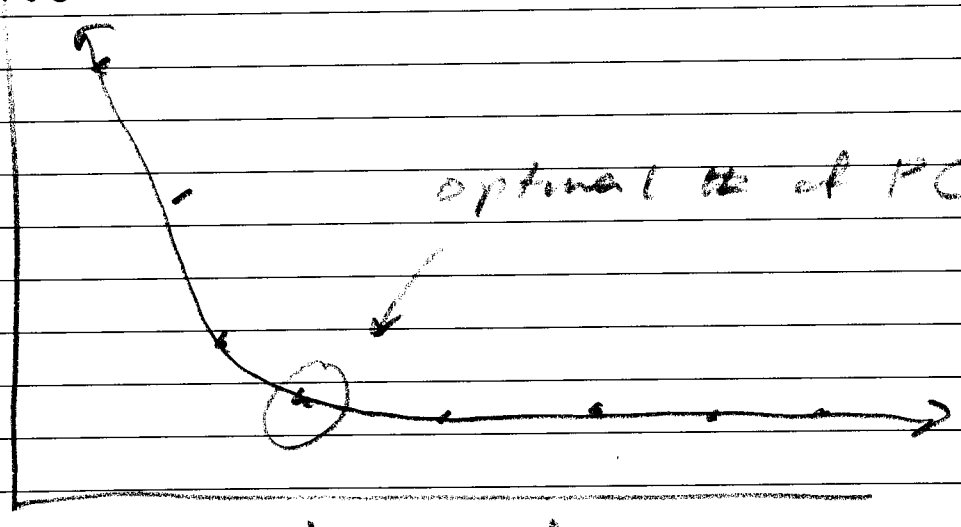
$$A = U \Sigma V^T$$

i.e. just a linear transformation

↳ this is just a generalization of the PCA math.

V^T: contains eigenvectors
Σ: contains eigenvalues

Interpreting PCA Results (scree plot):



input space vs. feature space

converting input into a more usable space

Practice

Date. 10.29.25
No. 18

Accuracy: how close a given set of measurements are to their actual values

		Predicted/Classified Neg	Pos
Actual	Neg	998	0
	Pos	1	1

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Confusion Matrix: helps assess classification model performance

↳ this is a Confusion Matrix

	Predicted Value	
Actual Value	TP	FN
	FP	TN

Type 1 vs Type 2 Errors:

AV \ PV		Type 2
	Type 1	

Type 1: false positives
Type 2: false negatives

Date.
No.

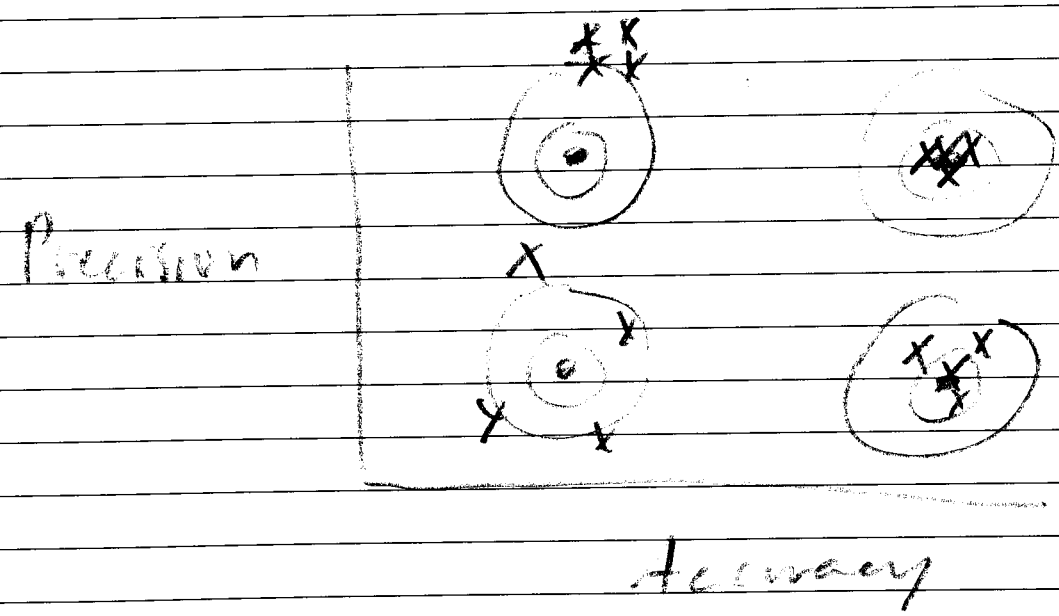
Evaluation Metrics :

- Accuracy
- Precision
- Recall
- F1 Score

Precision : how close measurements are to each other

$$\text{Precision} = TP / (TP + FP)$$

Visualizing Accuracy and Precision



Recall: Proportion of actual positives that were classified correctly

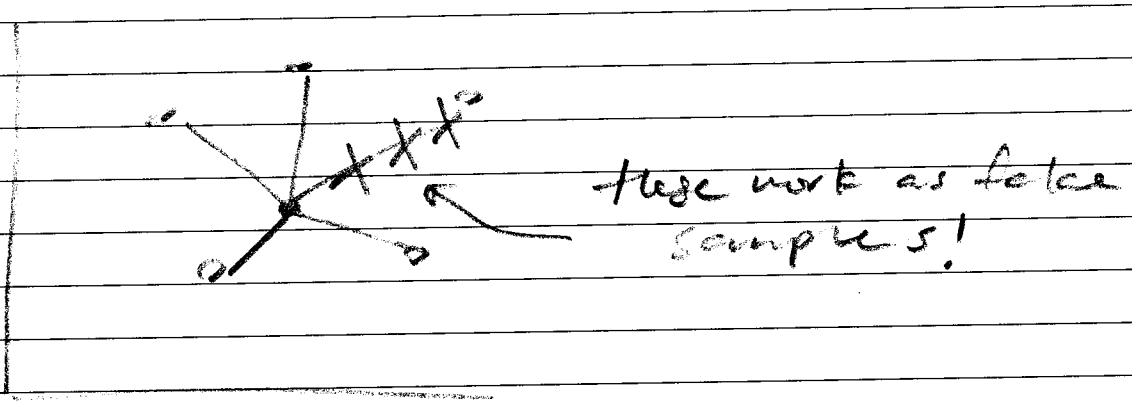
$$\text{Recall} = TP / (TP + FN)$$

F1 Score : balance between precision and recall

→ good for imbalanced datasets

Date.
No.

SMOTE: synthetic minority oversampling technique

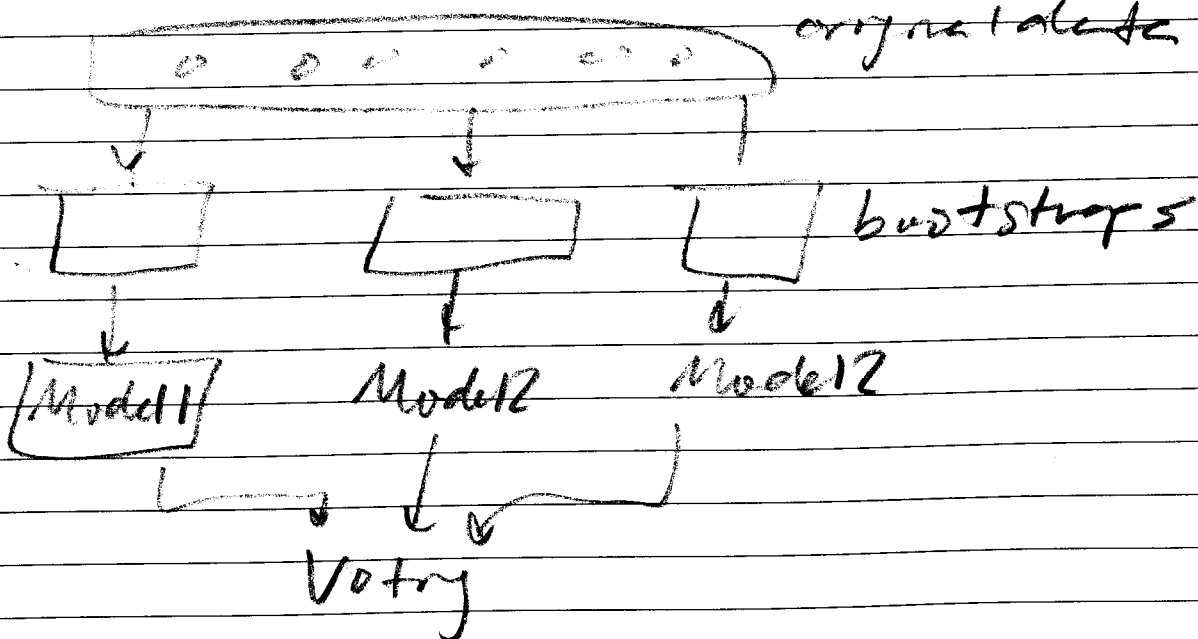


Ensemble Learning: combine models together to improve performance

Bootstrapping: Sampling method using random sampling with replacement

Bagging: Bootstrap aggregating

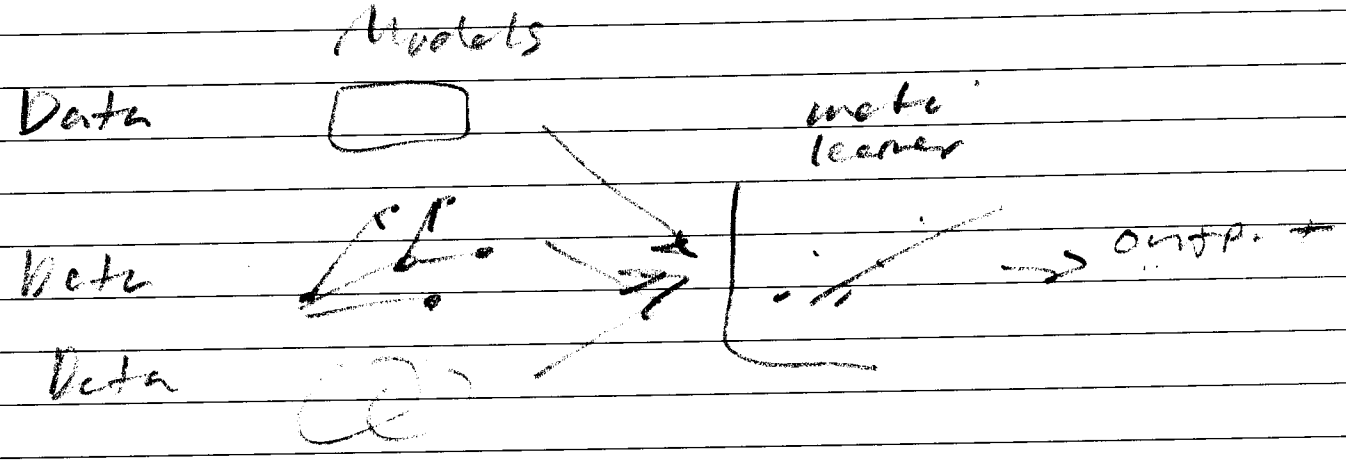
Voting:



Boosting: train models sequentially

Date.
No.

Stacking: pass outputs of simple models as input into a "meta-learner"



Accuracy vs. Precision vs. Recall:

Accuracy: when class distribution is balanced and FP and FN have similar consequences

Precision: when FP are costly or undesirable

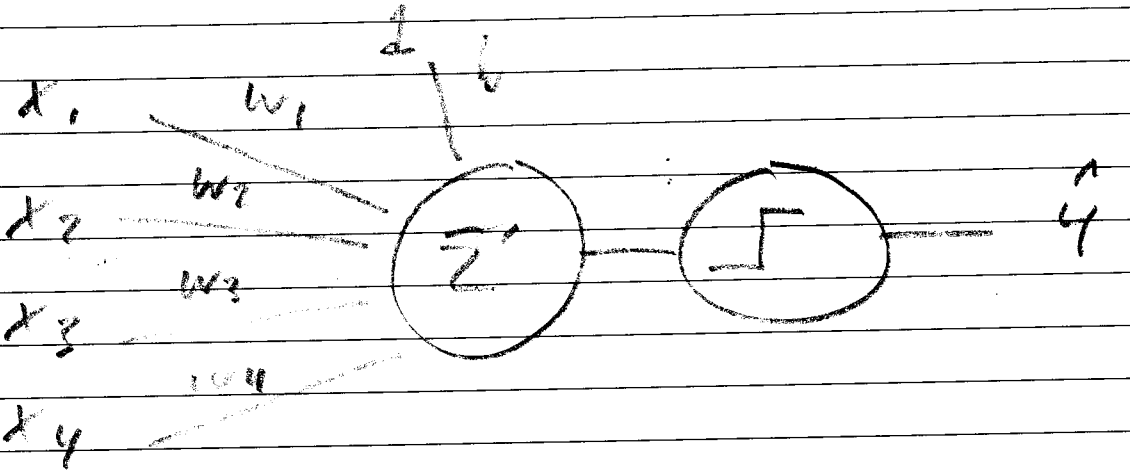
Recall: when FN are more critical than FP

F1: when you need a balance between precision and recall and the class distribution is imbalanced

Deep Learning: FFNs/MLPs

Date. 11.3.25
No. 19

Recall. Perceptrons



$f(\sum x_i w_i + b)$ st $f = \begin{cases} z > 0 \text{ then } 1 \\ z < 0 \text{ then } 0 \end{cases}$

Recall. Logistic Regression

$f(\sum x_i w_i + b)$ st $f = \frac{1}{1+e^{-z}}$

uses a logistic activation function instead of a step function so it is differentiable

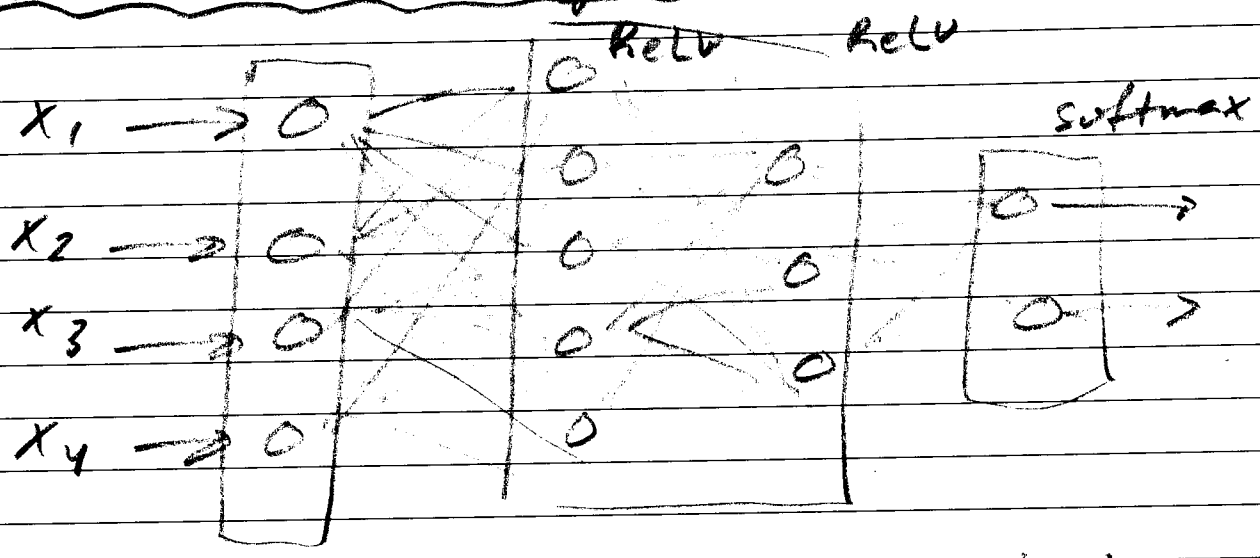
Generalized Artificial Neuron:

$f(Wx + b)$ st f not linear and differentiable

Deep Learning: subset of ML that uses artificial NN to do classification, regression, and representation learning

Date.
No.

Neural Networks: Diagram



input layer | hidden layers | output layer

- the use of hidden layers is what makes ML → deep learning

Architecture:

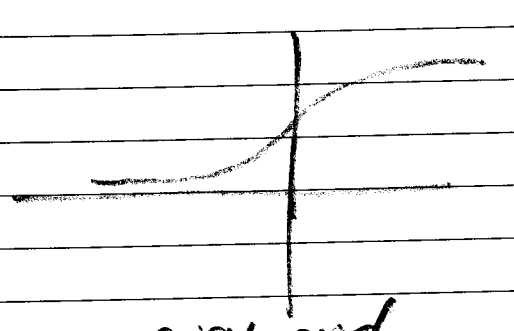
- # of layers (depth)
- # of neurons per layer (width)
- types of layers used (fully connected, convolutional)
- activation functions used

Activation Functions: introduce nonlinearity into the model. Allows networks to learn complex patterns

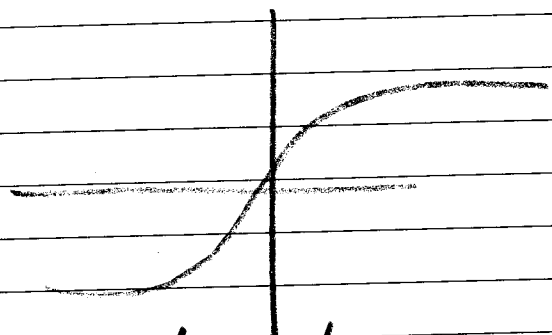
1. Sigmoid
2. Tanh
3. ReLU
4. Softmax

Date.
No.

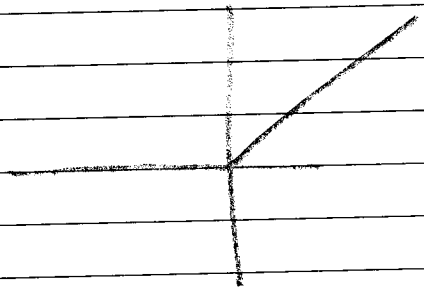
Activation functions cont:



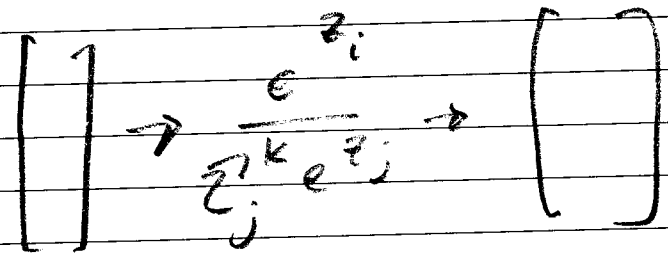
sigmoid



tanh



ReLU
most widely used



* Softmax used as the last layer of an activation function in a NN to create final output probabilities

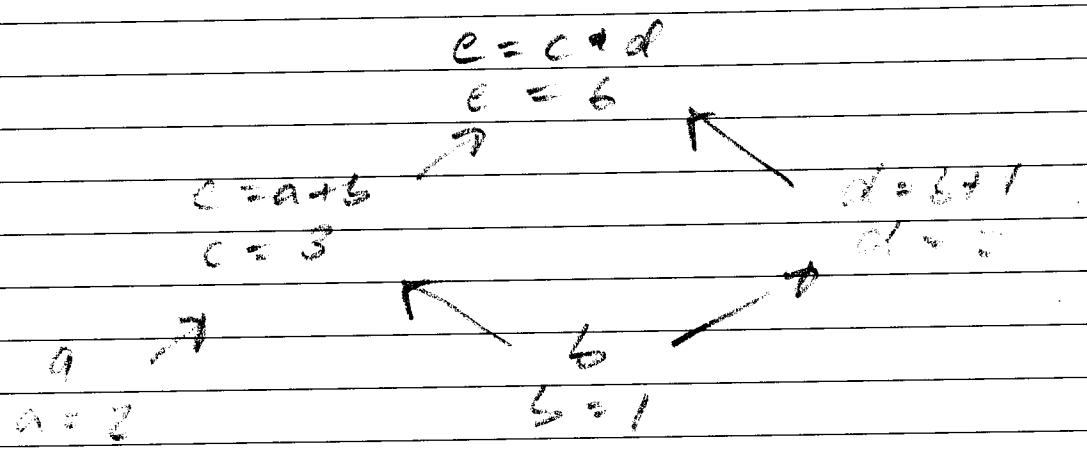
FFN/MLPs: feed-forward neural networks (also known as multi-layer perceptrons)

- data flows in one direction (input → output)
- layers are fully connected (or dense linear layers)

Date.
No.

Computation Graphs: directed graph used to evaluate math expressions

blueprint of functions performed during evaluation



i.e. $y = (a+b)(b+1)$

FFN Computation Graph:

$$X$$
$$\downarrow$$
$$z_1 = W_1 X + b_1$$
$$\downarrow$$
$$A_1 = ReLU(z_1)$$
$$\downarrow$$
$$z_2 = W_2 A_1 + b_2$$
$$\downarrow$$
$$A_2 = ReLU(z_2)$$
$$\downarrow$$
$$z_3 = W_3 A_2 + b_3$$
$$\downarrow$$
$$\hat{y} = softmax(z_3)$$

Gradient Descent and Backpropagation

Date. 11.5.25
No. 20

Loss Function: measures how far the network's predictions are from the ground truths

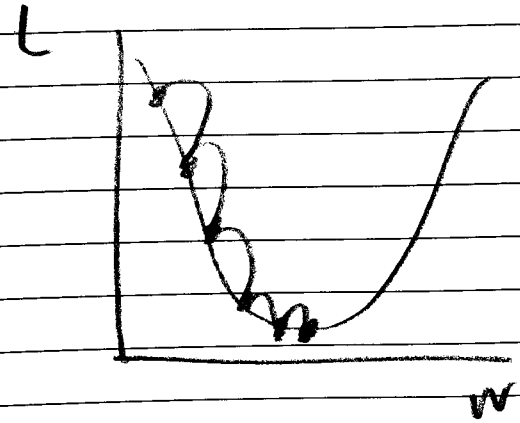
i.e. function of weights and biases that measures error

Backpropagation: method to compute how each weight and bias should change to reduce the loss

- determines direction and magnitude of change for every parameter
- uses the chain rule to get partial derivatives of the loss function with respect to each weight and bias

i.e. gradient of each learnable parameter (shows how sensitive loss fn is to each parameter)

Gradient Descent: algorithm to minimize loss fn by updating weights / biases in the direction that reduces error



$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w}$$

Date.
No.

Feature Engineering: early days of ML

- models relied heavily on manually extracted features

Representation Learning: modern NN

- models automatically learn features from raw data

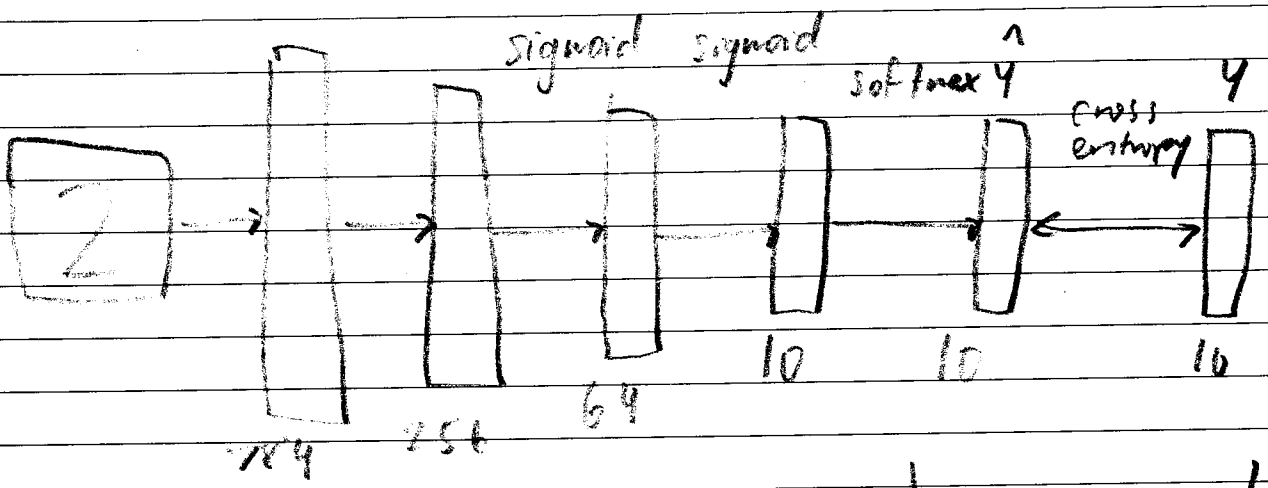
Latent Space: grouping similar inputs close together in space

- typically a lower dimensional representation of high-dimensional data
- still captures data's underlying features and structure

Computer Vision

Date. 11.10.25
No. 21

Recall. MNIST Feed-forward Network



note.

$\hat{y} = \text{argmax}([0, 0.5, 0.8, \ldots, 0.0])$,

$y = [0, 0, 1.0, \ldots, 0]$

Exam Q's for networks:

a) draw the computation graph for the network
b) give the derivative chain for calculating the gradient of the bias in the first layer
c) give the equation you'd actually use to calculate the gradients

Date.
No.

Ex.

a)

$x : \langle 784 \rangle$

w_1 → $z_1 = w_1^T x + b_1$ ← b_1

changes the dims ($784 \to 256$)

$A_1 = \sigma(z_1)$

w_2 → $z_2 = w_2^T A_1 + b_2$ ← b_2

$256 \to 10$

$\hat{y} = \text{softmax}(z_2)$

$\mathcal{L}_{CE}(\hat{y}, y)$ ← y

b) how would you update b_1?

$$b_1 \leftarrow b_1 - \alpha \frac{\partial \mathcal{L}}{\partial b_1}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial A_1} \frac{\partial A_1}{\partial z_1} \frac{\partial z_1}{\partial b_1} \quad \text{... derived from:}$$

$$\mathcal{L}(\text{softmax}(w_2^T \sigma(w_1^T x + b_1) + b_2), y)$$

note.

$$\frac{\partial \mathcal{L}}{\partial w_1} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_{11}} & \frac{\partial \mathcal{L}}{\partial w_{12}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{1\,784}} \\ \vdots & & \ddots & \\ \frac{\partial \mathcal{L}}{\partial w_{256\,1}} & \cdots & & \frac{\partial \mathcal{L}}{\partial w_{256\,784}} \end{bmatrix}$$

Date.
No.

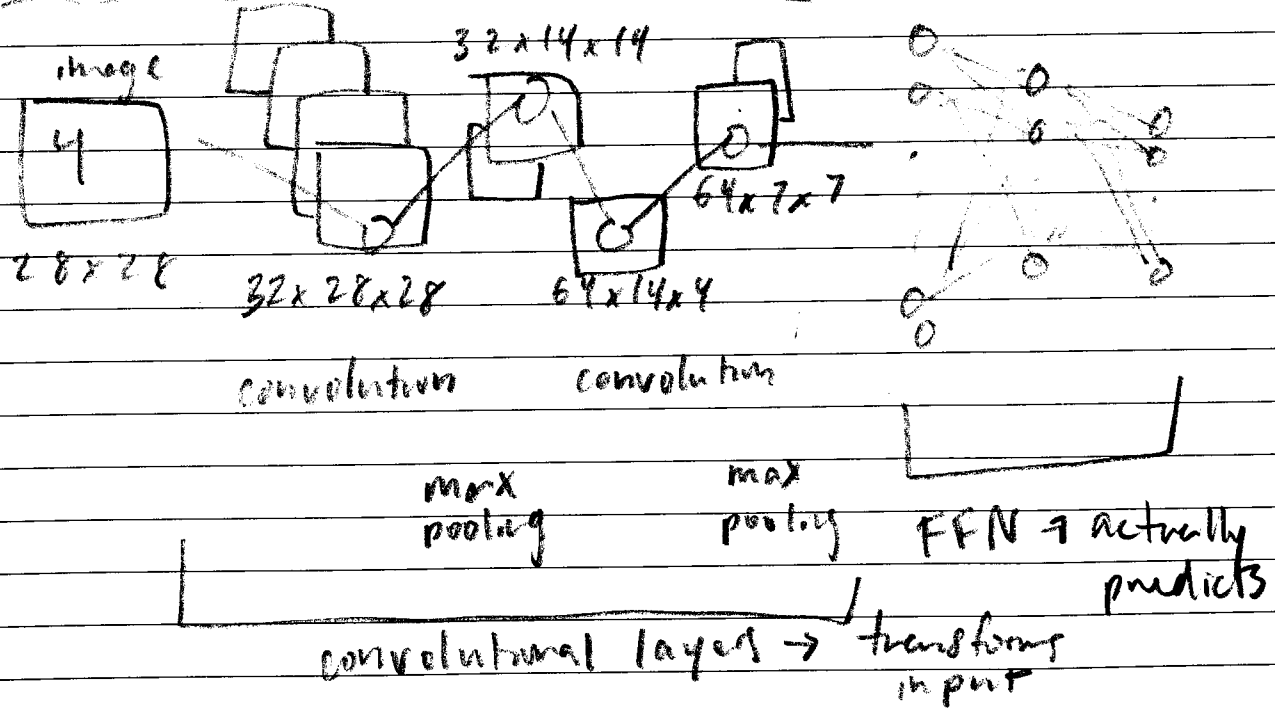
c) $\frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial A_1} \frac{\partial A_1}{\partial z_1} \frac{\partial z_1}{\partial b_1}$

$(\hat{y} - y)$ (w_2) $\sigma(z_1)(1-\sigma(z_1))$ (1)

so $\frac{\partial \mathcal{L}}{\partial b_1} = (\hat{y} - y)(w_2)\sigma(z_1)(1 - \sigma(z_1))$

and $b_1 \leftarrow b_1 - \alpha \frac{\partial \mathcal{L}}{\partial b_1}$

CNN Classification Architecture



Learned Representations: we learn a way to map:

input → feature vector

Date.
No.

Convolutional Neural Networks (CNNs):

a way to hierarchically apply filters to (typically) images that let the network recognize certain patterns

What's a Filter:

a small grid of numbers encoding some pattern we're looking for in an image

1	1	1
0	0	0
-1	-1	-1

Pooling Layers: compute average and max pooling

3	2	-3	2
1	6	20	5
4	-13	2	6
-2	3	9	3

original input

... average of quadrants

6	20
4	9

max pool

note. filter weights are trained just like in FFN, but keeps inputs in 2D

kw_1	kw_2	kw_3
kw_4	kw_5	kw_6
kw_7	kw_8	kw_9

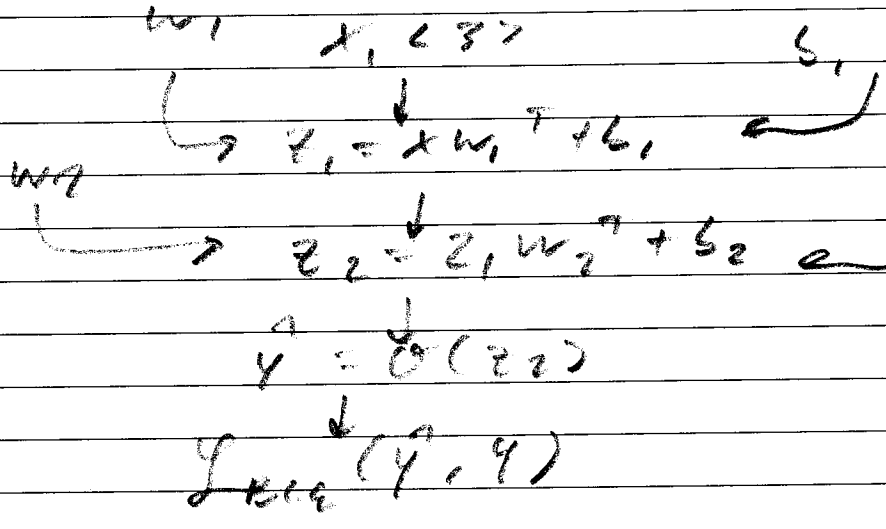
weights trained with backpropagation

Exam 2 Review

Date. 11/12/25
No. 22

Practice Packet Questions:

35) a. Draw the computation graph for the network



note. BCE (binary cross-entropy) for 1D output

just CE (cross-entropy) for >1D output

b) give the nested function for calculating the loss:

$$L = \mathcal{L}_{BCE}(\sigma((xW_1^T + b_1)W_2^T + b_2), y)$$

c) give the derivative chain for the gradient of the bias in the first layer

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial b_1}$$

d) give the equation you'd actually use to calculate the gradient

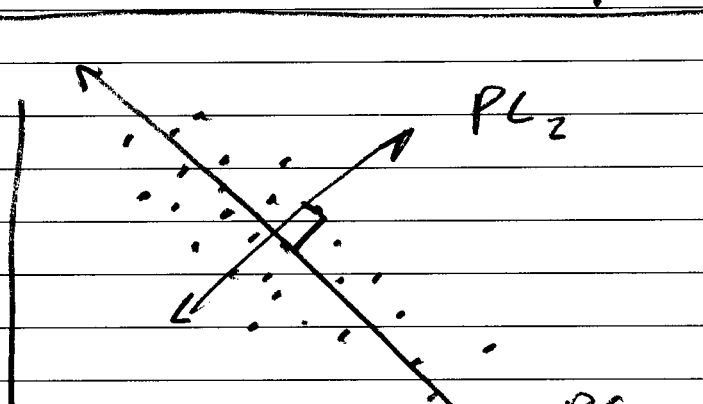
$$\frac{\partial \mathcal{L}}{\partial b_1} = (\hat{y} - y) W_2^T (1)$$

Date.
No.

Activation Functions: what allow NN to capture non-linear relationships

softmax used on final output layer of non-binary output

PCA and tsne (Dimensionality Reduction):



20) Draw the two principle components on the plot and label them

* PCA captures maximal amount of variance *

Common SVM Kernels:

Linear: $k = x_i^T x_j$

Polynomial: $k = (x_i^T x_j + c)^d$

* Radial Basis Function: $k = \exp(-\gamma \|x_i - x_j\|^2)$

Sequence Networks

Date. 11/19/25
No. 24

Recall: The networks we've discussed so far take in single samples that lack context

Sequence Data: How do we maintain information over time?

Ex.
- weather forecasting
- stock prices
- text, audio, video

Language as an Input: computers understand only numbers so we need to create numbers for input to enable computer understanding

ie Natural Language Processing (NLP)

Tokenization: the process of splitting a piece of input text into its component tokens

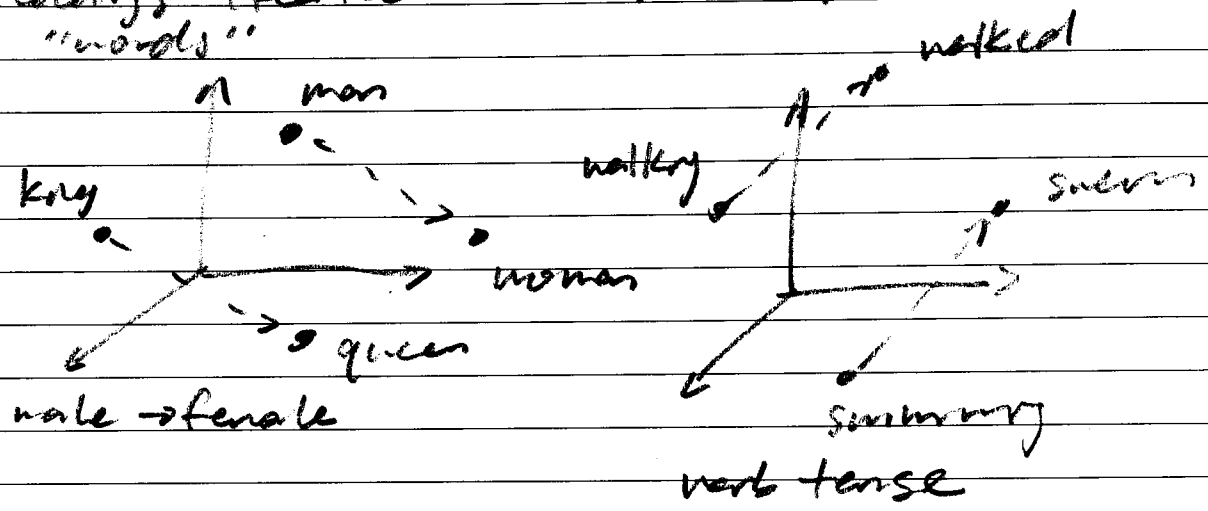
Can be done at different levels:
- word
- character
- byte
- sentence

The goal is to capture natural input while minimizing # of tokens

Date.
No.

Word Embeddings: we convert the tokens into embeddings (feature-vectors) to represent our "words"

ex's.



1 dimension for each feature of natural language

Sequence Networks: RNN

we can begin to solve context by using a Recurrent Neural Network (RNN)

(recurrent blocks/layers are like linear layers, and have architecture in slides)

Equations:

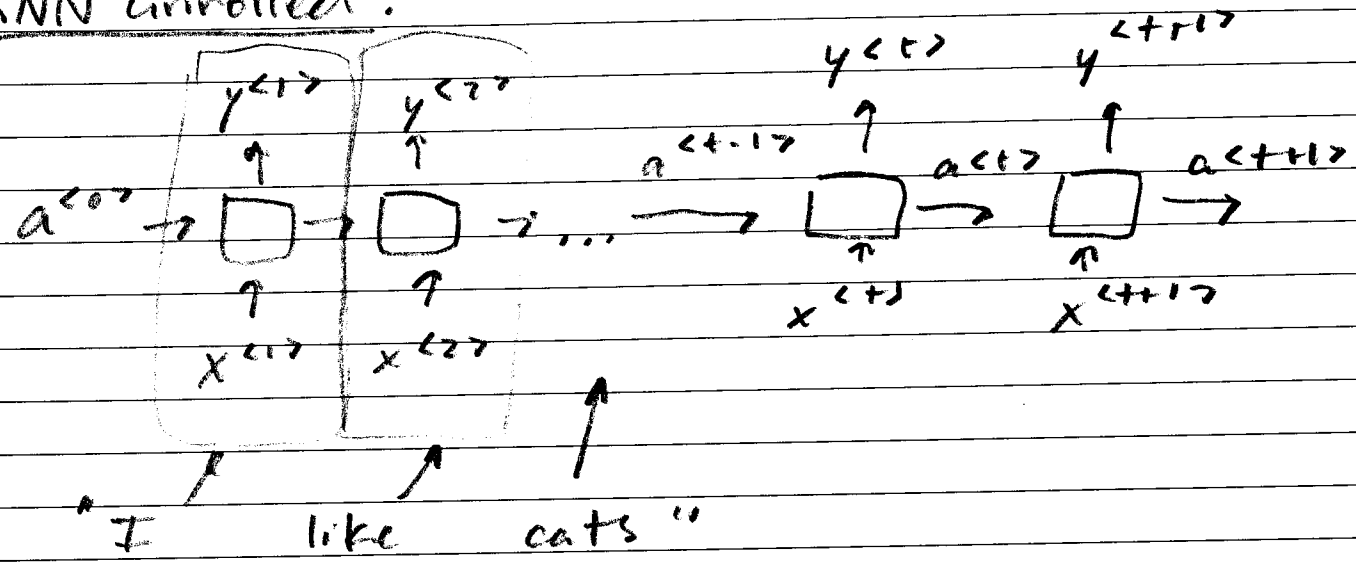
linear-layer: $z = xw^T + b$ FFN
$A = relu(z)$

recurrent-layer: $z = xw_i^T + Hw_h^T + b$ RNN
$H = \tanh(z)$

note. RNN use tanh since it is centered at 0

Date.
No.

RNN unrolled:



* these are recurrent because they recurse on themselves

Computation graph for RNN: only output on final token

"Good luck"

x_1 x_2

$z_1 = x_1 W_i^T + a_0 W_c^T + b_a$

$A_1 = \tanh(z_1)$

$z_2 = x_2 W_i^T + A_1 W_c^T + b_a$

$A_2 = \tanh(z_2)$

$z_3 = A_2 W_o^T + b_o$

$\hat{y} = \text{softmax}(z_3)$

[0.1 0.1 0.8]

<Henry, Leo, Pat>

but we wanted

[1.0 0 0]

backprop:

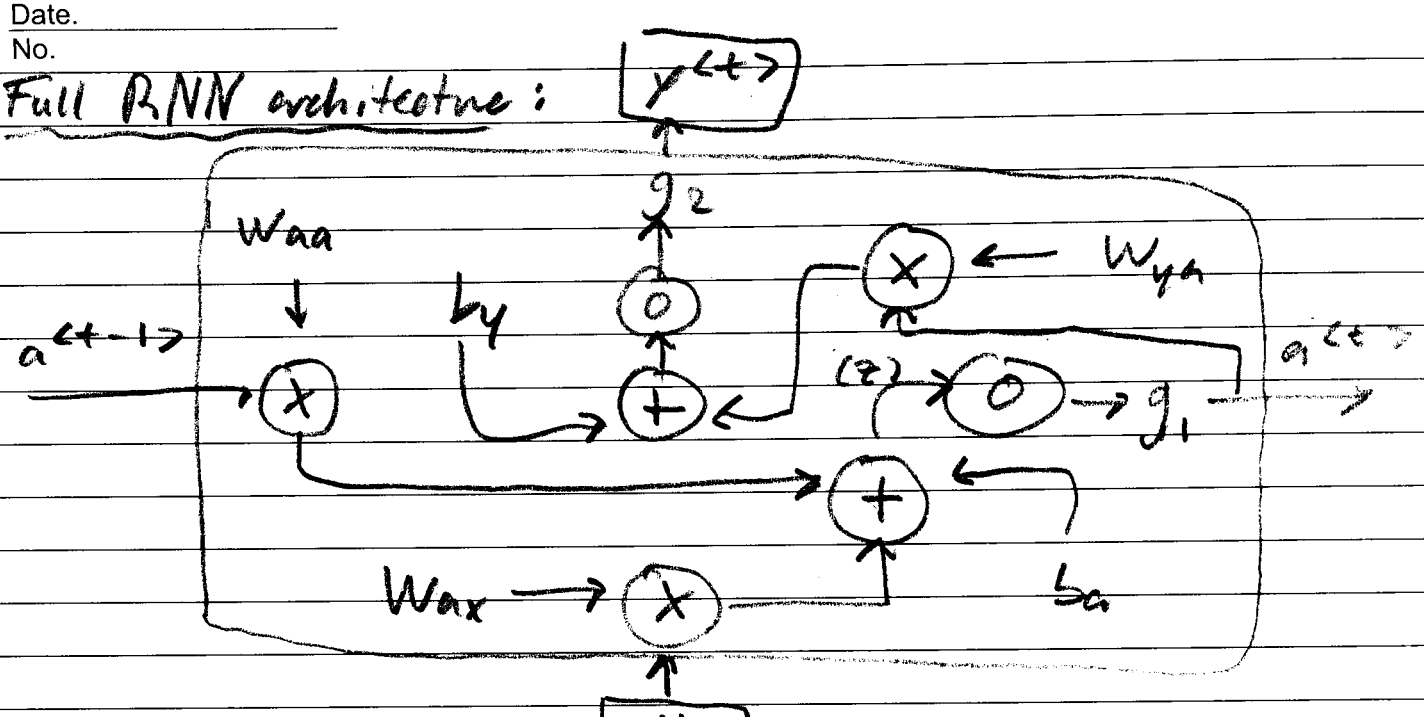
$$\frac{\partial \mathcal{L}}{\partial W_i} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial A_2} \frac{\partial A_2}{\partial z_2} \frac{\partial z_2}{\partial A_1} \frac{\partial A_1}{\partial z_1} \frac{\partial z_1}{\partial W_i}$$

need to go back to the first time W_i showed up since

$z_2 = \tanh(x_1 W_i^T + a_0 W_c^T + b_a) W_i^T + A_1 W_c^T + b_a$

Date.
No.

Full RNN architecture:



g_1 : A
$x^{<t>}$: input with index t
W_{ax} : W_i (weight matrix)
b_a : bias

W_{aa} : W_c (context matrix)

Pros and Cons of RNNs:

Pros:

- Any length input
- input size != model size
- can remember past information

Cons:

- slow
- vanishing gradient

Date.
No.

Vanishing / Exploding Gradients: RNNs are vulnerable to this because we compute the gradient backwards through time

RNN Advances: LSTMs and GRUs

Long Short-term Memory — LSTMs
Gated Recurrent Units — GRUs

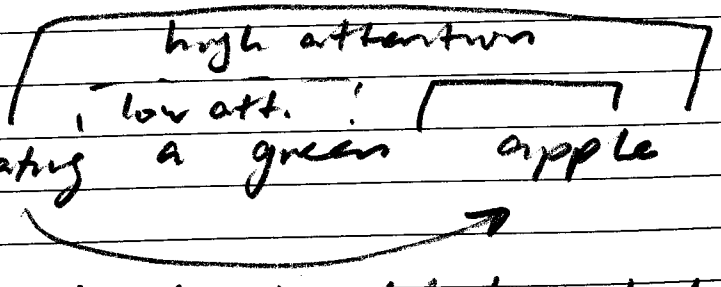
note. these terms are used interchangeably, but RNNs use LSTM or GRU under the hood

Gradient Clipping: put a maximum on the gradient value allowed to avoid the exploding gradient

Static Embeddings: RNN limitation

the embedding space does not change ... same words with different meanings have the same embedding!

Attention: Some part of our context are more important than others!



eating should update the embedding of apple apple that can be eaten vs. the company

introduces dynamic embeddings!

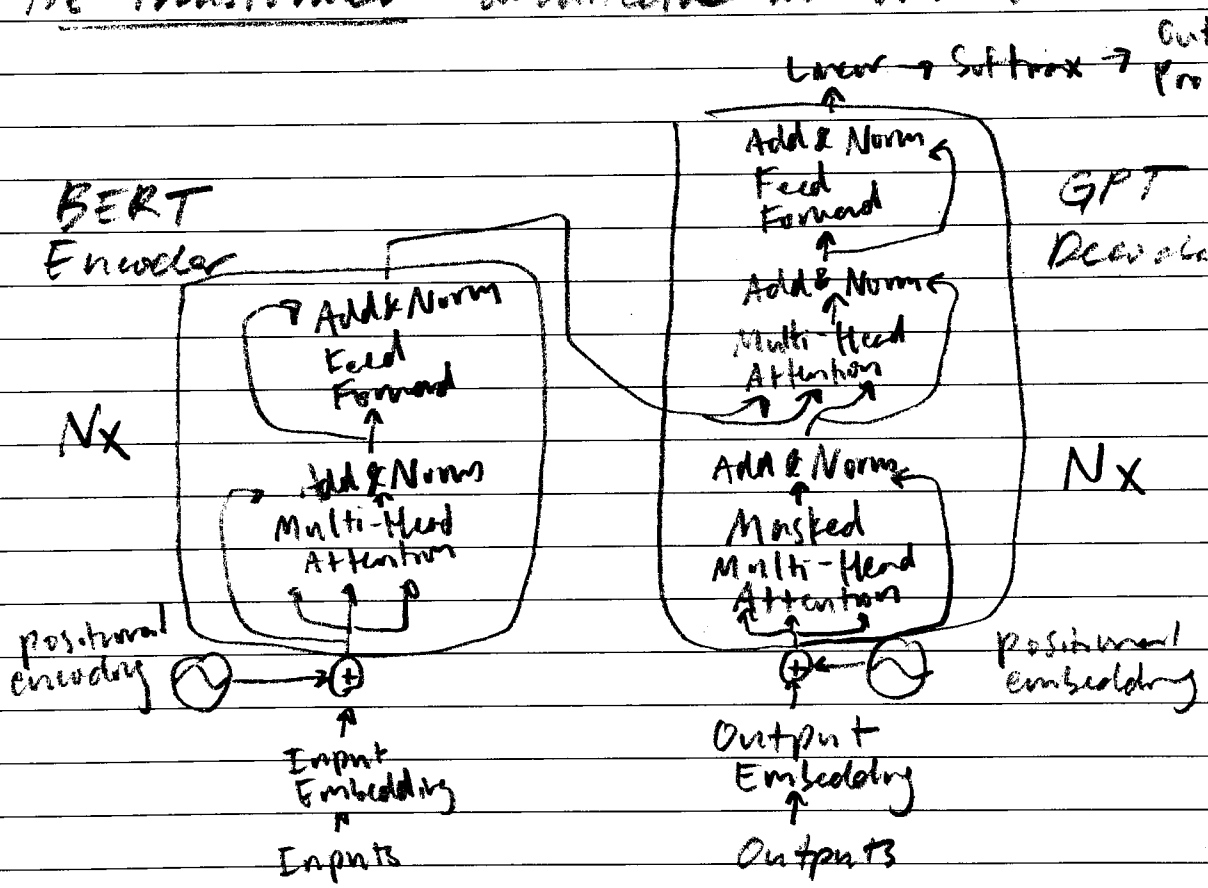
Date. 11/24/25
No. 25

Self-Attention

Attention: mechanism that allows tokens to adjust each other

She is eating a green apple

The Transformer: architecture in blocks



Attention is all you need!

We can calculate one-to-next attention in parallel, called **self-attention**

$$3\begin{bmatrix} \\ \\ \\ \end{bmatrix}_{512} \times \begin{bmatrix} | & | & | \\ | & | & | \\ | & | & | \end{bmatrix}_{3}^{512} = \begin{matrix} 0 \\ 1 \\ 2 \end{matrix}\begin{bmatrix} & & \end{bmatrix}$$

Q K^T 0 1 2

Date.
No.

Self-Attention: often attention is written as QKV which stems from the original paper. They model it as dictionary lookup

Query: what each input is looking for

Key: what each input contains

Value: How each input affects each output

Equations:

$$Attention(Q, K, V) = V \cdot softmax\left(\frac{QK^T}{\sqrt{d}}\right)$$

$$V = XW_V^T, \quad Q = XW_Q^T, \quad K = XW_K^T$$

w.r.t.,

$Q \cdot K$: Similarity of each query with each key

Softmax: creates attention distribution

Value: content we want to add

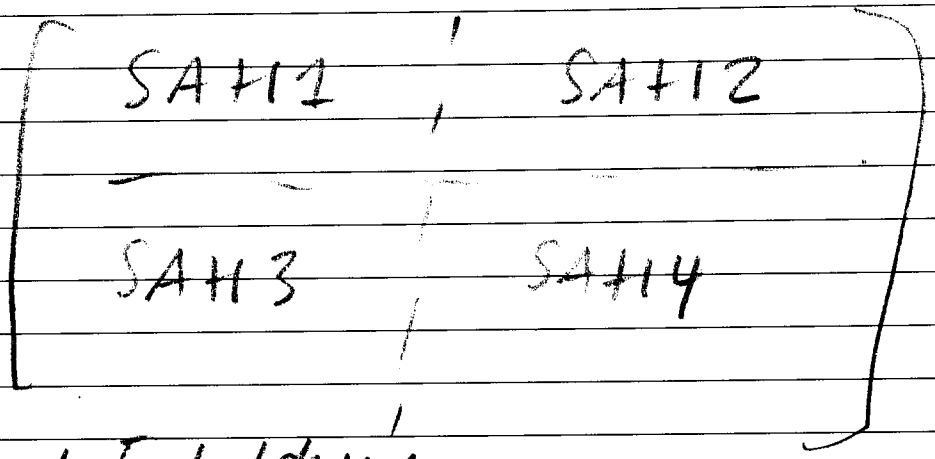
Multi-headed Self Attention:

we can use multiple attention heads that focus on different facets of textual context

we concatenate together all of these attention vectors and pass them through a single linear combination

Date.
No.

note. Multi-headed Self Attention done with one logically partitioned matrix



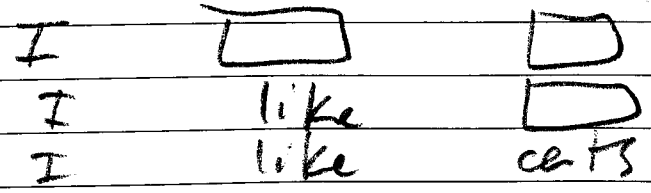
Positional Embeddings:

We need some method of including and maintaining the positional information of each token

To do this we pass each token's position into 128 sine functions each with a different frequency and add these to the input embeddings

Masking: during training, we need to make sure the transformer can't cheat.

We achieve this by masking (hiding future information)



Diffusion Models

Date. 12.1.25
No. 26

Diffusion Models are multimodal models that given a text prompt generate an image

These are a form of generative AI, like transformers

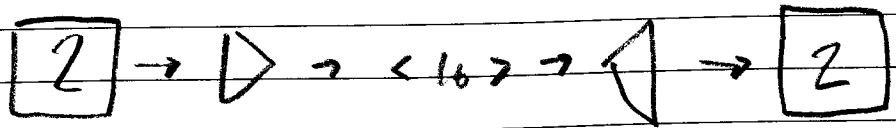
Generative Model is a type of ML model that aims to learn underlying distributions of the training data in order to generate new, similar data

Ex's.
- Markov chains
- Generative Adversarial Networks (GANs)
- Variational Auto-Encoders (VAEs)
- Diffusion Models

Generative Adversarial Networks:

GANs are a way to train a generative model by framing the problem as a supervised learning problem w/ two sub-models

1. Generator: Model we train to generate new ex's
2. Discriminator: Model that tries to classify ex's as real or fake

Auto Encoders: 

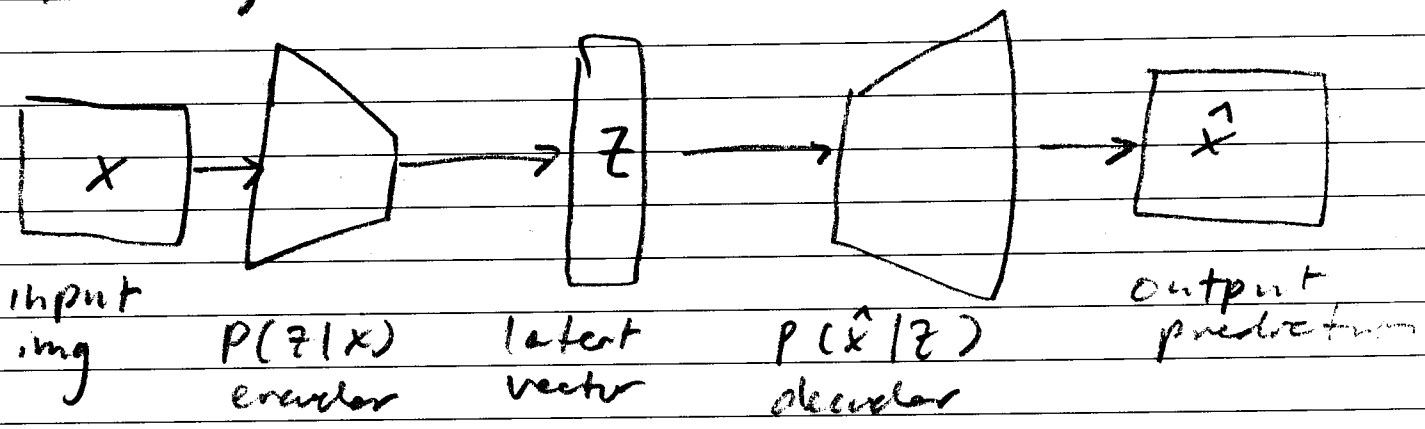
another 2-model system:

1. Encoder: maps each image to a latent vector
2. Decoder: maps each latent vector to an image

Date.
No.

Variational Auto-Encoders:

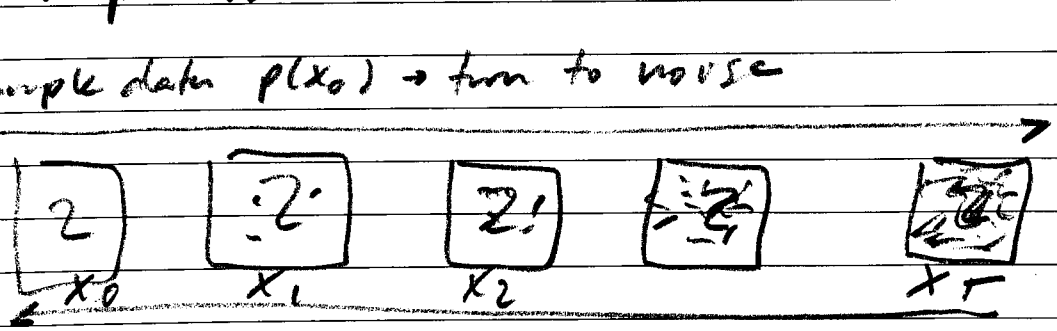
produces a noisy latent vector of the input, then the network randomly nudges it a bit, and the decoder has to reconstruct the original from the nudged vector



Diffusion Models:

trained by progressively adding Gaussian noise to a dataset and then learning to reverse this process

sample data $p(x_0)$ → turn to noise



$$p_T(x_T) \sim \mathcal{N}(0,1)$$

$$q(z_{t-1} \mid z_t) = \frac{q(z_t \mid z_{t-1})\, q(z_{t-1})}{q(z_t)}$$

↗ diffusion model architecture in slides ↗

Date.
No.

Image Noising (Forward Diffusion)

small gaussian noise is deterministically added to input image

$q(x_t | x_{t-1})$

$x_0 \rightarrow \cdots x_{t-1} \rightarrow x_t \rightarrow \cdots \rightarrow x_T$

[2] [2] [2]

Image Denoising (Backwards Diffusion)

model learns to reverse the deterministic Gaussian noise

$p_\theta(x_{t-1} | x_t)$

$x_T \rightarrow \cdots \rightarrow x_t \rightarrow x_{t-1} \rightarrow \cdots \rightarrow x_0$

[2] [2] [2]

$q(x_t | x_{t-1})$

New Model Architecture: U-Nets

comprised of 2 parts

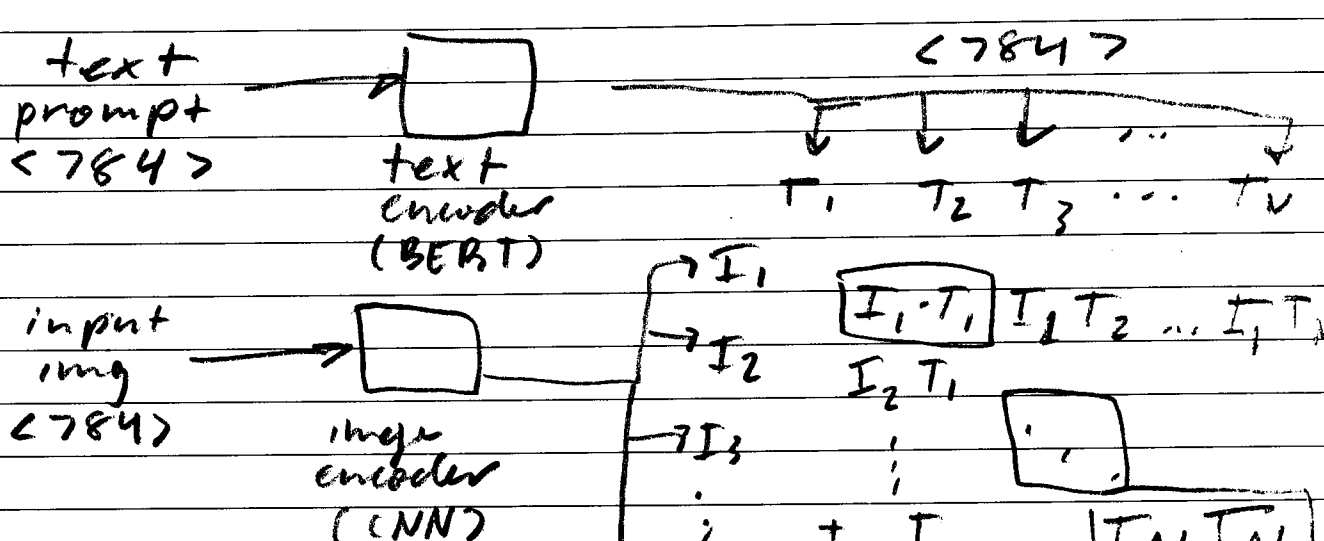
Contracting path: aims to reduce the size of image

Expanding path: aims to upsample the feature map

↳ architecture in slides

Date.
No.

Contrastive Language-Image Pretraining (CLIP):



use CLIP loss to compute similarity scores on the diagonal of the shared space

* this is in the training process so the model can learn images *

VAEs vs U-Nets in Diffusion Models:

1. VAE encoder encodes images into latent space
2. Diffusion process occurs in latent space where noise is iteratively added, and the UNet denoiser removes noise at each step
3. VAE decoder then reconstructs the image from the denoised latent vector

* we do this so the UNet only has to work in the smaller latent space *

Reinforcement Learning

Date. 12.3.25
No. 27

Reinforcement Learning: where an agent learns to make decisions by interacting with its environment to maximize its reward

Agent: The decision maker / learner

Environment: The external system in which the agent interacts

Reward (R): The feedback the agent gets after making a decision

Exploration vs Exploitation Problem:

The agent's trade off between:

Exploration: exploring / trying new actions to discover more information about the environment

Exploitation: repeating an action to yield high rewards

EvE Algorithm: Epsilon Greedy

w/ prob ε, the agent explores with prob $1-\varepsilon$, it exploits

Pros: very easy

Cons: very sensitive to ε, so agent might waste resources on bad actions

Date.
No.

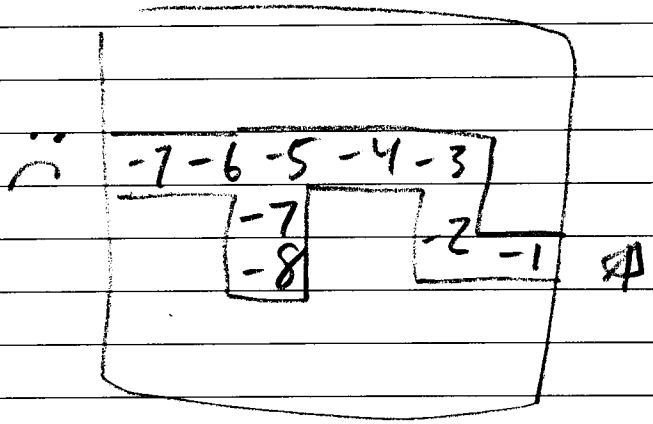
UCB Algorithm : Upper Confidence Bound 1

$$UCB\ 1 = v_i + 2\sqrt{\frac{\ln N}{n_i}}$$

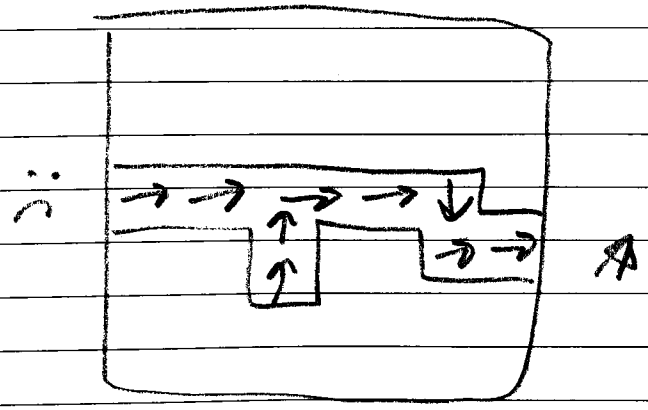
Pros: balances exploration and exploitation mathematically
Cons: expensive

Types of Reinforcement Learning :

Value Based: train the agent to learn which state is more valuable and take actions that lead to it



Policy Based: train the agent to learn which actions to take given a state



Date.
No.

Q-Learning: the value-based, model-free RL alg

The agent learns by updating values in a Q-table where each entry represents the value of taking a particular action in a given state

Note, it calculates the values via the Bellman Equation

The Bellman Equation:

$$V_\pi(s) = E_\pi\left[R_{t+1} + \gamma * V_\pi(s_{t+1}) \mid s_t = s \right]$$

(γ: hyperparameter)

- $V_\pi(s)$: value of states
- E_π: policy
- R_{t+1}: immediate reward
- $\gamma * V_\pi(s_{t+1})$: recursion

shows how the value of being in a certain state depends on the rewards of that state and the value of future states

Deep Q-Network (DQN)

instead of a Q-table, we train a network to predict the reward for each action, given the current state

Date.
No.

Monte-Carlo Tree Search (MCTS):

Search algorithm that approximates the optimal action to take from a given state by balancing EvE using random simulations

1. Selection: walk down tree using UCB1 to select the most promising node
2. Expansion: add new child nodes if tree hasn't been fully explored
3. Simulation: play a "random rollout" to determine an outcome
4. Backpropagation: update the node's WR, visit counts, and UCB1 scores for future selection

Policy in RL: defines the strategy the agent uses to determine its actions at each step. It does this w/o using value functions

The goal of RL is to find the optimal policy that maximizes the reward

Actor-Critic Methods: balance the strengths of policy-based (actor) and value-based (critic) methods, leading to more efficient and stable learning

LLMs and Agents

Date. 12/8/25
No. 28

Reinforcement Learning for LLMs: RLHF

Prompt → LLM → RL → output

↖ ↙

just patchwork on top of the base model

Group Relative Policy Optimization: GRPO

another way to do "fine-tuning" on an LLM → diagram in slides

Retrieval-Augmented Generation: RAG

Documents → Vector DB → Top-k → LLM → output
<784> ↑ chunks

user query
<784>

AI Agents: "AI Slop"

- ReAct: "reasoning and acting" → combines chain of thought (CoT) reasoning with external tool use (awk, sed, cat, ...)

Query → LLM (reason) → Tools </>

Action ↓

Response ← LLM (generate) ← Result ← Environment

Date.
No.

Model Context Protocol (MCP)

- move AI slop, return everything in JSON
- very typical right now
- standardized responses

from right page,

$$\frac{\partial L}{\partial w_i} = \frac{\partial y}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_4} \frac{\partial z_4}{\partial A_2} \frac{\partial A_2}{\partial z_2} \left(\frac{\partial z_2}{\partial w_i} + \frac{\partial z_2}{\partial A_1} \frac{\partial A_1}{\partial z_1} \frac{\partial z_1}{\partial w_i} \right)$$

$\frac{\partial z_2}{\partial w_i}$: x_2

$\frac{\partial z_2}{\partial A_1}$: w_c

$\frac{\partial A_1}{\partial z_1}$: $\frac{\partial \tanh}{\partial z_1}$

$\frac{\partial z_1}{\partial w_i}$: x_1

Date. 12.10.25
No. 29

Final Exam Review

R.N.N Computation Graph:

"Merry Christmas" (input)

< 0 1 > (tokenize)

[e(0) e(1)] (embed)

x_1 x_2

Vocab =
Merry - 0
Christmas - 1
Class - 2
Mon - 3
Tony - 4
Bui - 5

R.N.N: Forward Pass

$z_1 = x_1 W_i^T + A_0 W_c^T + b_h$

$A_1 = \tanh(z_1)$

$z_2 = x_2 W_i^T + A_1 W_c^T + b_h$

$A_2 = \tanh(z_2)$

$z_y = A_2 W_y^T + b_y$

$\hat{y} = \text{softmax}(z_y)$

$L = L_{CE}(\hat{y}, y)$

$\hat{y} = <0, 0, 0.1, 0.2, 0.2, 0.5>$

$Bui = \text{argmax}(\hat{y})$ ✗

$y = <0, 0, 1, 0, 0, 0>$

$L = L_{CE}(\hat{y}, y) \rightarrow$ 5 bad

Backward Pass:

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{z_y} \cdot \frac{\partial z_y}{\partial A_2} \frac{\partial A_2}{\partial z_2}$$

$$\frac{\partial z_2}{\partial W_i} = \frac{\partial z_2}{\partial W_i} + \frac{\partial z_2}{\partial A_1} \frac{\partial A_1}{\partial z_1} \frac{\partial z_1}{\partial W_i}$$

note. $z_2 = x_2 W_i^T + A_1 W_c^T + b_h$

$= x_2 W_i^T + \tanh(x_1 W_i^T + A_0 W_c^T + b_h) W_c^T + b_h$

Date.
No.

3-token Ex:

"Good luck everybody"
x_1 x_2 x_3

$z_1 = x_1 W_i^T + A_0 W_c^T + b_h$ note. $A_0 = [0]$

$A_1 = \tanh(z_1)$

$z_2 = x_2 W_i^T + A_1 W_c^T + b_h$

$A_2 = \tanh(z_2)$

$z_3 = x_3 W_i^T + A_2 W_c^T + b_h$

$A_3 = \tanh(z_3)$

$z_4 = A_3 W_4^T + b_4$

$\hat{y} = \text{softmax}(z_4)$

$L = J_{CE}(\hat{y}, y)$

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_4} \frac{\partial z_4}{\partial A_3} \frac{\partial A_3}{\partial z_3} \frac{\partial z_3}{\partial W_i}$$

$$\frac{\partial z_3}{\partial W_i} = \frac{\partial z_3}{\partial W_i} + \frac{\partial z_3}{\partial A_2} \frac{\partial A_2}{\partial z_2} \frac{\partial z_2}{\partial W_i}$$

$$\frac{\partial z_2}{\partial W_i} = \frac{\partial z_2}{\partial W_i} + \frac{\partial z_2}{\partial A_1} \frac{\partial A_1}{\partial z_1} \frac{\partial z_1}{\partial W_i}$$

Date.
No.