1. Questions

3.

(a) ANN has no idea of a sequence. It takes input one by one and does not store information about the past. In RNNs input comes as a stream/sequence, it processes them step by step and it also remembers what it saw in the past.

(b) RNNs struggle with long term dependencies because of the issue of vanishing Gradients.,

(c) LSTM's uses gates [forget gate, input gate, output gate] and using these information can flow unchanged.

(d) RNNs repeatedly multiply gradients by a no. $((\eta) < 1)$ so they shrink with time/more inputs. LSTM's using gates have additive updates

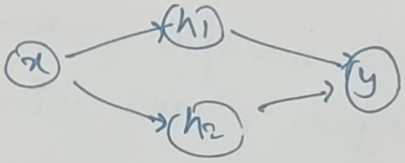
(e) ANN, simple prediction — LSTM's talking bots (small scale)
RNN, sentence completion

4. (a) The content of the word. Eg, the use of the word model can mean differently in diff contexts, so long term dependency is reqd.
NO, RNNs struggle for long range dependency, LSTMs are useful.

(b) LSTMs have a cell state that flows forward almost unchanged
it has Gates, which learn controlled values
1. Forget Gate : if the old info still relevant, if not, forget it.
2- Input Gate : is the new info to be remembered or not?
3 Output Gate : Does the remembered content need to be applied while generating the next output or not?

forget gate 0 ⟶ erase memory
when the content of a word changes in the sentence/text it should forget the old one and remember the new one.

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



(a) The idea behind Backprop is to compute the gradients efficiently, and then update the previous weight and biases, so as to reduce the cost function, and make the predictions closer to the actual outputs.

Each parameter affects the other one in a link, so being able to apply chain rule helps us to model all these dependencies.

(b) $\frac{\partial L}{\partial w_2}$, using chain rule $\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$

$z_2 = w_2 a_1 + b_2$

$z_1 = w_1 x + b_1$

$a_1 = \sigma(z_1)$

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

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

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

$$\Rightarrow (a_2 - y)(w_2)(a_1)(1 - a_1)(x)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1}$$

$$\Rightarrow (a_2 - y)(w_2)(a_1)(1 - a_1)(1)$$

(c) learning rate works on any parameter θ as $\theta := \theta - \alpha \frac{\partial L}{\partial \theta}$

we will update each parameter the same way

→ learning rate controls the ~~growth of the steps~~ convergence time of the algorithm, speed and efficiency of the algo. It shouldn't be too large or ~~of~~ too small.

1. $\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} \cdots \beta_d x_{id}$

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & \cdots & & x_{2d} \\ \vdots & & & & \\ 1 & x_{n1} & \cdots & & x_{nd} \end{bmatrix} \qquad \hat{y} = \vec{X}\vec{\beta}$$

$$J(\beta) = \frac{1}{2}\|y - X\beta\|^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

(a) Regression is just trying to find the best values of the parameters and fitting a better and better straight line to the data. So that diff b/w ~~the expectation and the~~ prediction and the actual output is as close as possible.

squared error is a good method, because it penalises large diff. in prediction even more and also makes -ve and +ve errors both positive because both are equally bad. and it also has a single global ~~min~~ min. So the local min is the global

(b) $\hat{y} = X\beta$ error $= y - X\beta$

$$J(\beta) = \frac{1}{2}\|y - X\beta\|^2 = \frac{1}{2}(y - X\beta)^T(y - X\beta) \quad [\because \|A\|^2 = A^TA]$$

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

$\nabla_\beta(\beta^TA\beta) = 2A\beta$

$\nabla_\beta(\beta^Tc) = c$

take grad. wrt β

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

$$= \frac{1}{2}\left[0 - 2X^Ty + 2X^TX\beta\right]$$

$$= \boxed{X^TX\beta - X^Ty}$$

setting it to 0

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

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

(c) matrix inversion is roughly of the order of $O(n^3)$

and if the data size is large (>1000) then doing the matrix computation and inversion will be very time-heavy. so it is not used for large datasets. Iterative methods are good because they scale to large data also and are bound to converge if the correct learning rate is chosen.