1. We have been given input features as x_i and the output as y_i

a) Linear regression assumes that the linear relationship $\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_n x_{in}$ is the solution. In matrix form $\hat{y} = X\beta$ ~~and~~ and linear regression is taking the error function as the least squares cost function. Since this represents the error, it is best to minimise it. This fact is used by it to learn the pattern and get the best values of β parameters so that the prediction on an ~~the~~ unseen input x_j is best. $\hat{y}_j$

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

$y \to n\times 1$
$X \to n\times(d+1)$
$\beta \to (d+1)\times 1$

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

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

$\leftarrow$ Since $y^T X\beta$ is a scalar
$y^T X\beta = (y^T X\beta)^T = \beta^T X^T y$

So we get $J(\beta) = \frac{1}{2}(y^T y - 2\beta^T X^T y + \beta^T X^T X\beta)$

take gradient wrt $\beta \Rightarrow \frac{1}{2}(0 - 2X^T y + 2X^T X\beta)$

minimising so setting gradient to zero gives $X^T X\beta = X^T y$

Normal equation $\Rightarrow \boxed{\beta = (X^T X)^{-1} X^T y}$

c) We know calculating the inverse is a long and complex method and for higher dimensions it just becomes more longer and complex and hence computationally expensive. Iterative methods prevents finding inverse of a matrix and is hence preferred over solving the normal equation directly.

② ⓐ Backpropagation involves using the chain rule again and again to calculate the gradients of the loss function with respect to different parameters in a neural network. It offers efficient ~~composition~~ computation because of the structure and functions used in the neural network. Its just composition of functions (linear or activation) and hence chain rule means just their products. Also the method is to first do a forward pass to compute activations and then a backward pass to get the error term (gradient) so each parameter's gradient is simply the product of the error term and input to that parameter.

ⓑ input x, $z_1 = w_1 x + b_1$, $a_1 = \sigma(z_1) = \frac{1}{1+e^{-z_1}}$, output $z_2 = w_2 a_1 + b_2$

$a_2 = \sigma(z_2) = \hat{y}$

given loss $L = -(y \log(a_2) + (1-y)\log(1-a_2))$ and $\frac{\partial L}{\partial z_2} = a_2 - y$

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

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

note : if $a = \frac{1}{1+e^{-k}}$

$\frac{da}{dk} = a(1-a)$

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

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

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

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

ⓒ $w_1' = w_1 - \alpha \frac{\partial L}{\partial w_1}$ $b_1' = b_1 - \alpha \frac{\partial L}{\partial b_1}$

$w_2' = w_2 - \alpha \frac{\partial L}{\partial w_2}$ $b_2' = b_2 - \alpha \frac{\partial L}{\partial b_2}$

update like this

[$\alpha \rightarrow$ Learning rate]

$\alpha \rightarrow$ determines the size of step taken each time it updates.

Too large = overshoots

Too small = slow / may get stuck.

③ ⓐ ANN takes all input at once and each input is treated independently but in case of RNN instead of taking the whole sequence at once, it takes them step by step and they are not treated independently by maintaining a hidden state that acts kind of like "memory".

ⓑ Backpropagation may involve many steps so the gradient may decrease exponentially and since these gradients are responsible for the relation between sequences, for long term dependencies the gradient is too small hence a problem. This is known popularly as vanishing gradient problem.

ⓒ The above problem is solved by LSTM using forget, input and output gates. These gates control the flow of information among multiple layers and selectively retain crucial information and forget unnecessary useless info.

ⓓ Well, the forget gate can be set to nearly 1 allowing gradients to pass through multiple layers unchanged during backpropagation.

ⓔ ANN → product recommendation in e-commerce sites based on history.
RNN → NLP (Predicting next word in short sentence)
LSTM → NLP (Fill in the blank involving large sentences or paragraphs too)

④ I lived in Ireland, so at school they made me learn how to speak ____ ?
ⓐ long term dependency

Yes, I think RNN would struggle (if we keep window size small)

ⓑ Memory cell is the cell state that contains the ~~info~~ information in flow which is being passed on ~~[illegible]~~ and gating mechanism has the ability to add input information as well as remove unnecessary info from this cell state. So any info that the gates think is important, is retained in the ~~[illegible]~~ cell state over long sequences.
Setting forget gate to zero means you don't ~~h~~ want to remember anything from before for the next prediction. An example/case I can think of is when you have two completely different statements together we want forget gate to become zero at the full stop.