**1. Explain the Activation Functions in your own language**

1. **sigmoid**
2. **tanh**
3. **ReLU**
4. **ELU**
5. **LeakyReLU**
6. **Swish**

**Answer:-**

a) Sigmoid: The sigmoid activation function is a smooth, S-shaped curve that maps any input value to a range between 0 and 1. It is commonly used in binary classification problems as it can squash the output into a probability-like value. However, it suffers from the vanishing gradient problem, which can make training deep neural networks challenging.

b) Tanh: The tanh (hyperbolic tangent) activation function is similar to the sigmoid function but maps the input values to a range between -1 and 1. It is also a smooth curve and is often used in hidden layers of neural networks. Tanh provides stronger gradients than the sigmoid function and helps to alleviate the vanishing gradient problem to some extent.

c) ReLU (Rectified Linear Unit): ReLU is a popular activation function that replaces negative values with zero and leaves positive values unchanged. It is computationally efficient and helps mitigate the vanishing gradient problem. ReLU is widely used in deep neural networks and has been successful in many applications. However, it can suffer from the "dying ReLU" problem where some neurons become permanently inactive, leading to dead gradients.

d) ELU (Exponential Linear Unit): ELU is a modified version of ReLU that allows negative values to have a small negative output. It has a smooth curve and addresses the "dying ReLU" problem by providing non-zero gradients for negative inputs. ELU can improve the performance of deep neural networks by reducing the vanishing gradient problem.

e) LeakyReLU: LeakyReLU is another variant of the ReLU function that allows small negative values instead of completely zeroing them out. It introduces a small slope for negative values, which helps overcome the "dying ReLU" problem. LeakyReLU can improve the model's robustness and prevent dead neurons.

f) Swish: Swish is a relatively new activation function that performs a smooth interpolation between the linear and nonlinear regions. It was designed to combine the best properties of ReLU and sigmoid functions. Swish has shown promising results in some cases by providing a non-monotonic activation function that retains some of the benefits of ReLU while introducing a smooth gradient-like behavior.

Overall, the choice of activation function depends on the specific problem and the characteristics of the data. Different activation functions have their strengths and weaknesses, and selecting the appropriate one can greatly impact the performance and convergence of a neural network.

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

**Answer:-**

When you increase the learning rate of an optimizer, it leads to larger updates to the model's parameters in each iteration. This can result in faster convergence during training, as the model adjusts more quickly to the training data. However, a very high learning rate can cause the optimization process to become unstable, leading to overshooting the optimal solution and potentially diverging.

On the other hand, when you decrease the learning rate, the updates to the model's parameters become smaller in each iteration. This can help the model to fine-tune its performance and converge more accurately to the optimal solution. However, a very low learning rate may cause the training process to become slower and take more iterations to reach convergence.

It is important to find an appropriate learning rate for the specific problem and model architecture. This is often done through experimentation and tuning. Techniques such as learning rate schedules or adaptive learning rate algorithms (e.g., Adam, RMSprop) can be used to dynamically adjust the learning rate during training to strike a balance between convergence speed and stability.

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

**Answer:-**

When you increase the number of internal hidden neurons in a neural network, it allows the model to learn more complex and intricate patterns in the data. By increasing the capacity of the network, it becomes more capable of capturing intricate relationships and nuances in the input data.

Increasing the number of hidden neurons provides the model with more degrees of freedom to represent and approximate the underlying function mapping the inputs to the outputs. This can lead to improved performance and higher accuracy, especially when dealing with complex or high-dimensional datasets.

However, there are a few considerations when increasing the number of hidden neurons:

1. Overfitting: Adding too many hidden neurons can make the model prone to overfitting, where it becomes overly sensitive to noise and specific training examples, leading to poor generalization on unseen data. Regularization techniques like dropout or weight decay can help mitigate overfitting.
2. Computational Complexity: Increasing the number of hidden neurons also increases the computational requirements of training and inference. Training larger networks may require more computational resources and longer training times.
3. Training Data Availability: Increasing model complexity by adding more hidden neurons requires a sufficient amount of training data to effectively learn the parameters. Insufficient training data relative to model complexity may lead to overfitting.

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

**Answer:-**

When you increase the size of batch computation, it means that more training examples are processed simultaneously during each iteration of the training process. This has several effects:

1. Improved Efficiency: Increasing the batch size can lead to improved computational efficiency. Processing multiple examples in parallel allows for better utilization of hardware resources, such as parallel processing capabilities of modern GPUs. This can result in faster training times, especially for large-scale models and datasets.
2. Smoother Gradient Estimates: With larger batch sizes, the gradients computed during backpropagation are based on more training examples. This can lead to smoother and more stable gradient estimates, reducing the variance in parameter updates and potentially improving convergence. In some cases, larger batch sizes can help escape from poor local optima during training.
3. Regularization Effects: Using larger batch sizes can have a regularizing effect on the model. The noise introduced by smaller batch sizes, such as mini-batch stochasticity, is reduced with larger batches. This can act as a form of implicit regularization, potentially improving generalization performance and reducing overfitting.

However, there are a few considerations when increasing the batch size:

1. Increased Memory Usage: Larger batch sizes require more memory to store intermediate activations and gradients during training. This can be a concern, particularly for models with limited memory resources, such as on-device or embedded systems.
2. Learning Dynamics: Increasing the batch size can influence the learning dynamics of the model. Larger batches may result in slower convergence or make it more challenging for the model to escape sharp local minima. It may require adjusting the learning rate or other hyperparameters to maintain stable training.
3. Generalization Performance: While larger batch sizes can provide more accurate gradient estimates, they may not always generalize as well as smaller batches. Smaller batch sizes often exhibit more exploration of the parameter space and can find better generalizable solutions.

**5. Why we adopt regularization to avoid overfitting?**

**Answer:-**

Regularization is adopted to avoid overfitting in machine learning models. Overfitting occurs when a model becomes too complex and starts to memorize the training data instead of learning general patterns and relationships. As a result, the model performs well on the training data but fails to generalize well to unseen data.

Regularization techniques help to address overfitting by adding a penalty term to the loss function during training. This penalty discourages the model from excessively relying on certain features or parameters, leading to a simpler and more generalized model. Here are a few key reasons why regularization is effective in preventing overfitting:

1. Complexity Control: Regularization methods, such as L1 or L2 regularization, introduce a regularization term that penalizes large parameter values. This encourages the model to prioritize smaller weights, effectively reducing the complexity of the model. By constraining the model's complexity, regularization helps prevent it from fitting noise or irrelevant details in the data.
2. Generalization Promotion: Regularization encourages the model to find simpler and more robust patterns in the data that are likely to generalize well to unseen examples. By discouraging over-reliance on specific training examples or features, regularization promotes the discovery of more generalizable patterns and reduces the likelihood of overfitting.
3. Bias-Variance Trade-off: Regularization helps strike a balance between bias and variance in the model. A highly flexible model with few regularization constraints may have low bias but high variance, leading to overfitting. By adding regularization, the model's bias increases slightly, but its variance decreases. This trade-off can lead to better generalization performance by reducing the model's sensitivity to small variations in the training data.

**6. What are loss and cost functions in deep learning?**

**Answer:-**

In deep learning, the terms "loss function" and "cost function" refer to the same concept. They represent a mathematical function that measures the discrepancy between the predicted outputs of a neural network and the true values or labels of the training data.

The loss/cost function quantifies the error or the difference between the predicted outputs and the actual outputs. It provides a measure of how well the model is performing on the training data. The goal of training a deep learning model is to minimize this loss function, as a lower loss indicates better alignment between the predicted outputs and the true outputs.

Different types of problems (classification, regression, etc.) and neural network architectures may require different loss functions. Some commonly used loss functions in deep learning include:

1. Mean Squared Error (MSE): This loss function is commonly used for regression tasks. It calculates the average squared difference between the predicted and true values.
2. Binary Cross-Entropy: This loss function is often used for binary classification problems. It measures the dissimilarity between the predicted probabilities and the true binary labels.
3. Categorical Cross-Entropy: This loss function is used for multi-class classification problems. It quantifies the difference between the predicted class probabilities and the true class labels.
4. Sparse Categorical Cross-Entropy: This is a variation of the categorical cross-entropy loss function that is used when the true class labels are integers instead of one-hot encoded vectors.
5. Kullback-Leibler Divergence: This loss function is used in generative models, such as variational autoencoders (VAEs). It measures the difference between the predicted probability distribution and a target distribution.

**7. What do you mean by underfitting in neural networks?**

**Answer:-**

Underfitting in neural networks refers to a situation where the model is not able to capture the underlying patterns and complexities in the training data. It occurs when the model is too simple or has insufficient capacity to learn the relationships between the input features and the target variable.

When a neural network underfits the data, it means that the model's performance on both the training data and the validation/test data is poor. The model fails to generalize well and exhibits high bias. It may struggle to make accurate predictions even on the training examples, let alone unseen data.

Signs of underfitting include:

1. High training and validation errors: The model's performance is consistently poor on both the training set and the validation/test set.
2. Poor convergence: The model's loss or error does not decrease significantly during training, and the performance plateaus at a suboptimal level.
3. Oversimplified predictions: The model's predictions are overly simplistic and fail to capture the complexities present in the data.
4. Low complexity or insufficient capacity: The model may have too few layers, too few neurons per layer, or lack sophisticated features that are necessary to capture the underlying patterns.

To address underfitting, several approaches can be considered:

1. Increase model complexity: Add more layers, increase the number of neurons, or use more sophisticated architectures to allow the model to capture complex patterns in the data.
2. Collect more data: Insufficient data can contribute to underfitting, and gathering more diverse and representative examples may help the model learn better.
3. Adjust regularization: Regularization techniques such as dropout or L1/L2 regularization can be adjusted to strike a balance between reducing overfitting and preventing excessive underfitting.
4. Modify hyperparameters: Adjusting learning rate, optimizer, batch size, or activation functions can also help in addressing underfitting.

**8. Why we use Dropout in Neural Networks?**

**Answer:-**

Dropout is a regularization technique used in neural networks to prevent overfitting. Overfitting occurs when a model becomes too specialized to the training data and performs poorly on unseen data. Dropout helps mitigate this issue by reducing the reliance of individual neurons on specific input features and encourages the network to learn more robust and generalizable representations.

During training, dropout randomly sets a fraction of the neurons' outputs to zero at each update, effectively "dropping out" those neurons. This dropout is applied independently to each training example and helps to prevent complex co-adaptations between neurons. By doing so, dropout acts as a form of model averaging, where different subsets of the network's neurons are activated for each training example.

The main advantages of using dropout are:

1. Regularization: Dropout adds noise to the network during training, which reduces the model's ability to memorize the training examples. It prevents overfitting and helps improve generalization by forcing the network to learn more robust and representative features.
2. Ensemble effect: Dropout can be seen as training multiple models with different subsets of neurons. At test time, the ensemble effect is approximated by using all the neurons but scaling down their outputs by the dropout probability. This can improve the model's performance and make it more resilient to noise.
3. Computational efficiency: Dropout effectively reduces the number of computations required during training. As neurons are randomly dropped out, the network becomes sparser, leading to faster training and less computational overhead.