

Architecting Intelligence

Q.1.

① Regression is the process of modeling the relationship between input features and a target variable by fitting a curve (like a line/hyperplane) that best represents the data trend. We minimize squared error because it penalizes larger errors more heavily (squaring amplifies big differences) and because the squared function is convex and differentiable, making it mathematically easy to find a unique global minimum.

② Starting with the vector form of the cost function:

$$\begin{aligned} J(\beta) &= \frac{1}{2} (\mathbf{y} - \mathbf{x}\beta)^T (\mathbf{y} - \mathbf{x}\beta) \\ &= \frac{1}{2} (\mathbf{y}^T - \beta^T \mathbf{x}^T)(\mathbf{y} - \mathbf{x}\beta) \\ &= \frac{1}{2} (\mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{x}\beta - \beta^T \mathbf{x}^T \mathbf{y} + \beta^T \mathbf{x}^T \mathbf{x}\beta) \end{aligned}$$

Since $\mathbf{y}^T \mathbf{x}\beta$ is a scalar, it is equal to its transpose $\beta^T \mathbf{x}^T \mathbf{y}$, we can combine the middle terms \rightarrow

$$J(\beta) = \frac{1}{2} (\mathbf{y}^T \mathbf{y} - 2\beta^T \mathbf{x}^T \mathbf{y} + \beta^T \mathbf{x}^T \mathbf{x}\beta)$$

Take the gradient w.r.t β and set it to zero.

$$\nabla_{\beta} J(\beta) = -\mathbf{x}^T \mathbf{y} + (\mathbf{x}^T \mathbf{x}) \beta = 0$$

$$\Rightarrow (\mathbf{x}^T \mathbf{x}) \beta = \mathbf{x}^T \mathbf{y}$$

$$\Rightarrow \boxed{\beta = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y}}$$

c) Calculating the inverse of $X^T X$ is computationally expensive, with a complexity of roughly $O(d^3)$. If the dimension d (number of features) is large, this becomes incredibly slow. Additionally if features are redundant (multicollinearity), the matrix $X^T X$ can be singular or close to singular, making inversion numerically unstable. Iterative methods like gradient descent are preferred because they gradually converge to the solution without needing to compute a computationally heavy matrix inverse.

Q.2

a) Backpropagation is an efficient method for calculating the gradient of the loss function with respect to every weight in the neural network. The core idea is to propagate the error signal (gradient) from the output layer backward to the input layer. It relies on "Chain Rule" of calculus.

b) $z_1 = w_1 x + b_1 ; \quad a_1 = \sigma(z_1)$
 $z_2 = w_2 a_1 + b_2 ; \quad a_2 = \sigma(z_2) = \hat{y}$ [Note: $\sigma(z) = \frac{1}{1+e^{-z}}$]
BCE loss: $L = -(y \log a_2 + (1-y) \log(1-a_2))$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} = \left(-\frac{y}{a_2} + \frac{1-y}{1-a_2} \right) a_2(1-a_2) \cdot a_1$$

$$= a_1(a_2 - y)$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_2} = (a_2 - y)$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} = (a_2 - y) w_2 \cdot a_1(1-a_1) x$$

$$= (a_2 - y) w_2 a_1 (1-a_1) x$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial b_1} = (a_2 - y) w_2 a_1 (1-a_1)$$

⑤ Iterative update & learning rate:-

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial L}{\partial W}$$

$$b_{\text{new}} = b_{\text{old}} - \eta \frac{\partial L}{\partial b} ; \quad \eta \rightarrow \text{learning rate}$$

Role of learning rate: The learning rate is a hyperparameter that controls the step size of the update.

- if η is too small, the model will learn very slowly and may get stuck in local minima.
- if η is too large, the updates may be too drastic causing the loss to oscillate or diverge instead of converging.

Q.3)

(a) Processing Inputs:-

- ANN: processes inputs as independent, fixed-size vectors. It has no internal memory of previous inputs.
- RNN: processes inputs sequentially. It maintains a hidden state that acts as memory, carrying information from previous time steps to influence the current output.

(b) Simple RNNs and Long-Term Dependencies:-

Simple RNNs struggle due to the vanishing gradient problem. As gradients are backpropagated through many timesteps, they are repeatedly multiplied by weight matrices (often < 1). This causes the gradient signal to decay exponentially, making it ~~possible~~ impossible for the network to learn relationships between distant inputs.

c) Role of Gates in LSTMs :- LSTMs utilize three gates (Forget, Input, Output) controlled by sigmoid activations. These gates regulate the flow of information, explicitly deciding what to keep, update, or discard from the cell state. This mechanism preserves relevant information over long sequences.

d) Addressing Vanishing Gradient :- LSTMs introduce a separate Cell State (C_t) with an additive update rule ($C_t = f_t C_{t-1} + i_t \tilde{C}_t$). This additive path (often called the constant error carousel) allows gradients to flow backward through time without being multiplied by a decaying factor at every step, effectively solving the vanishing gradient problem.

e) Example :-

- ANN :- Image Classification (e.g. → MNIST digits)
- RNN :- Short-term Weather Forecasting or Simple text generation.
- LSTM :- Machine translation (e.g. → Google Translate) or Speech Recognition.

Q.4) a) Long-Range Dependency Example :-

→ Example :- Consider the sentence : "The pilots, who had been flying for over ten hours without a break, were exhausted." The verb "were" must agree with the plural subject "pilots", despite the long intervening clause.

→ Standard RNN struggle :- Yes, a standard RNN would likely struggle. Due to the vanishing gradient problem, the signal from the subject "pilots" would decay

exponentially as it propagates through the long sequence, causing the network to lose track of the plurality by the time it needs to predict "were"

⑥ Memory Cell Intuition & Forget Gate Scenario:-

→ Intuition:- The gates act like physical valves.

The Input Gate decides what new info to add to the machine.

The Forget Gate decides what old info to remove.

Because the update to the cell state is additive (info is added to the stream rather than multiplied), gradients can flow backward without vanishing, allowing the network to retain specific data.

→ Machine Translation Scenario (Forget Gate ≈ 0):-

Consider a model translating a paragraph with multiple sentences. When the model encounters the full stop at the end of the 1st sentence, the forget gate needs to be fully active (close to 0).

This "resets" the cell state, clearing the context of the completed sentence so that it doesn't incorrectly influence the translation of the independent second sentence.