***PEAS*: Agent** AI chatbot perceives ***E*NVIRONMENT** (percept; chat interface, third-party plugins, user) through ***S*ENSORS** (text input, chat history, context) → agent function (maps from percept histories to actions) → acts through ***A*CTUATORS** (text output, image output, API). Based on available info (not nec complete), **Rational agent** choose actions that max ***P*ERFORMANCE MEASURE** (safety, speed, correctness): Best for whom, Maximise what, Unintended effects, Costs.

- **Fully (Partially) observable:** Sensors give agent access to complete state of environment relevant to choice of action.
- **Deterministic:** Next state of environment completely determined by current state and action executed by agent.
  - (vs **Stochastic: some randomness involved** eg players choose their move randomly in rock paper scissors)
  - **Strategic**: If envt also depend on actions of **other agents**, then is **also** strategic, unless other actions r predictable,eg dumb
- **Episodic:** Agent's experience divided into atomic episodes (perceive then act), choice of action depends on current episode itself (**Sequential**: Current decision can affect all future decisions, affected by previous sequence of actions).
- **Static (Dynamic):** Environment unchanged when agent is deliberating. (**Semi-dynamic**: Environment does not change with time, but agent's performance score does eg timer) .
- **Discrete (Continuous):** Limited no. of distinct, clearly defined percepts & actions.
- **Single (Multi) agent:** Any object(s) that maximise(s) performance measure (value depends on agent's behavior)

Agent is completely specified by **agent function** (environment → sensors → **agent function** → actuators → environment)

- **Simple reflex agent:** Acts based on current percept, and ignores percept history. Uses condition-action (if-then) rules.
- **Model-based:** Tracks how world changes (transition model) & info from percept; Simulates envt. Condition-action (if-then) rules
- **Goal-based:** Tracks state of the world and what it will be like if it does action A. Picks action that brings agent closer to goal.
- **Utility-based:** Also tracks state of world and what it will be like if it does action A. **Utility function** to assign score to any percept sequence (how happy it will be in such a state); internalise performance measure,check alignment.
- **Learning: Performance element** to select external actions; Critic to give feedback on performance; Learning element makes improvement; Problem generator suggests actions

---

***SEARCH PROB (predictable/no opponent → Uninformed / Informed Search)*:** States. Initial State. Goal State(s)/Test. Actions. Transition Model. Action cost function.

— — — **SEARCH ALGORITHMS** — — —

— (path matters for (un)informed, adversarial search) —

Goal-based, fully observable, deterministic, static, discrete.

⇒ Solution: A fixed sequence of actions → form a path to a goal

Order of node expansion - which states are prioritised

Tree search: Frontier (data structure) defines the search; where it is rare for two paths to reach the same state

Graph search: Remember all visited nodes in a hashmap

```
create frontier  create visited // expands less states
insert initial state to frontier and visited
while frontier is not empty:
   state = frontier.pop()              // may skip states
   for action in actions(state):       with less cost
      next state = transition(state, action)
      if next state in visited: continue
      if next state is goal: return solution
      frontier.add(next state)
      visited.add(state)
return failure
```

***UNINFORMED* SEARCH (BFS,uniform-cost,DFS,DLS,IDS)**

Blind search, no clue how close a state is to the goal(s).

Breadth-First Search (queue (layer by layer)**, FIFO)** Special case of uniform-cost search (equal step costs). f(n) = depth of node. Save space for BFS while preserving completeness: Apply goal-test when PUSHING a successor state to the frontier; *check if next state is goal BEFORE pushing to frontier queue*.

```
create frontier : queue          create visited
insert initial state to frontier
while frontier is not empty:
 state = frontier.pop()                  // goal test on the
 if state is goal: return solution       node popped
 if state in visited: continue   visited.add(state)
 for action in actions(state):
     next state = transition(state, action)
     // if next state is goal: return solution
     frontier.add(next state)
return failure
```

Uniform-Cost Search (PRIORITY queue (path cost))

- **A*** with h(n)=0. Expand the least cost unexpanded node.
- **Best-First Search** with f(n) = path cost (**FROM INITIAL state**)

Depth-First Search (stack, LIFO) Can optimise w backtracking search, where only 1 successor generated at a time, O(m) space

Depth-Limited Search (Dls) DFS with depth limit ℓ.

Iterative Deepening Search (IDS) DLS from 0, 1 … ∞

- IDS NOT always faster than DFS for time complexity. If each state has only a single successor and goal node at depth n, then IDS will run in O(n²) while DFS will run in O(n).

***INFORMED* SEARCH (Best-first Search)** domain info to *guide*

Best-First Search (priority queue f(n)): Eval function **f(n)** for how good a state is. Expand most desirable unexpanded node.

- **GREEDY Best-First Search** (**A*** with g(n) = 0) **f(n) = h(n)** Expand node with lowest h(n), appears closest (low cost) to goal
- **A* Search f(n) = g(n) + h(n)** g(n) = cost so far to n. h(n)= est cost from n to goal. f(n)= estimated total cost of path.

Admissible Heuristics h(n) ≤ h*(n) h*(n) = true cost

- h(n) never overestimates cost to reach a goal; conservative.
- **If h(n) is admissible ⇒ A* using TREE search is optimal.**

Dominance If $h_2(n) \geq h_1(n)$ **for all n**, then h_2 dominates h_1 and is closer to the true cost h*(n), incurs less search cost (on average). h_2 is better for search if admissible.

Consistent Heuristics (is admissible)(satisfies △ inequality)

- Successor n' of n generated by action a: **h(n) ≤ c(n, a, n') + h(n') and h(G)=0**. ⇒ f(n) is non-decreasing along any path.
- **If h(n) is consistent ⇒ A* using graph search is optimal.**
- **Consistent heuristic is admissible, but converse not true.**

** NO admissibility and consistency in local, adversarial search.*

Search	Informed,Uninformed	Local
Agent	Goal-based	Goal &/or Utility-
State space	Low to moderate	Very large. Games.
Time constraint	A solution / No solution	Good enough solutn
Solution	Search path (usually)	State

LOCAL SEARCH States, Initial state, Goal test (optional)

- **Successor function:** Possible states from a state
- Objective functions: Evaluate value / goodness of a state

State space landscape: Local max, Global max, Shoulder (same values, but progress possible). *Can still solve shortest path prob (state=random path; successor=add/remove subpaths*

Hill-Climbing / Steepest Ascent / Greedy Local Search

Find local maxima by traveling to **neighbouring** states with the **highest** value. If node has higher value than successors, then return it. It is a local maximum, but may not be a global max.

Simulated Annealing (*hill climbing, but allow bad moves*)

Randomly pick successor/action. If the successor has higher value, recurse. Else recurse with probability (=curr, next, T) of choosing a bad move that exponentially decreases. *Escapes local maxima by allowing bad moves occasionally*. If T decrease slowly enough, then finds a global optimum with high probability.

ADVERSARIAL SEARCH AND *GAMES (unpredictable)*

Fully observable, deterministic, discrete, terminal states exist (~~infinite run~~), two-player zero-sum (~~win-win~~), turn-taking.

- **States, Initial state, Terminal states** (where game ends)
- **Actions:** *Action(s)* gives a set of legal moves in *s*.
- **Transition**, **Utility function**(output the value of a state)

Minimax (recursive like DFS: choose highest **minimax** value)

- *Assume the MIN player plays optimally, trying to minimise player's value. Not optimal against a non-optimal MIN player.*
- Pick a move that **maximises** player's **utility**

Alpha-beta Pruning (prune useless branches)

- Will not change the decision

```
def alpha-beta(state):      [minimax: not highlighted]
  v = max_value(state, -∞, ∞)
  return action in successors(state) with value v
def max_value(state, α , β):   MAX player choose MAX
  if is_cutoff(state): return eval(state)
  if is_terminal(state): return utility(state)
  v = -∞
  for action, next_state in successors(state):
    v = max(v, min_value(next_state))
    α = max(α, v)              // α = highest value for MAX
    if v >= β : return v   // stop if v >= β
  return v
def min_value(state, α , β):   β = best choice for MIN
  if is_cutoff(state): return eval(state)
  if is_terminal(state): return utility(state)
  v = ∞
  for action, next_state in successors(state):
    v = min(v, max_value(next_state))
    β = min(β, v)              // β = lowest value for MAX
    if v <= α : return v   // stop if v <= α
  return v
```

Minimax with cutoff at a certain depth (refer: ~~strikethrough~~)

- For large/infinite game trees: Cutoff, Use evaluation function
- Replace is_terminal / utility with is_cutoff / eval
- is_cutoff return true if state is terminal, exceed time/depth limit
- eval return utility for terminal, heuristic value for non-terminal
- Evaluation (heuristic) function: Estimate how good a state is.

MACHINE LEARNING: SUPERVISED LEARNING Observe input-output pairs, learn fn to map input to output. **Learn from being given right answers X → Y**. Each eg has correct answer.

- Assume y is generated by a true mapping function *f*: $x \rightarrow y$. Use a learning algorithm to find a hypothesis *h*: $x \rightarrow \hat{y}$ (from Hypothesis class H) s.t. $h \approx f$, given a training set (set may contain errors) *{ $(x_1, f(x1)$ … $(x_N, f(x_N))$ }*.

Performance: True measure of hypothesis h (Good if h ≈ f) is how well it handles inputs it has not seen yet, ie the **test set**.

WEAKLY / SEMI-SUPERVISED LEARNING Correct answer given, but not precise, eg image has face, but dk exact location.

UNSUPERVISED LEARNING Agent learns patterns without labels/explicit feedback. **No correct answers.** Eg **Clustering**

REINFORCEMENT LEARNING Agent learns from reinforcements (rewards, punishments), eg game, maze.

REGRESSION (output is **continuous** value) → Measure **error**:

Eg Predict housing prices

Predicted $\hat{y}_i = h(x_i)$; true $y_i = f(x_i)$ for N samples *{$(x_1, f(x1)$…)}*

- **(Mean) Squared Error** **(Mean) Absolute Error**

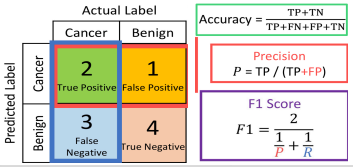
$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y)^2 \qquad \text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y} - y|^2$$

CLASSIFICATION → Measure **Correctness** and **Accuracy**:

$$\textbf{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} 1_{\hat{y}_i = y_i} \qquad \text{where } \hat{y}_i = h(x_i) \;;\; y_i = f(x_i)$$

Range is 0 to 1. 1 if **prediction** $\hat{y}_i = y_i$ **(true label)**. Take mean.

CLASSIFICATION (discrete output) → CONFUSION MATRIX:



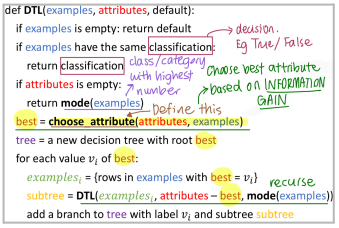
Recall TP / (TP+FN)	**Precision** TP / (TP+FP)
How many **relevant** items are **selected**? How many positive instances can be **recalled** (predicted)?	How many **selected** items are **relevant**? How **precise** were the positive predicted instances?
Maximise if FN, type II is very dangerous, eg Cancer (not music) prediction	Maximise this if FP, type I error) is very costly. eg Email spam, Satellite launch date prediction.

DECISION TREE FALSE that can always find ≥1 DT to perfectly label every eg in training set, no matter amount of data, attributes. - Data may not be consistent. If we train a DT that determines if a student passes/fails. Can't cleanly split using the training set gender as only attribute. **Attributes** Binary, Discrete, Binned/Discretised continuous valued (<10, 10-20, 20-40…)

Expressiveness DT can express any function of input attribute

- Function maps a vector of attribute values to a single output value (a decision eg T/F). • Internal nodes are tests, leaf specifies value to be returned. • *Trivially*, there is a consistent decision tree for ANY TRAINING SET, but probably **will not *generalise (prefer a more compact tree)* to new examples**.
- Size of Hypothesis class: For Bool functions with *n* Bool attributes, #distinct decision trees = #Bool functions = #distinct truth tables with 2^n rows (each row outputs T/F) ⇒ 2^{2^n}

GREEDY, TOP-DOWN, RECURSIVE algorithm for DTL



1 If remaining e.g.s are all True or False, return that value.
2 If a mix, then choose best attribute to split and recurse.
3 No egs left, return most common value frm node's parent's egs
4 No attributes left to split, return most common value of curr egs

Decision Tree Learning with INFORMATION GAIN

Entropy (randomness): $I(P(v_1)\ldots P(v_n)) = -\sum_{i=1}^{n} P(v_i)\log_2 P(v_i)$

- Entropy = 1 if 50:50 eg #true = #false q = prob of true
- Bool variables: $B(q) = -(q\log_2 q + (1-q)\log_2(1-q))$
- Dataset with p positive and n negative examples:

$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$ $\quad plog\ p\ ,\ nlog\ n$

Entropy of output of entire set is $B(\frac{p}{p+n})$.

A **chosen attribute** A **with** v **distinct values** divides training set E into subsets $E_1 \ldots E_v$ according to their values.

$$remainder(A) = \sum_{k=1}^{v} \frac{p_k+n_k}{p+n} I(\frac{p_k}{p_k+n_k}, \frac{n_k}{p_k+n_k})$$

Each subset E_k has p_k positive examples and n_k negative examples: $Remainder(A) = \sum_{k=1}^{v} \frac{p_k+n_k}{p_n} B(\frac{p_k}{p_k+n_k})$

Information Gain (highest = impt attrib) / Reduction in Entropy:

$IG(A)\ (in\ bits) = I(\frac{p}{p+n}, \frac{n}{p+n}) - remainder(A)$

= Entropy of this node - Entropy of children nodes

- **Continuous-valued Attributes:** Define a discrete-valued input attribute to partition values into a discrete set of intervals.
- **Missing values:** Assign most common val of A at node n to eg with missing data for A & at node n. Or filter by most common val amongst egs w same output. Or assign each possible value of A some prob, split egs with missing data. Or drop attribute/row.

b: branching factor | d: depth | m: maximum depth | C: path cost of optimal solution | ∈: minimum edge cost | G: goal state*

SEARCH	**TIME:** #nodes generated or expanded	**SPACE:** maximum #nodes in memory	**COMPLETE:** Solution / failure if there is	**OPTIMAL:** Always find least cost
BFS	1 + b + b² + … + b^d = **O(b^d)**.	**O(b^d)**. Worst case: expand last child in a branch.	Complete if b is finite.	Yes if same step cost everywhere.
Uniform-cost	**O($b^{C^*/\epsilon}$)**. Estimated depth = C*/∈	**O($b^{C^*/\epsilon}$)**.	Yes if ∈ > 0 and finite C*.	Yes if ∈>0. ∈=0 may cause zero cost cycle
DFS	**O(b^m)**.	**O(bm)**.	No, when infinite depth / back&forth loop.	No.
DLS	1 + b + b² … b^ℓ = **O(b^ℓ)**. ℓ = depth limit.	**O(bℓ)** if used with DFS.	No if ℓ is not the depth of optimal solution.	No if used with DFS.
IDS overhead = $\frac{nIDS - nDLS}{nDLS}$	$b^0 + (b^0 + b^1) \ldots + \ldots (b^0 \ldots + b^d) =$ (d+1)b^0 + (d)b^1 + … (1)b^d = **O(b^d)**.	**O(bd)** if used with DFS.	Yes.	Yes, if same step cost everywhere.Is
Greedy Best-First	**O(b^m)**. Good heuristic h(n) gives improve.	**O(b^m)**, keep all nodes in memory.	No, can go into loops.	No, heuristic function can be wrong.
A*	**O(b^m)**. Good heuristic h(n) gives improve.	**O(b^m)**, keep all nodes in memory.	Yes, tracks path so no back and forth. Unless infinite nodes with f ≤ f(G).	Depends on heuristics.
Minimax	**O(b^m)**.	**O(bm)**. **Depth first** exploration (then backtrack)	Yes, if the tree is finite.	Yes, against optimal opponent.
α-β Pruning	**O($b^{m/2}$)** with perfect ordering.			

• **Attribute with many values** (eg dates, phone numbers): Selected by IG as it splits the data well. In extreme cases, each branch has 1 eg, all +/-ve. So balance $Information\ Gain$ with #branches. Use $Gain\ Ratio = \frac{IG(A)}{SplitInformation(A)}$.

$Split\ Information(A) = -\sum_{i=1}^{d} \frac{|E_i|}{|E|} log_2 \frac{|E_i|}{|E|}$, where E = example.

• **Attributes with Differing Costs** (eg biopsy cost): Make decision tree use low-cost attributes, use Cost-Normalised-Gain $\frac{IG^2(A)}{Cost(A)}$ or $\frac{2^{IG(A)}-1}{(Cost(A)+1)^w}$, where $w \in [0,1]$ determines the relative importance of the cost vs information gain.

• **Continuous-valued output**: Need regression tree.

Overfitting: Decision Tree's performance is perfect on training data, but worse on training data. Decision tree captures data perfectly, **including noise**. *Dealt with using Occam's Razor:*
• Prefer short / simple hypotheses. • More susceptible to overfitting if too many attributes used • In favor: Short/simple (or Long/complex) hypothesis that fits the data is unlikely to (or may) be coincidence. • Against: Many ways to define small sets of hypotheses. Can use diff hypotheses representations.

PRUNING (eg minimum sample leaf pruning, max depth)
• Prevent nodes from being split if fail to cleanly separate e.g.s
• Results in **smaller tree**. DT has **higher accuracy** if tested using sample data as **noise can be ignored** through pruning.

LINEAR REGRESSION (fitting a line to data) Hypothesis $h_w(x) = w_0 + w_1 x$. Note m = # samples. **Loss function** $J_{MSE}(w) = \frac{1}{2m}\sum_{i=1}^{m}(w_1 x^{(i)} - y^{(i)})^2$ • Use **MSE** as differentiable everywhere. • **MAE** not differentiable everywhere. Emphasises error less, maybe better if relatively many outliers in data set. To improve loss function, use gradient. Take partial derivative:

• $-\frac{\partial J_{MSE}(w)}{\partial w_1} = -\frac{1}{m}\sum_{i=1}^{m}(w_1 x^{(i)} - y^{(i)})\ x^{(i)}$

• If $\frac{\partial J_{MSE}(w)}{\partial w_1}$ and $\frac{\partial J_{MSE}(w)}{\partial w_0}$ = 0, the loss function is minimised.

GRADIENT DESCENT (reverse hill climbing) Can't always find partial derivatives = 0. So estimate of gradient at each pt, move a small amt in steepest downhill direction, **till converge on point w min loss** • Start at some w. Pick a nearby w that reduces $J(w)$. Update weights using $w_j \leftarrow w_j - \alpha \frac{\partial J(w_0, w_1 \dots)}{\partial w_j}$, α = learning rate • α often decrease w more iterations. Start big to reach optimal values faster. Slow down for gradual converge.

LINEAR REGRESSION with GRADIENT DESCENT
MSE loss function is **convex** for **linear** regression. Hence **one minimum**, which is a **global minimum**.

$$w_0 \leftarrow w_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})$$

$$w_1 \leftarrow w_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})x^{(i)}$$

Gradient Descent • **Batch:** Consider all training examples. • **Mini-batch:** Consider a subset of egs each time. Cheaper (faster) / iteration. Randomness, may escape local minima. • **Stochastic:** Use ONE RANDOM data each time. Cheapest, (fastest) /iteration. More randomness, may escape local minima

LINEAR & LOGISTIC REGRESSION: Challenges, Solutions (Multivariable) Linear/Logistic Regression w Many Attribute

• Hypothesis for n features: $h_w(x) = \sum_{j=0}^{m} w_j x_j = w^T x$

• $w_j \leftarrow w_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})x_j^{(i)}$

where $x_j^{(i)}$ = value of feature j in i-th training example

Features of Different Scales → Mean Normalisation
• eg Each feature(row) has different number in leftmost column
• Standardisation: $x_j \leftarrow \frac{x_j - \mu_j}{\sigma_j}$, where $\frac{\mu_j = mean}{\sigma_j = standard\ deviation}$

Non-Linear Rship/Decision boundary → Transform features
• $x_1 \leftarrow x$, $f_2 \leftarrow x^2$, $f_3 \leftarrow x^3$ (expand feature / hypothesis class)

• **Polynomial regression**: $h_w(x) = w_0 + w_1 x + w_0 x^2 \dots$, where maximum degree is $n-1$; higher degree would be overfitting.

• $h_w(x) = \sigma(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2)$ ** $f_3 = x_1^2$

NORMAL EQUATION (*analytically* find best parameters)

BIAS, dummy feature related to bias term. Data matrix $X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & & x_n^{(2)} \\ 1 & \vdots & & \vdots \\ 1 & x_1^{(m)} & & x_n^{(m)} \end{bmatrix}$ Weights of params $w = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_n \end{bmatrix}$ Targets, eg house price $Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{bmatrix}$

$h_w(x) = Xw$

Find $w = (X^T X)^{-1} X^T Y$ that minimises J_{MSE}. What if $X^T X$ not invertible: Infinite solutions to normal equation; reasons:
• $m < n$, #training data < #features. More data required.
• Not all columns linearly independent. Can remove a feature that might be dependent on a linear combination of others.

	Gradient Descent	Normal Equation
Need to choose α	Yes	No
Iteration(s)	Many	None
Large #features n	No problem	Slow $(X^T X)^{-1} \rightarrow O(n^3)$
Feature scaling?	May be necessary	Not necessary
Constraints	-	$X^T X$ must be invertible

LOGISTIC REGRESSION Classification (continuous) has step/ threshold function → Discontinuous, not differentiable → Can't use linear regression. Logistic function (sigmoid), $\sigma(z) = \frac{1}{1+e^{-z}}$

• Hypothesis $h_w(x) = \sigma(w_0 + w_1 x_1 + w_1 x_2)$ outputs probability

• $J_{MSE}(w) = \frac{1}{2m}\sum_{i=1}^{m}(\frac{1}{1+e^{-(w_0+w_1 x_1+w_1 x_2)}} - y^{(i)})^2$

• $e^{-(w_0+w_1 x_1+w_1 x_2)}$ is Non-linear → **Non-convex** → Many local minima, **may not find global minimum**

Logistic Regression with CROSS-ENTROPY LOSS
If true value y and prediction $\hat{y}$ are close, then the Binary Cross-Entropy (closeness btwn *probability* distribution) is small → Good loss function. **BCE is convex for logistic regression**.

$BCE(y, \hat{y}) = -y log(\hat{y}) - (1-y) log(1-\hat{y})$

P(malignant)=0.8 P(not malignant)=0.2

For a set of m examples $\{(x^{(i)}, y^{(i)}), \dots, (x^{(m)}, y^{(m)})\}$,

$BCE\ loss,\ J_{BCE}(w) = \frac{1}{m}\sum_{i=1}^{m} BCE(y^{(i)}, h_w(x^{(i)}))$

Logistic Regression with Gradient Descent (same as linear)

$w_j \leftarrow w_j - \alpha \frac{\partial J_{BCE}(w_0, w_1 \dots)}{\partial w_j}$ $\frac{\partial J_{BCE}(w)}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})$

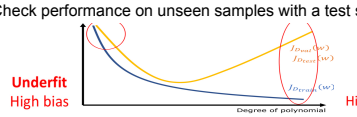
$\frac{\partial J_{BCE}(w)}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})x_1^{(i)}$

Multi-class Classification Train n diff logistic classifiers for all n classes. For each input, pick class i w max output from hypo

(More) Performance Measure: **Receiver Operator Characteristic (ROC) Curve** Model is more accurate than random chance if its ROC curve is above diagonal random line. **Area under Curve of ROC**: AUC is a concise metric (clearer comparisons) instead of a full figure. AUC > 0.5 (⇒model is better than chance) vs AUC ≈ 1 (⇒mode is very accurate).

Model Evaluation & Selection: Bias (model capacity) & Variance (model randomness) Train model h_i on training set. Pick model w lowest $J_{D_{val}}(w)$ (least error on validation set).

Check performance on unseen samples with a test set.



$J_{D_{val}}(w) \approx (high)\ J_{D_{train}}(w)$ $J_{D_{val}}(w) >> (low) J_{D_{train}}(w)$

Hyperparameter Tuning -Find best model. Pick hyperparams (polynomial deg, learning rate). Train model with hyperparams.
• Grid search(exhaustive search): Try all possible hyperparams
• Random search: Randomly select hyperparam • Successive halving: All possible hyperparam but with reduced resource. Successively increase resource w smaller set of hyperparam
• Bayesian method,estimate optimization space of hyperparam.
Evolutionary algorithms: To select population of hyperparams

REGULARISATION (else reduce #features) to fix **overfitting**. (perform better for unseen data. But can misclassify training data; high bias, low variance)

Linear Regression: Hypothesis $h_w(x) := w^T x$

$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{i=1}^{m}(w_j)^2$ similar for logistic

Grad descent: $w_n \leftarrow w_n - \gamma\frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})x_j^{(i)} - \gamma\frac{\lambda}{2m}w_n$

$1 - \frac{\gamma\lambda}{m}$ *slightly < 1* $= (1 - \frac{\gamma\lambda}{m})w_n - \gamma\frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})x_j^{(i)}$

Regularization parameter λ usually small (too large λ→ underfit) → Shrinks weight w_n

$w = (X^T X + \lambda I_m)^{-1} X^T Y$ dim of X: m(n+1). I_m:(n+1)(n+1). Y:m

Works even if $X^T X$ not invertible if λ>0

SUPPORT VECTOR MACHINES (SVM) : Maximise margin (dist btwn decision boundary & closest data pts of classes)
Assume linearly separable data. **Hard margin**:
Decision rule: $w \cdot x + b \geq 0$ then + | Maximise margin = $\frac{2}{\|w\|}$
Constrained optimisation problem (using lagrange method):
Primal optimise w, b: $min_w \frac{1}{2}\|w\|^2\ s.t.\ y^{(i)}(w \cdot x_{(i)} + b) - 1 \geq 0$

→ **Dual** optimise α: $max_{\alpha \geq 0} \sum_i a^{(i)} - \frac{1}{2}\sum_i\sum_j a^{(i)}a^{(j)}y^{(i)}y^{(j)}x^{(i)} \cdot x^{(j)}$

Assume not-linearly separable data. **Soft margin**: Allow egs to fall on wrong side, assign penalty based on distance $J(w) = C\sum_{i=1}^{m}\frac{(1+y^{(i)})}{2}cost_1(w^T x^{(i)}) + \frac{(1-y^{(i)})}{2}cost_{-1}(w^T x^{(i)}) + \frac{1}{2}\sum_{j=1}^{n} w_j^2$ $cost_1(z) = \max\{0, 1-z\}$ $cost_{-1}(z) = \max\{0, 1+z\}$

With non-linear decision boundary, **SVM with KERNEL TRICK**

Gaussian Kernel $K(u,v) := e^{-\frac{\|u-v\|^2}{2\sigma^2}}$ $\phi(u) \cdot \phi(v) = (u \cdot v)^d$

$\phi(x)$:transformed(by ϕ), maps to **infinite-dimensional** features $\Sigma a^{(i)}y^{(i)}\phi(x^{(i)}) \cdot \phi(x) + b = \Sigma a^{(i)}y^{(i)}K(x^{(i)}, x) + b \geq 0$ then +

Objective: $L(a) = \sum_i a^{(i)} - \frac{1}{2}\sum_i\sum_j a^{(i)}a^{(j)}y^{(i)}y^{(j)}K(x^{(i)}, x^{(j)})$

Data (pre)processing : *Feature Scaling (speeds up gradient descent).* **Feature Mapping (handcrafted for linear/logistic regression, kernel trick. Learned for neural networks).**

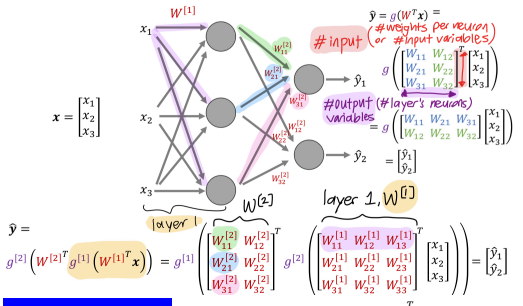
CNN - Convolution Dot product $l \times l$ submatrix in image with flipped $l \times l$ kernel/filter, get feature map. Padding: not lose edge pixel. • Input * Kernel → Output. **#input channels = #channels in each kernel** (each channel is 3D, concat kernels to get 4D tensor). **#kernels = #output channels**

Pooling Downsample feature maps (max/average/sum-pool). Reduces #dimensions. Reduces #weights & costs, learn faster.

Perceptron Learning Algorithm Initialise weights w (all 0 or random small values). For each instance, classify $\hat{y}^{(i)}$ with features $x^{(i)}$. Get **ONE *misclassified*** instance. Update weights $w \leftarrow w + \eta(y^{(i)} - \hat{y}^{(i)})x^{(i)}$ for η = learning rate. $\hat{y}$ = predicted (+1 if calculated is ≥0). y=true(label).

Activation function $\hat{y} = h_w(x) = g(\sum_{i=0}^{n} w_i x_i)$, $x_0 = 0$. Bias term adjusts threshold of activation for each neuron. **Sign** $g(z) = +1$ if $z \geq 0$, else -1. **Sigmoid** $\sigma(x) = \frac{1}{1+e^{-x}}$. **ReLu** $max(0, x)$

XNOR is not linearly separable. Can't use single layer neural network (AND, OR, NOR). Use **multi-layer neural network**:



$\hat{y} = g^{[2]}\left(W^{[2]^T} g^{[1]}\left(W^{[1]^T} x\right)\right) = g^{[1]}\left(\begin{bmatrix} W_{11}^{[2]} & W_{12}^{[2]} \\ W_{21}^{[2]} & W_{22}^{[2]} \\ W_{31}^{[2]} & W_{32}^{[2]} \end{bmatrix}^T g^{[2]}\left(\begin{bmatrix} W_{11}^{[1]} & W_{12}^{[1]} & W_{13}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} & W_{23}^{[1]} \\ W_{31}^{[1]} & W_{32}^{[1]} & W_{33}^{[1]} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right)\right) = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$

Forward propagation $a^{[L]} \equiv g^{[L]}(f^{[L]})$ and $f^{[L]} \equiv (W^{[L]})^T a^{[L-1]}$
$\Rightarrow \hat{y}(x) = g^{[L]}(f^{[L]}(g^{[L-1]}(\dots(g^{[1]}(f^{[1]}(x)))\dots)))$

Gradient descent Find $\frac{\delta L}{\delta w}$, L is loss function, eg $\frac{1}{2}(\hat{y} - y)^2$

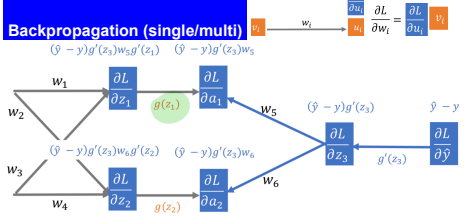
$\frac{d\hat{y}}{da} = \frac{d(\sigma(a))}{da} = \sigma'(a) = \sigma(a)(1 - \sigma(a)) = \hat{y}(1 - \hat{y})$ (as $\hat{y} = \sigma(a)$)

$\frac{\delta L}{\delta w} = \frac{\delta L}{\delta \hat{y}}\frac{\delta \hat{y}}{\delta a}\frac{\delta a}{\delta w} = (\hat{y} - y)\hat{y}(1 - \hat{y})x_i$ as $a = \Sigma w_i x_i$

$w \leftarrow w + \eta(\hat{y} - y)\hat{y}(1 - \hat{y})x$ **Perceptron: Learned feat mapg.**

Perceptron (vs Logistic Regression): Non-linear, non-robust decision boundary. Misclassification;bondary too close to points **(vs SVM):** SVM can maximise margin (→robust) and converge. Non-linear, robust decision boundary that is far from data points

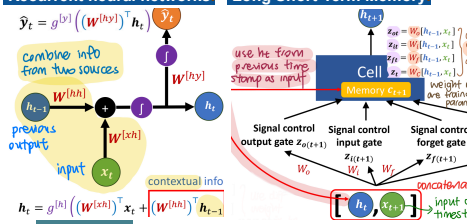
Backpropagation (single/multi)



Overfitting (can on generalise training, but not test data) → Regularise with • **Dropout**; **randomly** set some activations = 0
• **Early stopping** to prevent higher validation loss

Vanishing/Exploding gradient (small/large gradient multiplied again until almost 0/overflows) → Use non-saturating activation functions(ReLU, grad=1). Gradient clipping in range [min, max]

Recurrent neural networks **Long Short-Term Memory**



Use **self-attention** to capture context w/out waiting for prev time ⇒ **TRANSFORMER** - Input / output a sequence. Encoder-Decoder attention. Query based on prev decoder block's output Key & Value based on encoder's output. Decoder can use context info given by encoder.

Unsupervised Learning: Clustering **K-Means**

Cluster centroid, eg $\mu_1 = \frac{1}{m_1}\Sigma x^{(i)}$ **for** $x^{(i)}$ **points in cluster 1.**

Randomly initialise K centroids $\mu_1 \dots \mu_K$. Repeat till convergence:
• For i=1..m: $c^{(i)} \leftarrow$ index of cluster centroid ($\mu_1 \dots \mu_K$) closest to $x^{(i)}$
• For k=1..K: $\mu_K \leftarrow$ centroid of data pts $x^{(i)}$ assigned to cluster K
Each step never increases loss / distortion = ave dist of sample to its centroid = $J(c^{(1)} \dots c^{(m)}, \mu_1 \dots \mu_K) = \frac{1}{m}\Sigma \|x^{(i)} - \mu_{c(i)}\|^2$

Selecting K: Elbow method, plot loss against K. Pick K where after that point, loss decreases linearly.
K-Medoids: Use data points closest to centroids, as centroids.

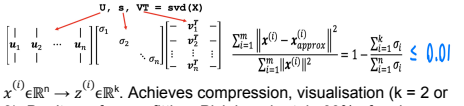
Hierarchical clustering Every point is a cluster. Merge 2 nearest clusters till all points in 1 cluster. Dendrogram shows merge against distance. Calculate dist w max, min, grp ave, centroid dist, Euclidean, Manhattan dist. High space & time complexity, impractical for large data. Apply in recommender, customer segment, medical imaging, analyse social network.

Dimensionality reduction (as #samples to learn h increases much w #features) Find new basis, capture most data variation

Principal Component Analysis $AX = X\Lambda$; X = eigenvectors $v_1 \dots v_n$; Λ = diagonal matrix of eigenvalues $\lambda_1 \dots \lambda_n$. ⇒ $A = X\Lambda X^{-1}$. By **singular value decomposition**, $X = U\Sigma V^T$; Σ = diagonal matrix of singular values of X ; U, V = left & right singular vectors

1. Covariance = $\frac{1}{m}\Sigma(\hat{x}^{(i)})(\hat{x}^{(i)})^T$ where $\hat{x}^{(i)} = x^{(i)} - \bar{x}$
2. Apply SVD to get U = [$u_1 \dots u_n$]
3. Get first k vectors. U_{reduce} = [$u_1 \dots u_k$]
4. Multiply. $z^{(i)} = U_{reduce}^T x^{(i)}$ for $0 \leq i \leq m$.

Reduction: $\tilde{U}^T X = Z \epsilon \mathbb{R}^{r \times m}$ Reconstruction: $\tilde{U}Z = X \approx X$



$x^{(i)} \epsilon \mathbb{R}^n \rightarrow z^{(i)} \epsilon \mathbb{R}^k$. Achieves compression, visualisation (k = 2 or 3). Don't use for overfitting. Pick k and retain 99% of variance, minimum k s.t. $\sum_{i=1}^{k}\sigma_i^2 / \sum_{i=1}^{n}\sigma_i^2 \geq 0.99$