Architecture Intelligence

Assignment 1

~ Azaad Kadiyan
(240249)

1.

Q1)

(a) Regression is basically a method to find a relation b/w input variables and output. In linear regression, we assume that this relationship is linear. So, we try to draw the best possible straight line that fits the data points.

for every input x_i, output predicted is $\hat{y}_i$. The difference b/w actual output y_i & predicted $\hat{y}_i$ is called error. Our aim is to choose parameters β such that this error is smallest.

We minimise the squared error because :—

(i) Squaring ensures all errors are positive and +ve & -ve errors don't cancel out.

(ii) Squared error leads to a smooth & differentiable cost function

(iii) It pushes model to avoid big mistakes because, since errors are squared, larger errors are penalised more than smaller ones.

Q1]

(b) Derivation of normal equation :

Least-squares cost function :-

$$J(\beta) = \frac{1}{2} \|y - X\beta\|^2$$

for any vector $\bar{v}$, $\|\bar{v}\|^2 = \bar{v}^T\bar{v}$

$$\Rightarrow J(\beta) = \frac{1}{2}(y - X\beta)^T(y - X\beta)$$

$$\Rightarrow J(\beta) = \frac{1}{2}\left[y^Ty - y^TX\beta - (X\beta)^Ty + (X\beta)^T(X\beta)\right]$$

$\beta^TX^Ty = y^TX\beta$; $\beta^TX^TX\beta$

$$\Rightarrow J(\beta) = \frac{1}{2}\left[y^Ty - 2\beta^TX^Ty + \beta^TX^TX\beta\right]$$

Now, $\frac{\partial J(\beta)}{\partial \beta} = ?$

we know, $\frac{\partial}{\partial \beta}(\beta^TX^Ty) = X^Ty$

& $\frac{\partial}{\partial \beta}(\beta^TA\beta) = 2A\beta$

$$\Rightarrow \frac{\partial J}{\partial \beta} = \frac{1}{2}\left[-2X^Ty + 2X^TX\beta\right]$$

$$\Rightarrow \frac{\partial J}{\partial \beta} = X^T(X\beta - y)$$

Setting gradient to 0,

$$\Rightarrow X^TX\beta = X^Ty$$

Assuming X^TX is invertible, multiply both sides by $(X^TX)^{-1}$

$$\Rightarrow \boxed{\beta = (X^TX)^{-1}X^Ty}$$

↳ normal equation.

Q1)

(c) Direct inversion of X^TX can cause problems in situations because →

(i) High dimensional data : when no. of features is large, X^TX becomes very large matrix. Inverting such a large matrix is computationally expensive and takes lot of time.

(ii) Instability : If features are highly correlated, X^TX can become singular in which case, the inverse either does not exist or is very unstable.

Because of these issues, iterative methods like gradient descent are preferred →

→ It does not require inversion of matrix

→ works decently with large datasets also

→ is memory efficient.

Q2)

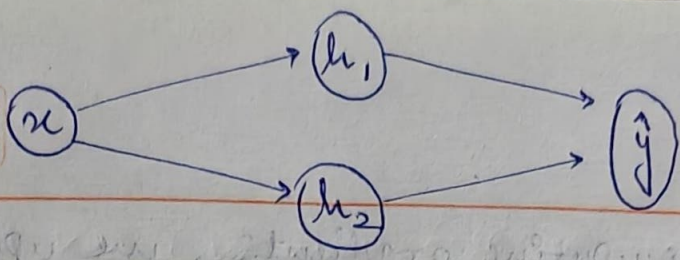
(a) Core Idea behind Backpropagation

We use the method of backpropagation to train a neural network. Main idea behind it → After network makes a prediction, we check how wrong that prediction is using a loss function and then we adjust the weights in a way that reduces this loss.

Now, the problem is that output depends on many weights. Because of this, we can't directly see how changing one weight affects the final loss. Chain rule helps resolve this issue.

Using chain rule, we can break complicated derivative into smaller ones. Backpropagation applies chain rule from output layer backwards by computing gradients layer by layer.

In this way, backpropagation tells how much each weight and bias contributed to the error and how to update them.



Q2)

(b)

Given: $z_1 = w_1 x + b_1$, $a_1 = \sigma(z_1)$

$z_2 = w_2 a_1 + b_2$, $a_2 = \sigma(z_2) = \hat{y}$

Loss function (BCE) $\Rightarrow L = -(y \log(a_2) + (1-y)\log(1-a_2))$

$$\frac{\partial L}{\partial z_2} = a_2 - y$$

(i) $\frac{\partial L}{\partial w_2} = ?$

chain rule

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} = \boxed{(a_2 - y)(a_1)}$$

(ii) $$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_2} = \boxed{(a_2 - y) \cdot (1)}$$

(iii) $$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

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

$$= \boxed{w_2 \cdot x \cdot a_1 \cdot (a_2 - y) \cdot (1 - a_1)}$$

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

$\frac{\partial \sigma}{\partial z} = \sigma(1-\sigma)$

(iv) $$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1}$$ ($\frac{\partial z_1}{\partial b_1} = 1$)

$$= \boxed{(a_2 - y) \cdot w_2 \cdot a_1 (1 - a_1)}$$

Q2)

(c) After computing gradients, we update parameters using gradient descent :

$$w = w - \eta \frac{\partial L}{\partial w} \quad \text{and} \quad b = b - \eta \frac{\partial L}{\partial b}$$

for each parameter,

$$w_2 \longrightarrow w_2 - \eta(a_2 - y)a_1$$
$$b_2 \longrightarrow b_2 - \eta(a_2 - y)$$
$$w_1 \longrightarrow w_1 - \eta(a_2 - y)w_2 a_1(1 - a_1)x$$
$$b_1 \longrightarrow b_1 - \eta(a_2 - y)w_2 a_1(1 - a_1)$$

Learning rate (η) controls how big the update steps are, basically how fast the model learns.
If η is too small, training becomes slow
If η is too large, training becomes unstable.

Q3)

(a) Artificial Neural Networks process inputs independantly. Each input is separate data point and network has no memory of previous inputs.

Recurrent Neural Network handles sequences. It processes inputs one step at a time and maintain a hidden state that carries info from previous steps. In this way, it remembers past inputs.

(b) During backpropagation, gradients become very small as they pass through many steps because of which, network fails to update weights associated with earlier inputs. Hence, simple RNNs tend to focus on recent inputs & struggle to learn rel^n on info from long ago.

(c) LSTMs introduce a memory cell along with 3 gates, input gate, forget gate, and output gate. Input gate decides what new info should be stored. forget gate decides what old info should be removed. Output gate decides what part of memory should be used to produce output. In this way, gates help preserve important info for longer durations.

(d) LSTMs allow info to flow through memory cell with minimal changes. Since memory cell can pass values almost unchanged, gradients do not shrink during backpropagation quickly. This prevents them from vanishing and allows LSTMs to learn long-term dependencies.

(e) One example task

ANN → predicting house prices using fixed features like area, no. of rooms, location, etc.

RNN → predicting next value in a time series where history matters

LSTM → language translation, where understanding a word may depend on content from earlier in the sentence.

Q4)

(a) In the sentence :

"The train that I took yesterday from Mumbai was delayed by 2 hours."

To choose verb "was", model needs to remember that subject is "train", not "Mumbai". Since, subject & verb are separated by many words, this is a long-range dependency.

Yes, a standard RNN would struggle to model this correctly because information about subject may get lost as sentence becomes longer. Due to vanishing gradients, RNN may fail to connect verb to correct subject.

Q4)

(b) An LSTM has a memory cell to store important info for long time periods. It has 3 gates :

(i) Input gate decides what new info should be stored

(ii) forget gate decides what old info should be removed

(iii) Output gate decides what info should be passed on to the next step.

Example in Machine Translation :
When translating a long sentence, gender or number of a noun introduced earlier in the sentence must be remembered until verb appears later. The intermediate irrelevant words can be removed, so forget gate should be close to zero for them. This ensures that important grammatical info is retained & correct translation is produced.