1. Is it okay to initialize all the weights to the same value as long as that value is selected randomly using He initialization?

A1. Yes, it is generally okay to initialize all the weights to the same value as long as that value is selected randomly using He initialization. This is because He initialization ensures that the variance of the outputs of each layer is roughly equal to the variance of its inputs, which helps prevent the vanishing/exploding gradients problem during training. Initializing all the weights to the same value can also help with symmetry breaking, as the random initialization ensures that each neuron can learn to detect different features in the input data.

1. Is it okay to initialize the bias terms to 0?

A2. It is generally okay to initialize bias terms to 0 in neural networks. This is because the network can learn the appropriate biases during training, and if all biases are initialized to the same value, they will all be updated to different values during training based on the gradients of the loss with respect to each bias. However, some research has suggested that initializing bias terms to a small constant value (such as 0.1) may help to improve the convergence and performance of neural networks in certain cases. Ultimately, the best approach may depend on the specific problem and architecture being used.

1. Name three advantages of the ELU activation function over ReLU.

A3. Here are three advantages of the Exponential Linear Unit (ELU) activation function over the Rectified Linear Unit (ReLU) activation function:

1. Reduced vanishing gradient problem: ELU has a nonzero gradient for negative inputs, which helps reduce the vanishing gradient problem that can occur when using ReLU. This can lead to faster convergence and better generalization.
2. Smoothness: The ELU function is smooth everywhere, including around zero, which can improve the stability and robustness of the model.
3. Better performance: In some cases, ELU has been shown to outperform ReLU on certain tasks, particularly on large datasets with deeper architectures.
4. In which cases would you want to use each of the following activation functions: ELU, leaky ReLU (and its variants), ReLU, tanh, logistic, and softmax?

A4. Here are some use cases for each of the mentioned activation functions:

1. ELU (Exponential Linear Unit):
   * ELU can be used in most cases where ReLU is used as an activation function.
   * ELU may perform better than ReLU for some datasets since it has a continuous output and avoids dead neurons problem.
   * ELU can also prevent overfitting due to its ability to push mean activations closer to zero.
2. Leaky ReLU and its variants (e.g. Parametric ReLU, Randomized ReLU):
   * Leaky ReLU can be used instead of ReLU to avoid the dead neurons problem.
   * Parametric and Randomized ReLU can be used when you want to add more flexibility to the model by allowing the slope of the negative part to be learned by the network.
3. ReLU (Rectified Linear Unit):
   * ReLU is the most commonly used activation function, and it works well in most cases, especially for deep neural networks.
   * ReLU can be used in any network architecture that requires an activation function.
4. Tanh (Hyperbolic Tangent):
   * Tanh can be used in the output layer of a neural network when you want to predict values between -1 and 1.
   * Tanh can also be used in the hidden layers of a neural network if you want your outputs to be centered around zero.
5. Logistic (Sigmoid):
   * Logistic function can be used in the output layer of a neural network when you want to predict probabilities between 0 and 1.
   * Logistic function can also be used in the hidden layers of a neural network if you want your outputs to be centered around zero and to have a continuous output.
6. Softmax:
   * Softmax is typically used in the output layer of a neural network when you want to classify data into multiple classes.
   * Softmax normalizes the output of the neural network into a probability distribution over the classes.

Top of Form

1. What may happen if you set the momentum hyperparameter too close to 1 (e.g., 0.99999) when using a MomentumOptimizer?

A5. When setting the momentum hyperparameter too close to 1 (e.g., 0.99999), the MomentumOptimizer will mainly use the direction of the past few gradients to update the weights, almost ignoring the current gradient. This can cause the optimizer to overshoot and oscillate around the minimum, potentially leading to slow convergence or divergence. In other words, it will have a lot of momentum and will be slow to change direction, which can cause it to miss the minimum or even diverge.

1. Name three ways you can produce a sparse model.

A6. There are several ways to produce a sparse model:

1. L1 regularization: It encourages weights to be exactly 0, resulting in sparse models.
2. Dropout regularization: It randomly sets some neurons to zero during training, which can result in sparse representations in some layers.
3. Weight pruning: It involves removing the weights with the smallest absolute values, which results in a sparse model.
4. Quantization: It involves reducing the number of bits used to represent the weights and activations, which can lead to sparsity.
5. Clustering: It involves grouping weights with similar values and representing them with a single weight, which can result in a sparse model.
6. Does dropout slow down training? Does it slow down inference (i.e., making predictions on new instances)?

A7. Yes, dropout slows down training because during training, the neurons are randomly dropped out with a certain probability, so the computations have to be recalculated with fewer neurons. This means that each training step takes longer to complete. However, it does not slow down inference because during inference, all neurons are used and no dropout is applied. Therefore, inference is typically faster than training.