1) The universal approximation theorem states that a feedforward neural network (NN) with a single hidden layer can approximate any function over some compact set, provided that it has enough neurons on that layer. This suggests that the number of neurons is more important than the number of layers. But in practice deep learning is obviously very successful at various prediction tasks. Why is that? Shouldn't all deep NNs be equivalent to single layered NNs with enough neurons? Why do we need depth when we could theoretically rewrite that neural network with a single layer?

Although you can approximate the function, there will be many possibilities. It is harder for your model to learn it. Deep neural networks are good at learning different features because the inner layers can spot them. Each layer ends up learning a different features based on the previous layer. With one shallow and big network, you will have to learn those features directly from the input, which is possible but very hard.

2) Here are a few types of gradient descent updates:

**Stochastic Gradient Descent:** Uses only single training example to calculate the gradient and update parameters.

**Batch Gradient Descent:** Calculate the gradients for the whole dataset and perform just one update at each iteration.

**Mini-batch Gradient Descent:** Mini-batch gradient is a variation of stochastic gradient descent where instead of single training example, mini-batch of samples is used. It’s one of the most popular optimization algorithms.

What are the pros and cons of each one and when should you use them? Walk us through your intuition and give us examples of when you are going to use what update.

**Stochastic gradient descent:** noisy/inaccurate, but very cheap, so allows to perform a lot of iterations. It is sometimes better to perform many small inaccurate updates than few accurate gradient updates.

**Batch gradient descent:** accurate gradient computation, but expensive. Since each update is expensive, learning rate cannot be very small (otherwise would be too computationally expensive). So one sometimes has to perform fewer updates with a larger learning rate. Therefore, the optimal point of the cost function may be missed.

**Mini-batch gradient descent:** medium between stochastic and batch gradient descent.

3) In neural networks, what is the role of the activation function and why do we need it?

*The activation function acts a nonlinear function and helps us avoid the lineairties.*

4) Our internal AI team has been training a model for very long and our biggest problem is that our model does not learn anything. Our cost function is not going down, and our gradients in a layer seem to be the same. We have been debugging our gradient update function but it seems like it is correct. Do you have any tips to help us solve this problem? Walk us through your thought process and show us on the board what is going on.

*questions 5 and 6 are repeated from last section*
This is because the initialization might have been the same. As a result we will not break the symmetry and the neural networks.

5) Consider the dataset: $D = \{x_{(1)}, \ldots, x_{(100)}\}$ with $x$ in $\mathbb{R}^3$. The problem is a 3-class classification problem. Consider a neural network architecture with 2 hidden layers of dimension 4 and 5 - using a sigmoid and a softmax respectively.

   a) How would you represent graphically this neural network?

   For a single input vector example $x = (x_1, x_2, x_3)$, (we do not consider the $x_0$ set as 1 - which was previously done to account for the bias), we can represent the neural network the following way:

   b) What are the feedforward equations for this neural network?

   For a single training example $x$ in $\mathbb{R}^3$ (here, we do not consider the $x_0$ set as 1 previously used to account for the bias), so $x$ is $(x_1, x_2, x_3)$ and we use a vector $b$ to represent the bias terms.

   
   
   $z_1 = W_1 x + b_1$ \hspace{1cm} (1) \hspace{1cm} $W_1 : (4,3)$ $b_1 : (4,1)$

   $a_1 = g(z_1)$ \hspace{1cm} (2)

   $z_2 = W_2 a_1 + b_2$ \hspace{1cm} (3) \hspace{1cm} $W_2 : (5,4)$ $b_2 : (5,1)$

   $a_2 = g(z_2)$ \hspace{1cm} (4)

   $z_3 = W_3 a_2 + b_3$ \hspace{1cm} (5) \hspace{1cm} $W_3 : (3,5)$ $b_3 : (3,1)$

   $h = \text{softmax}(z_3)$ \hspace{1cm} (6) \hspace{1cm} with $\text{softmax}(v) = (e^{v_1}, \ldots, e^{v_K})$

   Similar notations can be adopted with a matrix $\Theta$ that incorporates the bias, and $x$ incorporating a component equal to 1.
c) Describe the relationship between the graphical representation and the feedforward equations. What do the nodes represent? What do the edges represent?

- 1 node = 1 component of a vector (x or activation)
- 1 edge = 1 vector component * 1 matrix weight
  - Edges that point to same node mean that the scalars they represent are summed together to obtain the value of the node

d) What is the total number of parameters in this neural network?

Number of weights per equation:

1. $4 \times 3 + 4 = 16$
2. 0
3. $5 \times 4 + 5 = 25$
4. 0
5. $3 \times 5 + 3 = 18$
6. 0

Total = 59 parameters

6) Why should features be normalized when applying regularization to logistic regression?*

Consider a simple example: you try to predict height in meters from leg length $x_1$ in meters and arm length $x_2$ in centimeters

**Model:** $y = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

*Where the true hypothesis is*

$y = 1 \times x_1 + (\frac{1}{100}) \times x_2$

*If we regularize those weights, $\theta_1$ will be heavily regularized compared to $\theta_2$ because its value is 100x higher. However this value difference is just due to a difference in units.*

*Note that even if variables are measured in the same units, their orders of magnitude can be different, so normalization still must be applied.*

*Remark: in the height example, you should notice that the two variables considered are correlated (leg length and arm length), so in all rigor those two variables should not be both used as input of a linear regression model.*