1. Explain the Activation Functions in your own language

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

A1. Activation functions are mathematical functions applied to the output of each neuron in a neural network. They introduce non-linearity to the neural network, allowing it to model complex relationships between inputs and outputs. Here are the explanations of some commonly used activation functions:

a) Sigmoid Activation Function: The sigmoid function takes any input and squashes it into an output value between 0 and 1. This function is widely used in binary classification problems as it can be interpreted as the probability of belonging to a particular class.

b) Tanh Activation Function: The tanh function is similar to the sigmoid function but is shifted and scaled, so it outputs values between -1 and 1. This activation function is often used in neural networks as it can produce outputs that are centered around 0, which can help with training.

c) ReLU (Rectified Linear Unit) Activation Function: The ReLU function sets all negative inputs to zero and leaves all positive inputs unchanged. It is a popular choice for activation functions due to its simplicity and efficiency in training deep neural networks.

d) ELU (Exponential Linear Unit) Activation Function: The ELU function is similar to the ReLU function but is smooth for negative inputs, which can help with the vanishing gradient problem. The ELU function is also able to produce negative outputs, unlike ReLU.

e) LeakyReLU Activation Function: The LeakyReLU function is a modified version of the ReLU function, where negative inputs are multiplied by a small constant value instead of being set to zero. This helps to prevent dead neurons that can occur when ReLU outputs zero for negative inputs.

f) Swish Activation Function: The Swish function is a relatively new activation function that has been shown to outperform ReLU and other popular activation functions in some cases. It is a smooth function that is similar to the sigmoid function but has a non-monotonic property that can help with training.

Top of Form

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

A2. When you increase the optimizer learning rate, the algorithm will take larger steps in the weight space during training, which can make it converge faster. However, if the learning rate is set too high, the algorithm may overshoot the optimal weights and fail to converge. This can cause the loss to oscillate or even increase over time.

On the other hand, when you decrease the optimizer learning rate, the algorithm will take smaller steps in the weight space during training, which can make it converge more slowly but with better accuracy. However, if the learning rate is set too low, the algorithm may converge very slowly or get stuck in a suboptimal solution.

Thus, finding the appropriate learning rate is crucial for successful training of neural networks. Various techniques, such as learning rate schedules, learning rate annealing, and adaptive learning rate methods (e.g., Adam, Adagrad, etc.), have been developed to help determine an appropriate learning rate for a given task.

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

A3. When you increase the number of internal hidden neurons in a neural network, the model's capacity to fit complex patterns and relationships in the data increases. This can improve the performance of the model by reducing bias, as it can capture more complex features of the input data. However, if the number of neurons is increased too much, the model can start to overfit the training data, resulting in poor performance on unseen data. Additionally, increasing the number of neurons also increases the model's computational complexity, which can result in longer training times and increased memory usage. Therefore, the number of hidden neurons needs to be chosen carefully, balancing the trade-off between model complexity and performance.

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

A4. Increasing the batch size in training a neural network means that the optimizer calculates the gradients using a larger number of examples before updating the weights. This has several effects on the training process:

* **Faster convergence:** Increasing the batch size can lead to faster convergence, especially for large datasets, since each parameter update is based on a more accurate estimate of the true gradient.
* **Less noise:** With larger batch sizes, the gradients become less noisy, since the gradients of the individual examples are averaged out. This can result in more stable training, especially for complex architectures.
* **More memory usage:** Larger batch sizes require more memory to store the gradients and intermediate computations, which can be an issue for limited memory systems.
* **Slower training:** Larger batch sizes may require more time per epoch since each epoch requires more computation time, especially on large datasets.

Overall, increasing the batch size can be beneficial for training deep neural networks, but it also comes with tradeoffs that must be considered, such as increased memory usage and slower training.

5. Why we adopt regularization to avoid overfitting?

A5. We adopt regularization techniques to avoid overfitting in machine learning models. Overfitting is a phenomenon where the model learns to fit the training data too well and as a result, performs poorly on new, unseen data. In other words, the model becomes too complex and starts to memorize the training data, including the noise, rather than learning the underlying patterns in the data.

Regularization helps to prevent overfitting by adding a penalty term to the cost function during training. The penalty term discourages the model from overfitting by adding a cost to the weights that are too large, which leads to a simpler model. This is achieved by adding an L1 or L2 regularization term to the loss function, which encourages the weights to be small or close to zero.

Regularization also helps to generalize the model by reducing the variance in the model. By reducing the variance, the model is less sensitive to small fluctuations in the training data and can better generalize to new, unseen data.

In summary, regularization helps to prevent overfitting by adding a penalty term to the cost function during training, which discourages the model from becoming too complex and memorizing the training data.

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

A6. In deep learning, the loss and cost functions are used to measure the difference between the predicted output and the actual output of a model. The loss function is typically defined for a single training example, while the cost function (also known as the objective function) is defined over the entire training set.

The choice of loss and cost function depends on the specific task that the model is being trained for. For example, for binary classification tasks, the binary cross-entropy loss function is commonly used. For multi-class classification tasks, the categorical cross-entropy loss function is typically used.

The goal of training a deep learning model is to minimize the cost function by adjusting the weights and biases of the model. This is typically achieved using optimization algorithms such as stochastic gradient descent (SGD) or its variants, which iteratively update the parameters of the model to reduce the value of the cost function.

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

A7. Underfitting in neural networks refers to a situation where the model is too simple or not complex enough to capture the underlying patterns or relationships in the training data. This results in a model that performs poorly not only on the training data but also on the test data. An underfit model has high bias and low variance. Some common causes of underfitting are using a model that is too simple, using a small dataset, and training for too few epochs.

8. Why we use Dropout in Neural Networks?

A8. Dropout is a regularization technique that is used to prevent overfitting in neural networks. Overfitting occurs when a model becomes too complex and starts fitting too closely to the training data, which results in poor performance on new, unseen data.

Dropout works by randomly dropping out (i.e., setting to zero) a certain percentage of the input neurons in a layer during training. This forces the remaining neurons to learn more robust and generalizable features, since they cannot rely on the presence of any one particular input neuron. Dropout also effectively reduces the capacity of the model, which helps prevent overfitting.

During inference (i.e., making predictions on new instances), all neurons are used, but their output is scaled by the dropout rate. This ensures that the expected output of each neuron during inference is the same as its average output during training, which helps prevent over-reliance on any one particular neuron.