1. Explain the Activation Functions in your own language

1. Sigmoid

To modify the code in part (a) to use the sigmoid activation function, you can replace the activation argument in the output layer with "sigmoid":

1. Tanh  
   To modify the code in part (a) to use the hyperbolic tangent (tanh) activation function, you can replace the activation argument in the hidden layer with "tanh":
2. ReLU

To modify the code in part (a) to use the Rectified Linear Unit (ReLU) activation function, you can replace the activation argument in both the hidden layer and the output layer with "relu":

1. ELU  
   To modify the code in part (a) to use the Exponential Linear Unit (ELU) activation function, you can replace the activation argument in both the hidden layer and the output layer with "elu":
2. LeakyReLU

To modify the code in part (a) to use the Leaky ReLU activation function, you can replace the activation argument in both the hidden layer and the output layer with "LeakyReLU":

1. Swish

To modify the code in part (a) to use the Swish activation function, you can replace the activation argument in both the hidden layer and the output layer with "swish":

2. What happens when you increase or decrease the optimizer learning rate?

When you increase the learning rate of an optimizer, it can lead to faster convergence during training. This is because a higher learning rate allows the model to take larger steps in the parameter space, potentially reaching the optimal solution more quickly. However, if the learning rate is set too high, it can cause the training process to become unstable or even diverge, leading to poor performance or failure to converge.

On the other hand, when you decrease the learning rate, it slows down the convergence during training. This can be beneficial in situations where the model is oscillating or overshooting the optimal solution. A lower learning rate allows for smaller steps in the parameter space, which can help the model settle into a more optimal region. However, setting the learning rate too low can cause the training process to become very slow, especially in deep neural networks, where it may take a long time to converge or even get stuck in a suboptimal solution.

3. What happens when you increase the number of internal hidden neurons?

When you increase the number of internal hidden neurons in a neural network, it can potentially lead to increased model capacity and complexity. This increase in capacity allows the network to learn more complex and intricate patterns in the data, potentially improving its ability to represent and generalize from the training data.

Increasing the number of hidden neurons can help the model learn more fine-grained and nuanced representations of the input features. This can be especially useful in tasks where the data has high complexity or requires capturing intricate relationships. By increasing the number of hidden neurons, the model has more capacity to capture and represent these complex patterns.

4. What happens when you increase the size of batch computation?

When you increase the size of batch computation, it has several effects on the training process of a neural network:

1. Increased Training Speed: Larger batch sizes can lead to faster training times. This is because computations can be parallelized and optimized on modern hardware, such as GPUs, which are designed to process large batches efficiently. With larger batch sizes, the network can process more data in parallel, resulting in faster training iterations.
2. More Stable Gradient Estimates: With larger batch sizes, the gradient estimates used for updating the model parameters become more stable. This is because the gradients are averaged over a larger number of training examples, reducing the influence of individual noisy or outlier samples. The increased stability can help the training process converge faster and lead to better generalization.
3. Increased Memory Usage: Larger batch sizes require more memory to store the intermediate activations, gradients, and model parameters during training. This can become a limitation, especially when working with limited GPU memory or large-scale models. If the batch size is too large, it may lead to out-of-memory errors and prevent successful training.
4. Potential Decrease in Generalization: Increasing the batch size can sometimes lead to a decrease in generalization performance. This is because smaller batch sizes introduce more stochasticity in the gradient estimates, which can act as a regularization mechanism and prevent overfitting. With larger batch sizes, the network may become more prone to overfitting, particularly if the dataset is small or the model has a high capacity.
5. Why we adopt regularization to avoid overfitting?

Regularization is adopted to avoid overfitting in machine learning models. Overfitting occurs when a model learns to perform well on the training data but fails to generalize well to unseen data. Regularization techniques introduce additional constraints on the model during training to prevent it from excessively fitting the training data. Here are some reasons why regularization is used to address overfitting:

1. Model Complexity Control: Regularization helps control the complexity of a model by discouraging it from learning overly intricate and detailed patterns in the training data. By adding regularization terms to the loss function, the model is penalized for complex or large parameter values, encouraging it to favor simpler and more generalizable solutions.
2. Prevention of Overly Sensitive Models: Overfitting often occurs when a model becomes too sensitive to small fluctuations or noise in the training data. Regularization techniques, such as L1 or L2 regularization, add penalties to the loss function that encourage the model to minimize the impact of individual training examples or features, reducing sensitivity to noise.
3. Implicit Feature Selection: Regularization can act as a form of implicit feature selection, encouraging the model to focus on the most informative and relevant features for the task at hand. By penalizing the weights associated with less important features, regularization guides the model to prioritize the most discriminative features, reducing the risk of overfitting caused by irrelevant or redundant features.
4. Improved Generalization: Regularization helps improve the generalization performance of a model by reducing its reliance on specific patterns observed in the training data. By encouraging simpler and smoother solutions, regularization enables the model to capture the underlying patterns and trends in the data, rather than memorizing individual training examples.
5. Addressing Limited Training Data: Regularization is particularly important when working with limited training data. In such cases, overfitting becomes more likely as the model tries to fit noise or outliers in the data. Regularization techniques provide a form of regularization bias, biasing the model towards more reasonable and plausible solutions given the limited amount of training information
6. What are loss and cost functions in deep learning?  
   In deep learning, the terms "loss function" and "cost function" refer to the same concept, although they may be used interchangeably. They both represent a measure of how well a machine learning model is performing on a given task or problem.

A loss function quantifies the discrepancy between the predicted outputs of a model and the true labels or target values associated with the input data. It calculates a single scalar value that represents the error or loss incurred by the model for a particular set of parameters or weights.

The cost function, on the other hand, is a broader term that encompasses the loss function along with additional regularization terms or penalties, if applicable. The cost function represents the overall objective that the model aims to minimize during the training process. It combines the loss function with any regularization terms to compute the total cost or error associated with the model's predictions.

1. What do ou mean by underfitting in neural networks?

Underfitting in neural networks refers to a situation where the model is not able to capture the underlying patterns and relationships present in the training data, resulting in poor performance both on the training data and new, unseen data. It occurs when the model is too simple or lacks the capacity to adequately represent the complexity of the data.

In the context of neural networks, underfitting typically occurs when the model is not able to learn the underlying patterns due to insufficient model capacity or limitations in the training process. Here are some characteristics and indicators of underfitting:

1. High Bias: Underfitting is often associated with high bias, which means that the model is not able to capture the complexity of the data and is making strong assumptions or simplifications about the relationships between inputs and outputs.
2. Poor Training Performance: An underfit model performs poorly on the training data itself. It fails to fit the training data well, resulting in high training errors and low accuracy or performance metrics.
3. Poor Generalization: In addition to poor training performance, an underfit model also fails to generalize well to new, unseen data. It exhibits similar poor performance on the validation or test data, indicating that it is not able to capture the underlying patterns and make accurate predictions on unseen examples.
4. High Bias-Variance Trade-off: Underfitting occurs when the model has too much bias and not enough variance. It is often seen in cases where the model is too simple or has insufficient complexity to represent the true relationships in the data.
5. Why we use Dropout in Neural Networks?

Dropout is a regularization technique commonly used in neural networks to prevent overfitting. Overfitting occurs when a model performs well on the training data but fails to generalize well to new, unseen data. Dropout helps to address this issue by introducing randomness and reducing the co-adaptation of neurons during training.

The main idea behind dropout is to randomly "drop out" (i.e., set to zero) a proportion of the neurons in a given layer during each training step. By doing so, dropout forces the network to learn more robust and generalized features, as it prevents individual neurons from relying too heavily on specific input features or co-adapting with other neurons.