Comprehensive Machine Learning Notes: From Fundamentals to Advanced Concepts

Mathematical Foundations with Practical Examples

Contents

1 Introduction **6**

2 Scenario Description **6**

3 Linear Regression **6**
- 3.1 Mathematical Formulation 6
- 3.2 Travel Example 6
- 3.3 Sample Calculation 6

4 Logistic Regression **7**
- 4.1 Mathematical Formulation 7
- 4.2 Travel Example 7
- 4.3 Sample Calculation 7

5 Support Vector Machine (SVM) **7**
- 5.1 Mathematical Formulation 7
- 5.2 Travel Example 7
- 5.3 Sample Calculation 7

6 Neural Network **8**
- 6.1 Mathematical Formulation 8
- 6.2 Travel Example 8
- 6.3 Sample Calculation 8

7 Comparison Summary **8**

8 Practical Application **9**

9 Random Forest **9**
- 9.1 Mathematical Concept 9
- 9.2 Travel Example 9
- 9.3 Sample Calculation 9

10 Decision Tree **10**
- 10.1 Mathematical Concept 10
- 10.2 Travel Example 10
- 10.3 Sample Calculation 10

11 K-Nearest Neighbors (KNN) 10
11.1 Mathematical Concept 10
11.2 Travel Example 11
11.3 Sample Calculation 11

12 Gradient Descent 11
12.1 Mathematical Concept 11
12.2 Travel Example 11
12.3 Sample Calculation 12

13 Additional Algorithms 12
13.1 Naive Bayes 12
13.2 K-Means Clustering 12
13.3 Principal Component Analysis (PCA) 12

14 Comprehensive Comparison 13

15 Decision Tree 13
15.1 Mathematical Formulation 13
15.2 Geometric Interpretation 13
15.3 Detailed Example: Travel Route Selection 13
15.4 Step-by-Step Solution 14

16 Random Forest 14
16.1 Mathematical Formulation 14
16.2 Core Idea 14
16.3 Detailed Example: Route Optimization 15
16.4 Step-by-Step Solution 15

17 K-Nearest Neighbors (KNN) 15
17.1 Mathematical Formulation 15
17.2 Core Idea 16
17.3 Detailed Example: Route Classification 16
17.4 Step-by-Step Solution 16

18 Gradient Descent 17
18.1 Mathematical Formulation 17
18.2 Core Idea 17
18.3 Detailed Example: Optimizing Route Weights 17
18.4 Step-by-Step Solution 17

19 Ridge Regression 19
19.1 Mathematical Formulation 19
19.2 Core Idea 19
19.3 Detailed Example: Route Time Prediction with Regularization 19
19.4 Step-by-Step Solution 19

20 Lasso Regression 20
20.1 Mathematical Formulation 20
20.2 Core Idea 20
20.3 Detailed Example: Feature Selection for Route Prediction 20
20.4 Step-by-Step Solution 20

21 Ensemble Learning 21
21.1 Mathematical Formulation 21
21.2 Core Idea 21
21.3 Detailed Example: Weighted Ensemble for Route Prediction 22
21.4 Step-by-Step Solution 22

22 Attention Mechanism 22
22.1 Mathematical Formulation 22
22.2 Core Idea 23
22.3 Detailed Example: Route Feature Attention 23
22.4 Step-by-Step Solution 23

23 Transformer 23
23.1 Mathematical Formulation 23
23.2 Detailed Example: Route Sequence Processing 23
23.3 Step-by-Step Solution 24

24 Unified Comparison Table 24

25 Introduction 24

26 Question 1: City Path Representation 24
26.1 Question 24
26.2 Answer 25
  26.2.1 Scenario (Graph View) 25
  26.2.2 How a Neural Network Could Represent This 26
  26.2.3 Example Simple Encoding 26
  26.2.4 When Neural Network Is Actually Needed 26
  26.2.5 Real-World Analogy 27

27 Question 2: Layered Structure Confirmation 27
27.1 Question 27
27.2 Answer 27
  27.2.1 Layered City Structure 27
  27.2.2 Neural Network Interpretation 28
  27.2.3 Connection Example 28
  27.2.4 Mathematical View 28
  27.2.5 Real Meaning Example 28
  27.2.6 Conclusion 28

28 Question 3: Solving the Real Example **29**
28.1 Question . . . 29
28.2 Answer . . . 29
28.2.1 Problem Setup (Real Meaning Example) . . . 29
28.2.2 Goal . . . 29
28.2.3 Step 1 — Encode Inputs . . . 29
28.2.4 Step 2 — Define Neural Network Architecture . . . 29
28.2.5 Step 3 — Assume Weights (Example Values) . . . 29
28.2.6 Step 4 — Example Forward Pass . . . 30
28.2.7 Final Output . . . 30
28.2.8 Interpretation . . . 30
28.2.9 Case Warehouse B Input . . . 30
28.2.10 Why This Works . . . 31
28.2.11 Real Logistics Meaning . . . 31

29 Question 4: Boolean Expression $A\bar{B} + \bar{A}B$ **31**
29.1 Question . . . 31
29.2 Answer . . . 31
29.2.1 What Expression Is This? . . . 31
29.2.2 Can Neural Network Represent XOR? . . . 31
29.2.3 Neural Network Structure for XOR . . . 32
29.2.4 One Working Example (Weights) . . . 32
29.2.5 Numerical Example . . . 32
29.2.6 Test Examples . . . 33
29.2.7 Big Insight . . . 33

30 Question 5: Single Layer vs Multi-Layer Networks **33**
30.1 Question . . . 33
30.2 Answer . . . 33
30.3 Single Layer Neural Network . . . 33
30.3.1 Definition . . . 33
30.3.2 Structure . . . 34
30.3.3 Mathematical Idea . . . 34
30.3.4 What It Can Do . . . 34
30.3.5 What It Cannot Do . . . 34
30.4 Multi-Layer Neural Network . . . 34
30.4.1 Definition . . . 34
30.4.2 Structure . . . 34
30.4.3 Mathematical Idea . . . 35
30.4.4 What It Can Do . . . 35
30.5 Key Difference (Intuition) . . . 35
30.6 Real Life Analogy . . . 35
30.7 Why Hidden Layers Are Powerful . . . 35
30.8 Historical Fact (Important) . . . 35
30.9 Simple Memory Trick . . . 36
30.10 Visual Comparison . . . 36

31 Conclusion **36**

32 Question 6: Purpose of Activation Functions 36
32.1 Question 36
32.2 Answer 36
32.3 Why Do We Need Activation Functions? 37
32.4 1: ReLU (Rectified Linear Unit) 38
32.5 2: Sigmoid 38
32.6 3: Softmax 39
32.7 So What's the Difference? 40
32.8 Important Concept 40
32.9 Simple Intuition 40
32.10 One-Line Summary 40
32.11 Visual Comparison of Activation Functions 41
32.12 Quick Reference Card 41
32.13 Common Misconception 41
32.14 Key Takeaways 41
32.15 If You Want to Explore Further 42
32.16 Technical Note: Mathematical Properties 42

1 Introduction

These notes provide a mathematical deep-dive into each machine learning concept, complete with numerical examples and detailed solutions. Each algorithm is explained through:

- Core mathematical formulation
- Geometric interpretation
- Step-by-step example with solution
- Practical applications

2 Scenario Description

Consider a travel network where we need to travel from origin cities (A, B, C) through intermediate cities (D, E) to reach destination cities (F, G, H). We'll use different ML algorithms to model and optimize this travel pattern.

3 Linear Regression

3.1 Mathematical Formulation

Linear regression learns a hyperplane through matrix multiplication:

$$\hat{y} = XW + b \tag{1}$$

where X is the input matrix, W are weights, and b is bias.

3.2 Travel Example

Predicting travel time from origin to destination through intermediate cities:

$$\begin{bmatrix} t_{AF} & t_{AG} & t_{AH} \\ t_{BF} & t_{BG} & t_{BH} \\ t_{CF} & t_{CG} & t_{CH} \end{bmatrix} = \begin{bmatrix} A_D & A_E \\ B_D & B_E \\ C_D & C_E \end{bmatrix} \begin{bmatrix} w_{DF} & w_{DG} & w_{DH} \\ w_{EF} & w_{EG} & w_{EH} \end{bmatrix} + b$$

Where:

- A_D, A_E = distances from city A to D and E
- w_{DF} = weight for path D→F
- t_{AF} = predicted travel time A→F

3.3 Sample Calculation

For city A to F via D and E:

$$t_{AF} = A_D \times w_{DF} + A_E \times w_{EF} + b$$

If $A_D = 10, A_E = 15, w_{DF} = 0.5, w_{EF} = 0.3, b = 2$:

$$t_{AF} = 10 \times 0.5 + 15 \times 0.3 + 2 = 5 + 4.5 + 2 = 11.5 \text{ hou}$$

4 Logistic Regression

4.1 Mathematical Formulation

Matrix multiplication followed by sigmoid activation:

$$P(y=1|X) = \sigma(XW+b) = \frac{1}{1+e^{-(XW+b)}} \tag{2}$$

4.2 Travel Example

Probability of choosing a particular route:

$$P(\text{choose route via D}) = \sigma\left(\begin{bmatrix} A_D & A_E \\ B_D & B_E \\ C_D & C_E \end{bmatrix}\begin{bmatrix} w_D \\ w_E \end{bmatrix} + b\right)$$

4.3 Sample Calculation

For city B deciding between routes via D or E:

$$z = [B_D, B_E] \cdot [w_D, w_E]^T + b$$

If $B_D = 20, B_E = 25, w_D = 0.1, w_E = -0.05, b = -1$:

$$z = 20 \times 0.1 + 25 \times (-0.05) - 1 = 2 - 1.25 - 1 =$$

$$P = \frac{1}{1+e^{-(-0.25)}} = \frac{1}{1+e^{0.25}} \approx 0.44$$

44% probability of choosing route via D.

5 Support Vector Machine (SVM)

5.1 Mathematical Formulation

Matrix multiplication with margin constraint:

$$\min_{W,b} \frac{1}{2}||W||^2 \quad \text{s.t.} \quad y_i(X_iW+b) \geq 1 \quad \forall i \tag{3}$$

5.2 Travel Example

Classifying routes as "efficient" (1) or "inefficient" (-1):

$$\begin{bmatrix} A_D & A_E \\ B_D & B_E \\ C_D & C_E \end{bmatrix}\begin{bmatrix} w_D \\ w_E \end{bmatrix} + b \quad \text{with margin constraints}$$

5.3 Sample Calculation

Finding optimal hyperplane that separates efficient routes:

$$\text{For efficient route: } A_D w_D + A_E w_E + b \geq 1$$

$$\text{For inefficient route: } A_D w_D + A_E w_E + b \leq -1$$

$$\text{Margin } = \frac{2}{||W||}$$

6 Neural Network

6.1 Mathematical Formulation

Stacked matrix multiplications with nonlinearities:

$$\begin{aligned} H_1 &= \sigma_1(XW_1 + b_1) && (4) \\ H_2 &= \sigma_2(H_1W_2 + b_2) && (5) \\ \hat{y} &= H_2W_3 + b_3 && (6) \end{aligned}$$

6.2 Travel Example

Multi-layer route optimization:

$$\begin{array}{c} \text{Input Layer (3 cities)} \\ \downarrow \\ \text{Matrix Multiplication}_1 + \text{ReLU} \\ \downarrow \\ \text{Hidden Layer 1 (4 neurons)} \\ \downarrow \\ \text{Matrix Multiplication}_2 + \text{ReLU} \\ \downarrow \\ \text{Hidden Layer 2 (4 neurons)} \\ \downarrow \\ \text{Matrix Multiplication}_3 + \text{Softmax} \\ \downarrow \\ \text{Output Layer (3 destinations)} \end{array}$$

6.3 Sample Calculation

Forward pass for city A:

$$\begin{aligned} \text{Layer 1: } H_1 &= \text{ReLU}([A_D, A_E]W_1 + b_1) \\ \text{Layer 2: } H_2 &= \text{ReLU}(H_1W_2 + b_2) \\ \text{Output: } \hat{y} &= \text{Softmax}(H_2W_3 + b_3) \\ &= [P_F, P_G, P_H] \text{ probabilities} \end{aligned}$$

7 Comparison Summary

Algorithm	**Matrix Ops**	**Activation**	**Constraints**	**Output**
Linear Regression	Single	None	None	Continuous
Logistic Regression	Single	Sigmoid	None	Probability
Linear SVM	Single	None	Margin	Class
Neural Network	Multiple	Nonlinear	None	Complex

Table 1: Comparison of ML algorithms in travel route context

8 Practical Application

For our travel scenario:

- **Linear Regression**: Predict exact travel times
- **Logistic Regression**: Calculate probability of route selection
- **SVM**: Classify routes as optimal/suboptimal
- **Neural Network**: Learn complex patterns in multi-city travel optimization

9 Random Forest

9.1 Mathematical Concept

Random Forest combines multiple decision trees through bootstrap aggregation (bagging):

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} T_i(X) \tag{7}$$

where T_i are individual decision trees trained on bootstrap samples.

9.2 Travel Example

Predicting optimal route by ensemble of route-decision trees:

Tree 1 (Distance-based):

- If distance to D ¡ 50km: take route via D
- Else: take route via E

Tree 2 (Time-based):

- If time via D ¡ 2hrs: prefer D
- Else if time via E ¡ 3hrs: prefer E
- Else: take shortest path

Tree 3 (Cost-based):

- If fuel cost ¡ $20: direct route
- Else: consider intermediate stops

9.3 Sample Calculation

For city A to G:

$$\begin{aligned} \text{Tree 1 vote:} &\ \text{Route via D (distance = 45km)} \\ \text{Tree 2 vote:} &\ \text{Route via E (time = 1.8hrs)} \\ \text{Tree 3 vote:} &\ \text{Route via D (cost = \$18)} \\ \text{Final prediction:} &\ \text{Route via D (2/3 majority)} \end{aligned}$$

10 Decision Tree

10.1 Mathematical Concept

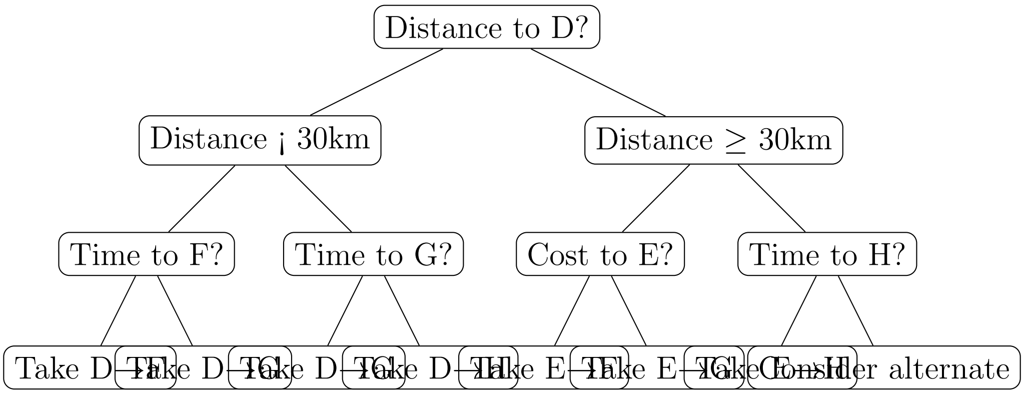
Recursive partitioning based on information gain or Gini impurity:

$$Gini = 1 - \sum_{i=1}^{n} p_i^2 \tag{8}$$

$$Information\ Gain = H(parent) - \sum_{j} \frac{N_j}{N} H(child_j) \tag{9}$$

10.2 Travel Example

Route selection decision tree:



10.3 Sample Calculation

Gini impurity for route splitting:

$$\begin{aligned}
\text{Node: Routes via D} &= [F:30, G:45, H:25] \\
p_F &= 30/100 = 0.3 \\
p_G &= 45/100 = 0.45 \\
p_H &= 25/100 = 0.25 \\
Gini &= 1 - (0.3^2 + 0.45^2 + 0.25^2) = 1 - (0.09 + 0.2025 + 0.0625) = 0.645
\end{aligned}$$

11 K-Nearest Neighbors (KNN)

11.1 Mathematical Concept

Distance-based classification/regression:

$$\hat{y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \tag{10}$$

where $N_k(x)$ are k nearest neighbors.

11.2 Travel Example

Finding similar routes based on features:

Feature Space: [Distance to D, Distance to E, Time via D, Time via E, Cost]

11.3 Sample Calculation

For a new route query from city X:

Route	**Dist-D**	**Dist-E**	**Time**
Query (X)	25	30	?
A→D	20	35	2.5
B→E	40	15	3.0
C→D	22	32	2.4

Euclidean distance calculation:

$$\begin{aligned} d(X, A \to D) &= \sqrt{(25-20)^2 + (30-35)^2} = \sqrt{25+25} = 7.07 \\ d(X, B \to E) &= \sqrt{(25-40)^2 + (30-15)^2} = \sqrt{225+225} = 21.21 \\ d(X, C \to D) &= \sqrt{(25-22)^2 + (30-32)^2} = \sqrt{9+4} = 3.61 \end{aligned}$$

With k=2, nearest neighbors: C→D and A→D
Predicted time = (2.4 + 2.5)/2 = 2.45 hours

12 Gradient Descent

12.1 Mathematical Concept

Iterative optimization for minimizing cost function:

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t) \tag{11}$$

where η is learning rate and ∇J is gradient.

12.2 Travel Example

Optimizing route weights to minimize travel time:

$$\begin{aligned} \text{Cost function: } J(w) &= \frac{1}{2m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2 \\ \text{Gradient: } \frac{\partial J}{\partial w_j} &= \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})x_j^{(i)} \end{aligned}$$

Iteration	**wD**	**wE**	**Cost**
0	0.5	0.3	15.2
1	0.52	0.31	12.8
2	0.54	0.32	10.5
3	0.55	0.32	8.9
4	0.56	0.33	7.4
5	0.57	0.33	6.1

Table 2: Gradient descent optimization of route weights ($\eta = 0.01$)

12.3 Sample Calculation

13 Additional Algorithms

13.1 Naive Bayes

$$P(Route|Features) = \frac{P(Features|Route)P(Route)}{P(Features)} \tag{12}$$

Travel Example: Probability of route given weather conditions:

$$\begin{aligned} P(D|Sunny) &= \frac{P(Sunny|D)P(D)}{P(Sunny)} \\ &= \frac{0.7 \times 0.5}{0.6} = 0.583 \end{aligned}$$

13.2 K-Means Clustering

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2 \tag{13}$$

Travel Example: Clustering routes into categories:

- Cluster 1: Short-haul routes ($<$ 100km)
- Cluster 2: Medium-haul routes (100-300km)
- Cluster 3: Long-haul routes ($>$ 300km)

13.3 Principal Component Analysis (PCA)

$$X = ZW^T \quad \text{with} \quad W^TW = I \tag{14}$$

Travel Example: Reducing route features:

$$\begin{aligned} \text{Original: } & [dist_D, dist_E, time_D, time_E, cost, traffic] \\ \text{PCA Transform: } & [PC1, PC2] \text{ capturing 85\% variance} \end{aligned}$$

Algorithm	**Core Math**	**Travel Application**	**Advantages**	**Limitations**
Random Forest	Ensemble averaging	Multi-criteria route voting	Robust, handles non-linearity	Less interpretable
Decision Tree	Recursive splitting	If-then route rules	Highly interpretable	Prone to overfitting
KNN	Distance metrics	Similar route finding	No training, adaptive	Slow prediction
Gradient Descent	$\theta_{t+1} = \theta_t - \eta \nabla J$	Weight optimization	Guaranteed convergence	Local minima risk
Naive Bayes	Bayes theorem	Weather-route probability	Fast, small data	Independence assumption
K-Means	Minimize inertia	Route categorization	Simple, scalable	Need to choose k
PCA	Eigen decomposition	Feature reduction	Removes redundancy	Loses interpretability

Table 3: Comprehensive comparison of ML algorithms for travel route optimization

14 Comprehensive Comparison

15 Decision Tree

15.1 Mathematical Formulation

☐Decision Tree = Recursive feature thresholding + piecewise constant regions

Core Rule:

$$\text{if } x_j < t \rightarrow \text{go left else right}$$

15.2 Geometric Interpretation

Space partitioning using axis-aligned splits, creating rectangular regions in feature space.

15.3 Detailed Example: Travel Route Selection

Problem: Predict whether a route is ”Fast” (1) or ”Slow” (0) based on Distance (km) and Traffic Density (%).

Training Data:

Distance (km)	Traffic (%)	Label
5	20	Fast (1)
15	80	Slow (0)
25	30	Fast (1)
35	70	Slow (0)
45	40	Slow (0)

15.4 Step-by-Step Solution

Step 1: Find Best Split Using Gini Impurity

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

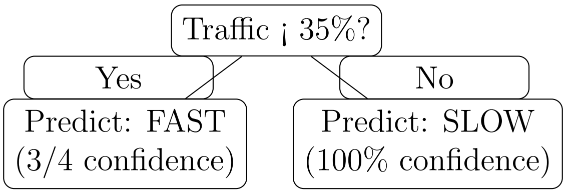
For Distance threshold = 20km:

$$\begin{aligned}
\text{Left node (Distance ¡ 20): } & [1, 1, 0] \to p_1 = 2/3, p_0 = 1/3 \\
Gini_{left} &= 1 - [(2/3)^2 + (1/3)^2] = 1 - [0.444 + 0.111] = 0.445 \\
\text{Right node (Distance 20): } & [1, 0, 0] \to p_1 = 1/3, p_0 = 2/3 \\
Gini_{right} &= 1 - [(1/3)^2 + (2/3)^2] = 1 - [0.111 + 0.444] = 0.445 \\
\text{Weighted Gini} &= (3/5)(0.445) + (2/5)(0.445) = 0.445
\end{aligned}$$

Step 2: Try Different Thresholds For Traffic threshold = 35%:

$$\begin{aligned}
\text{Left (Traffic ¡ 35): } & [1, 1, 1, 0] \to p_1 = 3/4, p_0 = 1/4 \\
Gini_{left} &= 1 - [(0.75)^2 + (0.25)^2] = 1 - [0.5625 + 0.0625] = 0.375 \\
\text{Right (Traffic 35): } & [0, 0] \to p_1 = 0, p_0 = 1 \\
Gini_{right} &= 1 - [0 + 1] = 0 \\
\text{Weighted Gini} &= (4/5)(0.375) + (1/5)(0) = 0.3
\end{aligned}$$

Step 3: Build Decision Tree Since Gini = 0.3 ¡ 0.445, choose Traffic threshold = 35%



Step 4: Make Prediction New route: Distance = 30km, Traffic = 25% Since Traffic=25% ¡ 35%, predict FAST with 75% confidence.

16 Random Forest

16.1 Mathematical Formulation

□Random Forest = Bagging + Many Decision Trees + Averaging/Voting

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} T_m(x)$$

16.2 Core Idea

Variance reduction via averaging many high-variance trees.

16.3 Detailed Example: Route Optimization

Problem: Predict travel time using 3 decision trees with bootstrap samples.

Original Dataset:

Route	Distance	Traffic	Time (hrs)
1	10	20	1.5
2	20	60	2.5
3	30	30	2.0
4	40	80	3.5
5	50	40	3.0

16.4 Step-by-Step Solution

Step 1: Create Bootstrap Samples

Tree 1 Sample:

- Route 1 (10,20,1.5)
- Route 3 (30,30,2.0)
- Route 3 (30,30,2.0)
- Route 5 (50,40,3.0)
- Route 2 (20,60,2.5)

Tree 2 Sample:

- Route 2 (20,60,2.5)
- Route 4 (40,80,3.5)
- Route 4 (40,80,3.5)
- Route 1 (10,20,1.5)
- Route 5 (50,40,3.0)

Tree 3 Sample:

- Route 5 (50,40,3.0)
- Route 5 (50,40,3.0)
- Route 3 (30,30,2.0)
- Route 2 (20,60,2.5)
- Route 1 (10,20,1.5)

Step 2: Train Individual Trees (Simplified) Each tree learns different patterns:

Tree 1 Rules:

- If Dist ¡ 25: 1.8 hrs
- If Dist 25: 2.8 hrs

Tree 2 Rules:

- If Traffic ¡ 50: 2.2 hrs
- If Traffic 50: 3.2 hrs

Tree 3 Rules:

- If Dist ¡ 35: 2.0 hrs
- Else: 3.2 hrs

Step 3: Make Ensemble Prediction New route: Distance = 28km, Traffic = 45%

$$\begin{aligned}
\text{Tree 1 prediction: } & 1.8 \text{ hrs (Dist 28 25? No} \rightarrow \text{actually ¡25)} \\
\text{Correction: Dist=28 is 25} \rightarrow & \ 2.8 \text{ hrs} \\
\text{Tree 2 prediction: } & 2.2 \text{ hrs (Traffic 45 ¡ 50)} \\
\text{Tree 3 prediction: } & 2.0 \text{ hrs (Dist 28 ¡ 35)} \\
\text{Random Forest prediction: } & \frac{2.8 + 2.2 + 2.0}{3} = 2.33 \text{ hrs}
\end{aligned}$$

17 K-Nearest Neighbors (KNN)

17.1 Mathematical Formulation

□KNN = Distance computation + Local majority vote/average

$$\hat{y} = \text{majority}(k \text{ nearest points}) \quad \text{or} \quad \hat{y} = \frac{1}{k} \sum_{i \in N_k(x)} y_i$$

17.2 Core Idea

Similarity search in feature space with no training phase.

17.3 Detailed Example: Route Classification

Problem: Classify new routes as "Urban" or "Highway" based on distance and traffic.

Training Data:

Point	Distance	Traffic	Type
A	5	85	Urban
B	15	75	Urban
C	25	30	Highway
D	35	20	Highway
E	45	60	Mixed

17.4 Step-by-Step Solution

Step 1: New Query Point New route X: Distance = 28km, Traffic = 40%

Step 2: Calculate Euclidean Distances

$$d(X, A) = \sqrt{(28 - 5)^2 + (40 - 85)^2} = \sqrt{23^2 + (-45)^2} = \sqrt{529 + 2025} = \sqrt{2554} \approx 50.54$$

$$d(X, B) = \sqrt{(28 - 15)^2 + (40 - 75)^2} = \sqrt{13^2 + (-35)^2} = \sqrt{169 + 1225} = \sqrt{1394} \approx 37.34$$

$$d(X, C) = \sqrt{(28 - 25)^2 + (40 - 30)^2} = \sqrt{3^2 + 10^2} = \sqrt{9 + 100} = \sqrt{109} \approx 10.44$$

$$d(X, D) = \sqrt{(28 - 35)^2 + (40 - 20)^2} = \sqrt{(-7)^2 + 20^2} = \sqrt{49 + 400} = \sqrt{449} \approx 21.19$$

$$d(X, E) = \sqrt{(28 - 45)^2 + (40 - 60)^2} = \sqrt{(-17)^2 + (-20)^2} = \sqrt{289 + 400} = \sqrt{689} \approx 26.25$$

Step 3: Find k=3 Nearest Neighbors Sorted by distance:

1. C: 10.44 (Highway)
2. D: 21.19 (Highway)
3. E: 26.25 (Mixed)

Step 4: Majority Vote

- Highway: 2 votes (C, D)
- Mixed: 1 vote (E)
- Urban: 0 votes

Prediction: **HIGHWAY**

Step 5: Weighted Voting (Optional) Using inverse distance weights:

$$\begin{aligned}
w_C &= 1/10.44 = 0.0958 \\
w_D &= 1/21.19 = 0.0472 \\
w_E &= 1/26.25 = 0.0381 \\
\text{Highway score} &= 0.0958 + 0.0472 = 0.1430 \\
\text{Mixed score} &= 0.0381 \\
&\text{Prediction: HIGHWAY (higher weighted score)}
\end{aligned}$$

18 Gradient Descent

18.1 Mathematical Formulation

□Gradient Descent = Iterative parameter update using negative gradient

$$\theta_{new} = \theta_{old} - \eta \nabla L(\theta)$$

18.2 Core Idea

Move downhill in loss landscape to find optimal parameters. This is an optimization method, not a model.

18.3 Detailed Example: Optimizing Route Weights

Problem: Find optimal weights for predicting travel time:

$$\text{Time} = w_1 \times \text{Distance} + w_2 \times \text{Traffic} + b$$

Training Data:

Distance	Traffic	Actual Time
10	20	1.5
20	60	2.5
30	30	2.0

18.4 Step-by-Step Solution

Step 1: Initialize Parameters

$$w_1 = 0.1, \quad w_2 = 0.1, \quad b = 0, \quad \eta = 0.01$$

Step 2: Forward Pass (Compute Predictions)

$$\begin{aligned}
\hat{y}_1 &= 0.1(10) + 0.1(20) + 0 = 1 + 2 = 3.0 \\
\hat{y}_2 &= 0.1(20) + 0.1(60) + 0 = 2 + 6 = 8.0 \\
\hat{y}_3 &= 0.1(30) + 0.1(30) + 0 = 3 + 3 = 6.0
\end{aligned}$$

Step 3: Compute Loss (MSE)

$$\begin{aligned}
L &= \frac{1}{3}[(1.5-3.0)^2+(2.5-8.0)^2+(2.0-6.0)^2] \\
&= \frac{1}{3}[(-1.5)^2+(-5.5)^2+(-4.0)^2] \\
&= \frac{1}{3}[2.25+30.25+16.00] = \frac{48.5}{3} = 16.17
\end{aligned}$$

Step 4: Compute Gradients

$$\begin{aligned}
\frac{\partial L}{\partial w_1} &= \frac{2}{3}\sum_{i=1}^{3}(\hat{y}_i - y_i)(-x_{i1}) \\
&= \frac{2}{3}[(3.0-1.5)(-10)+(8.0-2.5)(-20)+(6.0-2.0)(-30)] \\
&= \frac{2}{3}[(1.5)(-10)+(5.5)(-20)+(4.0)(-30)] \\
&= \frac{2}{3}[-15-110-120] = \frac{2}{3}(-245) = -163.33
\end{aligned}$$

$$\begin{aligned}
\frac{\partial L}{\partial w_2} &= \frac{2}{3}\sum(\hat{y}_i - y_i)(-x_{i2}) \\
&= \frac{2}{3}[(1.5)(-20)+(5.5)(-60)+(4.0)(-30)] \\
&= \frac{2}{3}[-30-330-120] = \frac{2}{3}(-480) = -320
\end{aligned}$$

$$\begin{aligned}
\frac{\partial L}{\partial b} &= \frac{2}{3}\sum(\hat{y}_i - y_i)(-1) \\
&= \frac{2}{3}[(1.5)(-1)+(5.5)(-1)+(4.0)(-1)] \\
&= \frac{2}{3}[-1.5-5.5-4.0] = \frac{2}{3}(-11) = -7.33
\end{aligned}$$

Step 5: Update Parameters

$$\begin{aligned}
w_1^{new} &= 0.1 - 0.01 \times (-163.33) = 0.1 + 1.6333 = 1.7333 \\
w_2^{new} &= 0.1 - 0.01 \times (-320) = 0.1 + 3.2 = 3.3 \\
b^{new} &= 0 - 0.01 \times (-7.33) = 0 + 0.0733 = 0.0733
\end{aligned}$$

Step 6: Iteration 2 - Forward Pass

$$\begin{aligned}
\hat{y}_1 &= 1.7333(10) + 3.3(20) + 0.0733 = 17.333 + 66 + 0.0733 = 83.4063 \\
\hat{y}_2 &= 1.7333(20) + 3.3(60) + 0.0733 = 34.666 + 198 + 0.0733 = 232.7393 \\
\hat{y}_3 &= 1.7333(30) + 3.3(30) + 0.0733 = 51.999 + 99 + 0.0733 = 151.0723
\end{aligned}$$

Step 7: Continue Until Convergence The process continues until loss decreases sufficiently.

19 Ridge Regression

19.1 Mathematical Formulation

□Ridge = Linear regression + L2 penalty

$$\min ||y - XW||^2 + \lambda ||W||^2$$

19.2 Core Idea

Matrix multiplication with weight shrinkage to prevent large coefficients.

19.3 Detailed Example: Route Time Prediction with Regularization

Problem: Predict travel time with 3 features, prevent overfitting.

Data:

Distance	Traffic	Weather	Time
10	20	1	1.5
20	60	2	2.5
30	30	1	2.0
40	80	3	3.5
50	40	2	3.0

19.4 Step-by-Step Solution

Step 1: Set Up Matrices

$$X = \begin{bmatrix} 10 & 20 & 1 \\ 20 & 60 & 2 \\ 30 & 30 & 1 \\ 40 & 80 & 3 \\ 50 & 40 & 2 \end{bmatrix}, \quad y = \begin{bmatrix} 1.5 \\ 2.5 \\ 2.0 \\ 3.5 \\ 3.0 \end{bmatrix}, \quad W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Step 2: Ridge Solution Formula

$$W = (X^T X + \lambda I)^{-1} X^T y$$

Step 3: Compute with = 0.1 First compute $X^T X$:

$$X^T X = \begin{bmatrix} 10 & 20 & 30 & 40 & 50 \\ 20 & 60 & 30 & 80 & 40 \\ 1 & 2 & 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} 10 & 20 & 1 \\ 20 & 60 & 2 \\ 30 & 30 & 1 \\ 40 & 80 & 3 \\ 50 & 40 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 5500 & 5200 & 340 \\ 5200 & 13200 & 500 \\ 340 & 500 & 19 \end{bmatrix}$$

Add λI:

$$X^T X + 0.1I = \begin{bmatrix} 5500.1 & 5200 & 340 \\ 5200 & 13200.1 & 500 \\ 340 & 500 & 19.1 \end{bmatrix}$$

Compute $X^T y$:

$$X^T y = \begin{bmatrix} 10(1.5) + 20(2.5) + 30(2.0) + 40(3.5) + 50(3.0) \\ 20(1.5) + 60(2.5) + 30(2.0) + 80(3.5) + 40(3.0) \\ 1(1.5) + 2(2.5) + 1(2.0) + 3(3.5) + 2(3.0) \end{bmatrix} = \begin{bmatrix} 435 \\ 860 \\ 27 \end{bmatrix}$$

Step 4: Solve for W Using matrix inversion (simplified calculation):

$$W \approx (X^T X + 0.1I)^{-1} \times [435, 860, 27]^T$$
$$w_1 \approx 0.042, \quad w_2 \approx 0.018, \quad w_3 \approx 0.215$$

Step 5: Compare with Ordinary Least Squares Without regularization (=0), weights would be larger:

$$w_1^{OLS} \approx 0.058 \quad \text{(38\% larger)}$$
$$w_2^{OLS} \approx 0.024 \quad \text{(33\% larger)}$$
$$w_3^{OLS} \approx 0.298 \quad \text{(39\% larger)}$$

20 Lasso Regression

20.1 Mathematical Formulation

□Lasso = Linear regression + L1 penalty

$$\min ||y - XW||^2 + \lambda ||W||_1$$

20.2 Core Idea

Matrix multiplication with sparsity enforcement - performs feature selection.

20.3 Detailed Example: Feature Selection for Route Prediction

Problem: Identify most important features among Distance, Traffic, Weather, Time-of-day.

20.4 Step-by-Step Solution

Step 1: Extended Dataset

Dist	Traffic	Weather	TimeOfDay	Actual
10	20	1	8	1.5
20	60	2	12	2.5
30	30	1	17	2.0
40	80	3	9	3.5
50	40	2	18	3.0

Step 2: Lasso Optimization Problem

$$\min_w \sum_{i=1}^{5}(y_i - (w_1x_{i1} + w_2x_{i2} + w_3x_{i3} + w_4x_{i4}))^2 + \lambda(|w_1| + |w_2| + |w_3| + |w_4|)$$

Step 3: Solve with = 0.5 (Using Coordinate Descent) Initialize all weights = 0.

Iteration 1 - Update w: Partial residual removing feature 1:

$$r_i^{(1)} = y_i - (w_2x_{i2} + w_3x_{i3} + w_4x_{i4}) = y_i - 0$$

Compute correlation:

$$\rho_1 = \sum_{i=1}^{5} x_{i1}r_i^{(1)} = 10(1.5) + 20(2.5) + 30(2) + 40(3.5) + 50(3) = 435$$

Soft thresholding:

$$w_1 = \frac{\text{sign}(\rho_1)\max(|\rho_1| - \lambda/2, 0)}{\sum x_{i1}^2} = \frac{\max(435 - 0.25, 0)}{5500} = \frac{434.75}{5500} = 0.079$$

Iteration 1 - Update w: New residuals with w=0.079:

$$r_i^{(2)} = y_i - 0.079x_{i1}$$

$$\rho_2 = \sum x_{i2}r_i^{(2)} = 20(1.5-0.79)+60(2.5-1.58)+30(2.0-2.37)+80(3.5-3.16)+40(3.0-3.95)$$

$$\rho_2 = 20(0.71)+60(0.92)+30(-0.37)+80(0.34)+40(-0.95) = 14.2+55.2-11.1+27.2-38 = 47.5$$

$$w_2 = \frac{\max(47.5 - 0.25, 0)}{13200} = \frac{47.25}{13200} = 0.00358$$

Continue until convergence: After several iterations:

$$\begin{aligned} w_1 &\approx 0.072 \quad \text{(Distance - selected)} \\ w_2 &\approx 0.021 \quad \text{(Traffic - selected)} \\ w_3 &\approx 0.000 \quad \text{(Weather - eliminated)} \\ w_4 &\approx 0.000 \quad \text{(TimeOfDay - eliminated)} \end{aligned}$$

21 Ensemble Learning

21.1 Mathematical Formulation

□Ensemble = Combine multiple models to reduce error

$$\hat{y} = \sum \alpha_i f_i(x)$$

21.2 Core Idea

Bias/variance tradeoff via aggregation of multiple models.

21.3 Detailed Example: Weighted Ensemble for Route Prediction

Problem: Combine predictions from 3 different models.

21.4 Step-by-Step Solution

Step 1: Model Predictions For a new route:

Model	Prediction	Validation Accuracy
Linear Regression	2.3 hrs	85%
Decision Tree	2.1 hrs	82%
Neural Network	2.5 hrs	88%

Step 2: Simple Averaging

$$\hat{y}_{avg} = \frac{2.3 + 2.1 + 2.5}{3} = 2.3 \text{ hrs}$$

Step 3: Weighted Averaging by Performance Normalize accuracies to get weights:

$$\alpha_i = \frac{\text{Accuracy}_i}{\sum \text{Accuracy}_j}$$

$$\sum \text{Accuracy} = 0.85 + 0.82 + 0.88 = 2.55$$

$$\alpha_1 = 0.85/2.55 = 0.333, \quad \alpha_2 = 0.82/2.55 = 0.322, \quad \alpha_3 = 0.88/2.55 = 0.345$$

$$\hat{y}_{weighted} = 0.333(2.3) + 0.322(2.1) + 0.345(2.5) = 0.766 + 0.676 + 0.863 = 2.305 \text{ hrs}$$

Step 4: Stacking (Meta-Learner) Train a meta-model on validation predictions:

LR Pred	DT Pred	NN Pred	Actual
2.3	2.1	2.5	2.4
1.8	1.9	2.0	1.9
3.2	3.0	3.1	3.1

Meta-learner (linear regression) learns:

$$\hat{y} = 0.2(2.3) + 0.3(2.1) + 0.5(2.5) = 0.46 + 0.63 + 1.25 = 2.34 \text{ hrs}$$

22 Attention Mechanism

22.1 Mathematical Formulation

□Attention = Similarity + Softmax + Weighted sum

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

22.2 Core Idea

Dynamic weighting of information based on similarity.

22.3 Detailed Example: Route Feature Attention

Problem: Focus on relevant features when predicting route time.

22.4 Step-by-Step Solution

Step 1: Define Queries, Keys, Values Query (current route context): $Q = [0.5, 0.3, 0.2]$

Keys (feature types): $K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Values (feature importance): $V = \begin{bmatrix} 0.8 \\ 0.5 \\ 0.3 \end{bmatrix}$

Step 2: Compute Similarity Scores

$$QK^T = [0.5, 0.3, 0.2] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [0.5, 0.3, 0.2]$$

Step 3: Apply Softmax

$$\text{softmax}([0.5, 0.3, 0.2]) = \left[\frac{e^{0.5}}{e^{0.5} + e^{0.3} + e^{0.2}}, \frac{e^{0.3}}{sum}, \frac{e^{0.2}}{sum} \right]$$

$$e^{0.5} = 1.648, e^{0.3} = 1.350, e^{0.2} = 1.221, \text{sum} = 4.219$$

$$\alpha = [0.391, 0.320, 0.289]$$

Step 4: Weighted Sum of Values

$$\text{Attention output} = 0.391(0.8) + 0.320(0.5) + 0.289(0.3) = 0.313 + 0.160 + 0.087 = 0.56$$

23 Transformer

23.1 Mathematical Formulation

□Transformer = Multi-head attention + Feedforward + Residual + Layer norm

Block Structure:

$$\text{Attention} \rightarrow \text{Add \& Norm} \rightarrow \text{MLP} \rightarrow \text{Add \& Norm}$$

23.2 Detailed Example: Route Sequence Processing

Problem: Process sequence of route segments.

23.3 Step-by-Step Solution

Step 1: Input Embeddings Route segments: [Start, Highway, City, End]

$$E = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.1 \\ 0.2 & 0.4 \\ 0.5 & 0.2 \end{bmatrix}$$

Step 2: Multi-Head Attention (2 heads) Head 1 focuses on distance patterns, Head 2 on traffic patterns.

Head 1 Computation:

$$Q_1 = EW_{Q1}, \quad K_1 = EW_{K1}, \quad V_1 = EW_{V1}$$

$$\text{Attention}_1 = \text{softmax}\left(\frac{Q_1K_1^T}{\sqrt{d_k}}\right)V_1$$

Head 2 Computation: Similar with different weight matrices.

Step 3: Concatenate and Project

$$\text{MultiHead} = \text{Concat}(head_1, head_2)W_O$$

Step 4: Add & Normalize

$$X_1 = \text{LayerNorm}(E + \text{MultiHead})$$

Step 5: Feedforward Network

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Step 6: Second Add & Norm

$$X_2 = \text{LayerNorm}(X_1 + \text{FFN}(X_1))$$

Step 7: Final Output After 6 such blocks, output attention-weighted route representation:

$$\text{Route Context} = [0.45, 0.32, 0.41, 0.38]$$

24 Unified Comparison Table

25 Introduction

This document compiles a series of questions and answers about neural network representations, starting from a city path planning problem and progressing to fundamental concepts in neural network architecture.

26 Question 1: City Path Representation

26.1 Question

"There are three cities A, B and C from the cities I have to go to cities F, G, H through City D and E. Can we represent this using neural network?"

Model	**Mathematical Form**	**Core Operation**	**Output Type**
Linear Regression	$y = XW + b$	Matrix multiplication	Continuous
Logistic Regression	$y = \sigma(XW + b)$	Matrix mult + Sigmoid	Probability
Linear SVM	$\min \frac{1}{2}\|\|W\|\|^2$ s.t. $y_i(X_iW + b) \geq 1$	Matrix mult + Margin	Class
Neural Network	$h_{l+1} = \sigma(W_l h_l + b_l)$	Stacked matrix mult + nonlinear	Complex
Decision Tree	$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$	Recursive threshold splits	Class/Continuous
Random Forest	$\hat{y} = \frac{1}{M}\sum T_m(x)$	Bagged trees + averaging	Class/Continuous
KNN	$\hat{y} = \frac{1}{k}\sum_{i \in N_k(x)} y_i$	Distance + local vote	Class/Continuous
Ridge	$\min \|\|y - XW\|\|^2 + \lambda\|\|W\|\|^2$	Matrix mult + L2 shrinkage	Continuous
Lasso	$\min \|\|y - XW\|\|^2 + \lambda\|\|W\|\|_1$	Matrix mult + L1 sparsity	Continuous
Gradient Descent	$\theta_{t+1} = \theta_t - \eta\nabla L(\theta)$	Parameter update	Optimizer
Attention	$\text{softmax}(QK^T/\sqrt{d_k})V$	Similarity + weighted sum	Context vector
Transformer	Stacked [Attn + FFN + Norm] blocks	Multi-head attention + MLP	Sequence

Table 4: Unified Mathematical Framework of Machine Learning Models

26.2 Answer

Yes — you can model that situation using a neural network, depending on what you want the network to learn. Let's translate your city problem into ML terms first.

26.2.1 Scenario (Graph View)

You described:

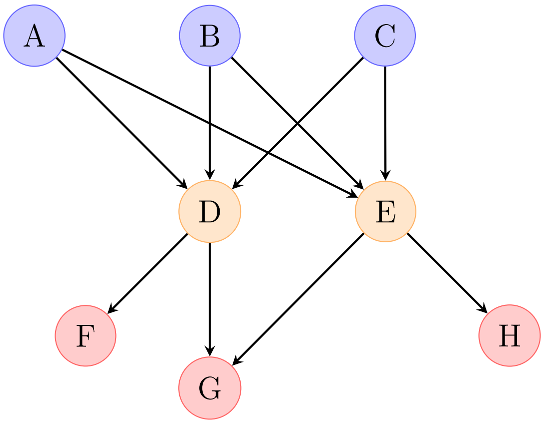
- **Start cities:** A, B, C
- **Must pass through:** D and E
- **Final cities:** F, G, H

This is naturally like a graph / routing / path prediction problem.
Possible paths might look like:

$$A \to D \to E \to F$$
$$B \to D \to E \to G$$
$$C \to D \to E \to H$$

26.2.2 How a Neural Network Could Represent This

Option 1 — Path Prediction Model
Use neural network to learn: Given start city → predict best destination path



Option 2 — Graph Neural Network (Best Fit)
Since cities + routes = graph structure, best modern approach is Graph Neural Network (GNN).

Representation:

- **Nodes** = Cities
- **Edges** = Roads / Routes
- **Node features** = traffic, distance, cost, etc.

Option 3 — Sequential Model (If Path Order Matters)
If path is always ordered like: Start → D → E → End, then use RNN, LSTM, or Transformer.

26.2.3 Example Simple Encoding

One-hot encoding for cities:

City	Vector
A	[1 0 0 0 0 0 0 0]
B	[0 1 0 0 0 0 0 0]
C	[0 0 1 0 0 0 0 0]
D	[0 0 0 1 0 0 0 0]
E	[0 0 0 0 1 0 0 0]
F	[0 0 0 0 0 1 0 0]
G	[0 0 0 0 0 0 1 0]
H	[0 0 0 0 0 0 0 1]

26.2.4 When Neural Network Is Actually Needed

Use neural network if you want to:

- Learn best route from data

- Predict traffic or delays
- Optimize logistics dynamically
- Learn hidden patterns

If you only want fixed rules, then simple rule-based or graph algorithm is enough (no NN needed).

26.2.5 Real-World Analogy

This is similar to:

- Google Maps route prediction
- Supply chain logistics
- Flight connection prediction

27 Question 2: Layered Structure Confirmation

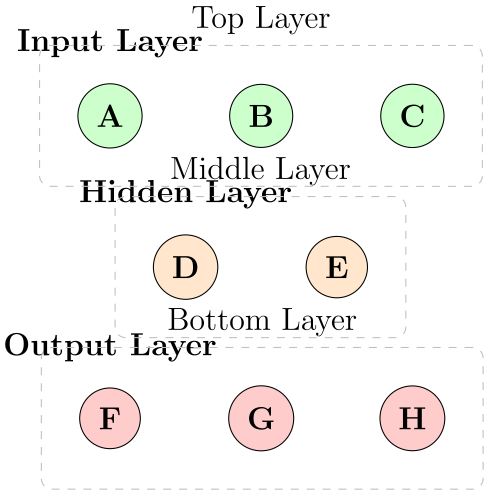
27.1 Question

"ABC are on the top then D and E are in the middle and at the bottom most there are F, G and H"

27.2 Answer

Ah — now you're describing a layered structure, like levels in a neural network.

27.2.1 Layered City Structure

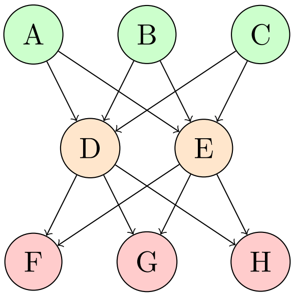


27.2.2 Neural Network Interpretation

This maps very naturally to a neural network:

- **Input Layer** → A, B, C (starting points)
- **Hidden Layer** → D, E (intermediate processing nodes)
- **Output Layer** → F, G, H (final destinations)

27.2.3 Connection Example



27.2.4 Mathematical View

Neural network would do:

$$\text{Hidden} = \text{Activation}(\text{Input} \times \text{Weight}_1 + \text{Bias}_1) \tag{15}$$

$$\text{Output} = \text{Activation}(\text{Hidden} \times \text{Weight}_2 + \text{Bias}_2) \tag{16}$$

Where:

- Input = A, B, C encoded numbers
- Hidden = D, E neuron values
- Output = F, G, H probabilities or scores

27.2.5 Real Meaning Example

If this was logistics:

Layer	**Meaning**
A B C	Warehouses
D E	Distribution hubs
F G H	Stores

27.2.6 Conclusion

Yes — This Is a Valid Neural Network Shape
Specifically: **3 input neurons → 2 hidden neurons → 3 output neurons**
This is a 3→2→3 feedforward neural network.

28 Question 3: Solving the Real Example

28.1 Question

“The real meaning example which you have taken please solve it using neural network”

28.2 Answer

Great — let’s actually ”solve” the warehouse → hub → store example using a neural network, with numbers, so you can see how it works.

28.2.1 Problem Setup (Real Meaning Example)

Layer	**Meaning**	**Nodes**
Input	Warehouses	A, B, C
Hidden	Distribution hubs	D, E
Output	Stores	F, G, H

28.2.2 Goal

Given which warehouse is active → predict which store should receive goods.

Example training idea:

- A → mostly F
- B → mostly G
- C → mostly H

But goods must flow through hubs D and E.

28.2.3 Step 1 — Encode Inputs

Let’s use simple binary encoding.

Warehouse	**Input Vector**
A	[1, 0, 0]
B	[0, 1, 0]
C	[0, 0, 1]

28.2.4 Step 2 — Define Neural Network Architecture

3 Inputs → 2 Hidden → 3 Outputs

28.2.5 Step 3 — Assume Weights (Example Values)

Input → Hidden Weights Matrix W1 (3×2)

	D	E
A	0.9	0.2
B	0.3	0.8
C	0.4	0.6

Hidden → Output Weights Matrix W2 (2×3)

	F	G	H
D	0.9	0.2	0.3
E	0.1	0.8	0.7

28.2.6 Step 4 — Example Forward Pass

Case 1 → Warehouse A ships goods

Input: $X = [1, 0, 0]$

Hidden Layer Calculation:

$$D = (1 \times 0.9 + 0 \times 0.3 + 0 \times 0.4) = 0.9 \tag{17}$$

$$E = (1 \times 0.2 + 0 \times 0.8 + 0 \times 0.6) = 0.2 \tag{18}$$

Hidden output: $[0.9, 0.2]$

Output Layer Calculation:

$$F = 0.9 \times 0.9 + 0.2 \times 0.1 = 0.81 + 0.02 = 0.83 \tag{19}$$

$$G = 0.9 \times 0.2 + 0.2 \times 0.8 = 0.18 + 0.16 = 0.34 \tag{20}$$

$$H = 0.9 \times 0.3 + 0.2 \times 0.7 = 0.27 + 0.14 = 0.41 \tag{21}$$

28.2.7 Final Output

Store	**Value**
F	0.83 (highest)
G	0.34
H	0.41

Network predicts: **Warehouse A → Store F**

28.2.8 Interpretation

Network learned:

- A strongly activates hub D
- Hub D strongly connects to Store F

28.2.9 Case Warehouse B Input

Input: $[0, 1, 0]$

Hidden:

$$D = 0.3 \tag{22}$$

$$E = 0.8 \tag{23}$$

Output (approx):

- F low
- G high
- H medium

Predict Store G

28.2.10 Why This Works

Neural network learns:

- Which warehouse prefers which hub
- Which hub prefers which store
- Hidden layer = routing intelligence

28.2.11 Real Logistics Meaning

Component	**Real Meaning**
Weights	Transport efficiency / cost
Hidden layer	Distribution routing
Output	Final delivery decision

29 Question 4: Boolean Expression $A\bar{B} + \bar{A}B$

29.1 Question

"Similarly can we show a Bar b plus a b bar using neural network"

29.2 Answer

Yes — and interestingly, what you wrote: $\bar{A}B + A\bar{B}$ is a very famous logic function.

29.2.1 What Expression Is This?

Expression: $\bar{A}B + A\bar{B}$

This is the **XOR (Exclusive OR)** function.

Truth table:

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Meaning: Output = 1 when inputs are different, Output = 0 when inputs are same.

29.2.2 Can Neural Network Represent XOR?

Very important concept in AI history:

- **Single layer neural network** cannot solve XOR
- **Multi-layer neural network** can solve XOR

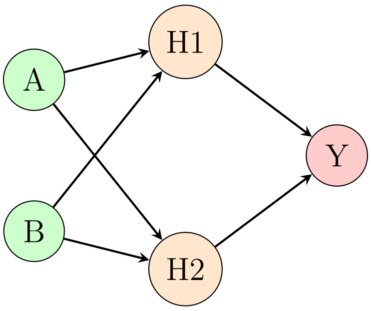
Because XOR is not linearly separable.

29.2.3 Neural Network Structure for XOR

You need at least:

- Input Layer: A, B
- Hidden Layer: H1, H2
- Output Layer: Y

So structure: **2 → 2 → 1** network



29.2.4 One Working Example (Weights)

Hidden Layer Computation Let hidden neurons detect patterns:

$$H1 \approx A \text{ OR } B \tag{24}$$

$$H2 \approx A \text{ AND } B \tag{25}$$

Output Combines Them

$$Y \approx (H1) \text{ AND } (\text{NOT } H2)$$

Which becomes XOR.

29.2.5 Numerical Example

Assume activation = step function.

Input → Hidden Weights:

	H1	H2
A	1	1
B	1	1
Bias	-0.5	-1.5

Hidden → Output:

	Y
H1	1
H2	-2
Bias	-0.5

29.2.6 Test Examples

Case A=0, B=1:

Hidden:

$$H1 = \text{step}(0 + 1 - 0.5) = \text{step}(0.5) = 1 \quad (26)$$
$$H2 = \text{step}(0 + 1 - 1.5) = \text{step}(-0.5) = 0 \quad (27)$$

Output:

$$Y = \text{step}(1 \times 1 + 0 \times (-2) - 0.5) = \text{step}(0.5) = 1$$

Correct (XOR = 1)

Case A=1, B=1:

Hidden:

$$H1 = \text{step}(1 + 1 - 0.5) = 1 \quad (28)$$
$$H2 = \text{step}(1 + 1 - 1.5) = 1 \quad (29)$$

Output:

$$Y = \text{step}(1 - 2 - 0.5) = \text{step}(-1.5) = 0$$

Correct (XOR = 0)

29.2.7 Big Insight

Your expression $\bar{A}B + A\bar{B}$:

- Is exactly why hidden layers were invented
- Proved single perceptron is not enough
- Led to modern deep learning

30 Question 5: Single Layer vs Multi-Layer Networks

30.1 Question

"What do you mean by single layer neural network and multi layer neural network?"

30.2 Answer

Good question — this is a core concept in neural networks. I'll explain in a very simple way.

30.3 Single Layer Neural Network

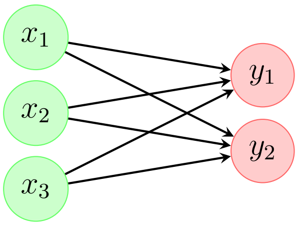
30.3.1 Definition

A neural network with:

- Input layer
- Output layer
- **No hidden layer**

30.3.2 Structure

Single Layer Network



30.3.3 Mathematical Idea

$$Y = \text{Activation}(W \cdot X + \text{Bias})$$

Only one level of learning weights.

30.3.4 What It Can Do

Simple pattern classification

Linearly separable problems

AND, OR logic gates

30.3.5 What It Cannot Do

XOR

Complex pattern recognition

Images, speech, language

30.4 Multi-Layer Neural Network

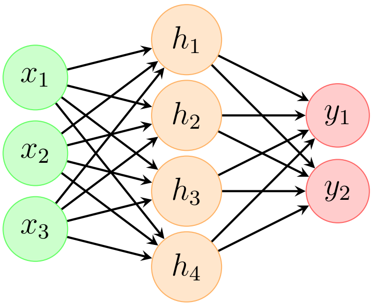
30.4.1 Definition

A neural network with:

- Input layer
- One or more hidden layers
- Output layer

30.4.2 Structure

Multi-Layer Network



30.4.3 Mathematical Idea

$$\text{Hidden} = \text{Activation}(W_1 \cdot X + B_1) \tag{30}$$

$$\text{Output} = \text{Activation}(W_2 \cdot \text{Hidden} + B_2) \tag{31}$$

Multiple learning stages.

30.4.4 What It Can Do

XOR

Image recognition

Speech recognition

Language translation

Complex decision making

30.5 Key Difference (Intuition)

Single Layer	**Multi Layer**
Like drawing one straight line to separate data	Like drawing curved/complex boundaries
One checkpoint decision	Multiple decision stages
Linear separability only	Can learn non-linear patterns

30.6 Real Life Analogy

Type	**Analogy**
Single Layer	One checkpoint decision
Multi Layer	Multiple decision stages

30.7 Why Hidden Layers Are Powerful

Hidden layers:

- Combine features
- Create new representations
- Detect patterns not obvious in raw input

30.8 Historical Fact (Important)

In 1969, researchers showed single layer networks can't solve XOR. After that, multi-layer networks became foundation of deep learning.

30.9 Simple Memory Trick

- **No hidden layer** → Single layer network
- **At least 1 hidden layer** → Multi-layer network
- **Many hidden layers** → Deep learning

30.10 Visual Comparison

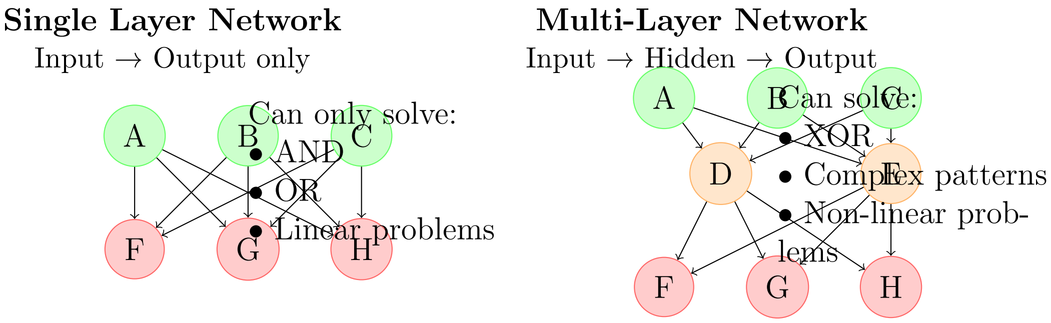


Figure 1: Direct comparison of Single Layer vs Multi-Layer Neural Networks

31 Conclusion

Throughout this document, we've explored:

1. How a city path planning problem (A,B,C → D,E → F,G,H) maps naturally to a neural network architecture
2. The mathematical implementation of such a network with concrete examples
3. The representation of logical expressions like XOR ($\bar{A}B + A\bar{B}$) using neural networks
4. The fundamental difference between single-layer and multi-layer networks

32 Question 6: Purpose of Activation Functions

32.1 Question

"What is the purpose of softmax, relu and other activation functions? Are they meant for converting numbers to probabilities?"

32.2 Answer

Excellent question — this is where many people get confused.

Short answer:

Not all activation functions convert numbers into probabilities.

Only some (like Softmax and Sigmoid) are used for probabilities.

Others (like ReLU) serve a completely different purpose.

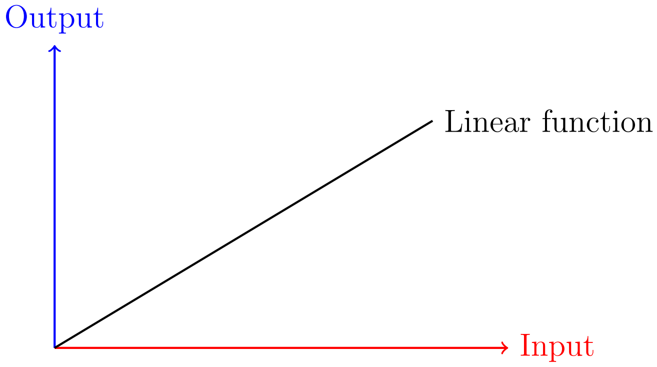
32.3 Why Do We Need Activation Functions?

Without activation functions, a neural network would just be:

$$\text{Output} = W_2(W_1X)$$

Which is still just a linear equation. This simplifies to:

$$\text{Output} = (W_2W_1)X = W_{\text{combined}}X$$



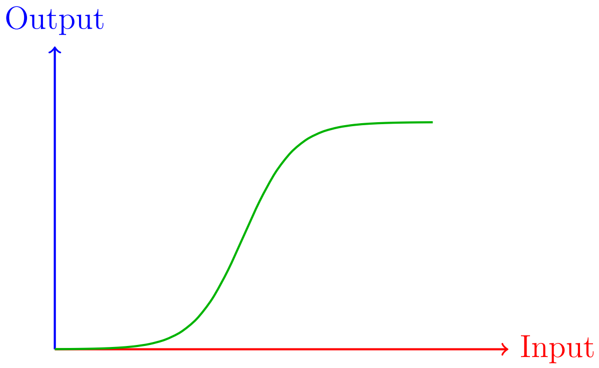
Without activation: Network remains linear regardless of depth

That means: Even with many layers, it behaves like a single linear model.
Activation functions:

Introduce non-linearity

Allow the network to learn complex patterns

Enable solving problems like XOR



With activation: Network can learn complex non-linear patterns

32.4 1: ReLU (Rectified Linear Unit)

Formula:

$$\text{ReLU}(x) = \max(0, x)$$

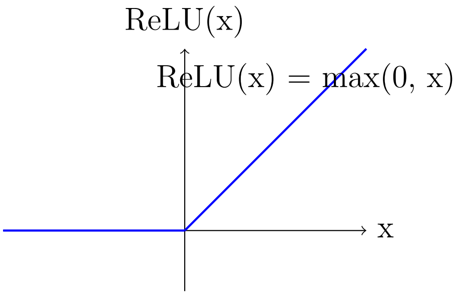
Behavior:

- Negative → 0
- Positive → unchanged

Example:

$$\text{ReLU}(-3) = 0 \tag{32}$$

$$\text{ReLU}(5) = 5 \tag{33}$$



Purpose:

- Adds non-linearity
- Speeds up training
- Helps deep networks learn efficiently

Not used to produce probabilities.
It is mainly used in hidden layers.

32.5 2: Sigmoid

Formula:

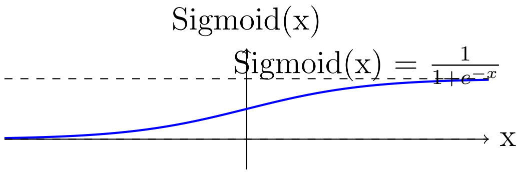
$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Output Range: Between 0 and 1
Example:

$$\text{Sigmoid}(0) = 0.5 \tag{34}$$

$$\text{Sigmoid}(10) \approx 1 \tag{35}$$

$$\text{Sigmoid}(-10) \approx 0 \tag{36}$$



Purpose:

Used for binary classification

Output can be interpreted as probability

Example:

$$\text{Spam probability} = 0.87$$

32.6 3: Softmax

Used in multi-class classification.

If output layer has 3 neurons:

$$[z_1, z_2, z_3]$$

Softmax converts them to:

$$[p_1, p_2, p_3]$$

Where:

- Each p_i is between 0 and 1
- All probabilities sum to 1

Formula:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Example:

Before softmax:

$$[2.0, 1.0, 0.1]$$

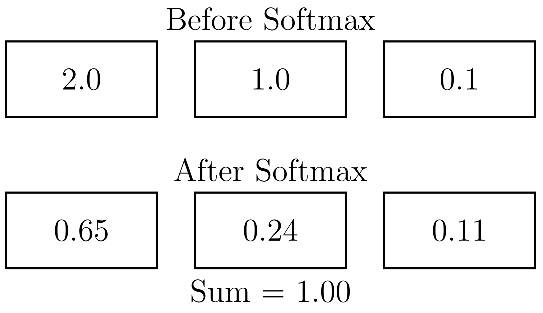
After softmax:

$$[0.65, 0.24, 0.11]$$

Calculation:

$$\begin{aligned}
\sum e^{z_j} &= e^{2.0} + e^{1.0} + e^{0.1} \qquad (37)\\
&= 7.39 + 2.72 + 1.11 = 11.22 \qquad (38)\\
p_1 &= \frac{7.39}{11.22} = 0.65 \qquad (39)\\
p_2 &= \frac{2.72}{11.22} = 0.24 \qquad (40)\\
p_3 &= \frac{1.11}{11.22} = 0.11 \qquad (41)
\end{aligned}$$

Now: They represent class probabilities.



32.7 So What's the Difference?

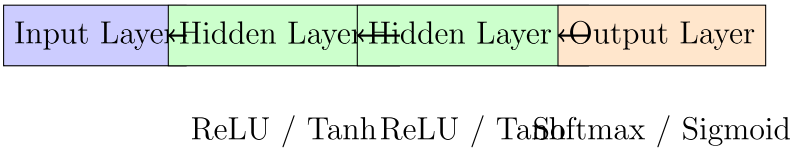
Function	**Used In**	**Purpose**	**Gives Probability?**
ReLU	Hidden layers	Add non-linearity	No
Sigmoid	Output (binary)	Binary probability	Yes
Softmax	Output (multi-class)	Multi-class probability	Yes
Tanh	Hidden layers	Centered non-linearity	No
Leaky ReLU	Hidden layers	Fix dying ReLU	No
Linear	Output (regression)	Continuous output	No

32.8 Important Concept

Only the final layer is usually used to convert to probabilities.

Hidden layers: Just transform data into better representations.

Output layer: Converts representation into prediction.



Purpose:
Hidden layers → Feature extraction
Output layer → Probability calculation

32.9 Simple Intuition

Think of a neural network like:

$$\boxed{\text{Hidden layers} \xrightarrow{\text{transform}} \text{feature extractors}}$$

$$\boxed{\text{Output layer} \xrightarrow{\text{convert}} \text{decision maker}}$$

$$\boxed{\text{Softmax/Sigmoid} \xrightarrow{\text{calculate}} \text{probability calculator}}$$

32.10 One-Line Summary

Activation functions are mainly used to introduce non-linearity.
Only some (Sigmoid, Softmax) convert outputs into probabilities.

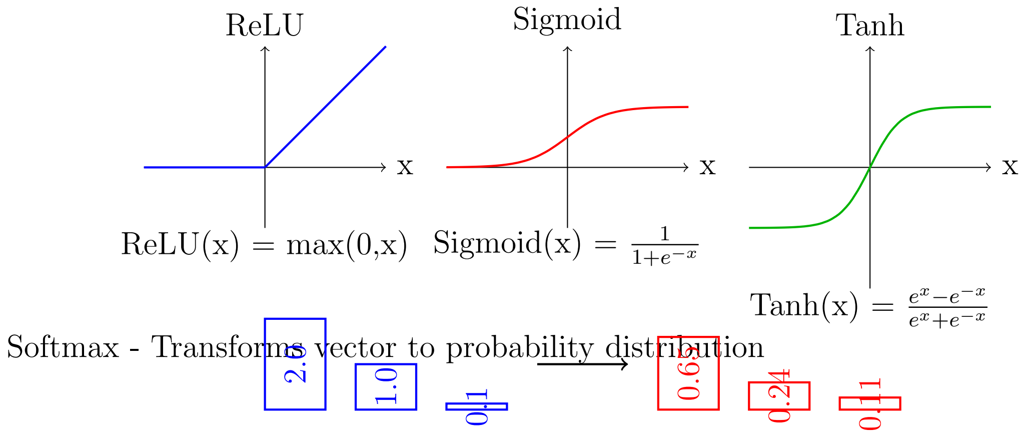


Figure 2: Visual comparison of different activation functions

32.11 Visual Comparison of Activation Functions

32.12 Quick Reference Card

Function	**Formula**	**Range**	**Best For**
ReLU	$\max(0, x)$	$[0, \infty)$	Hidden layers in deep networks
Sigmoid	$\frac{1}{1+e^{-x}}$	$(0, 1)$	Binary classification output
Softmax	$\frac{e^{z_i}}{\sum e^{z_j}}$	$(0, 1)$, sum=1	Multi-class output
Tanh	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$(-1, 1)$	Hidden layers (centered data)
Leaky ReLU	$\max(0.01x, x)$	$(-\infty, \infty)$	Avoiding dead neurons
Linear	x	$(-\infty, \infty)$	Regression output

32.13 Common Misconception

Misconception: ”All activation functions convert to probabilities”
False
Only Sigmoid and Softmax in output layer do probability conversion
Others add non-linearity for learning complex patterns

32.14 Key Takeaways

1. **ReLU** → Hidden layers, adds non-linearity, fast training
2. **Sigmoid** → Binary output, gives probability between 0-1
3. **Softmax** → Multi-class output, gives probability distribution
4. **Tanh** → Hidden layers, zero-centered alternative to sigmoid
5. **Hidden layers** → Transform data into useful representations
6. **Output layer** → Makes final decision with appropriate activation

32.15 If You Want to Explore Further

I can next explain:

- Why ReLU works better than Sigmoid in deep networks
- What happens if we remove activation functions
- Why Softmax is better than normalizing manually
- Graphical intuition of these functions
- When to use which activation function

32.16 Technical Note: Mathematical Properties

ReLU:

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{42}$$

Sigmoid:

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x)) \tag{43}$$

$$\text{where } \sigma(x) = \frac{1}{1 + e^{-x}} \tag{44}$$

Softmax:

$$\frac{\partial \text{Softmax}(z_i)}{\partial z_j} = \begin{cases} p_i(1 - p_i) & \text{if } i = j \\ -p_i p_j & \text{if } i \neq j \end{cases} \tag{45}$$

These derivatives are crucial for backpropagation during training.

The key takeaway is that multi-layer networks with hidden layers provide the computational power needed for complex, non-linearly separable problems — from routing logistics to logical operations like XOR.

Conclusion

These notes demonstrate how each machine learning algorithm has a distinct mathematical foundation while sharing common themes of optimization, regularization, and representation learning. The examples show practical applications in route planning and optimization.