1. Is it OK to initialize all the weights to the same value as long as that value is selected

randomly using He initialization?

**No, it is not recommended to initialize all the weights to the same value, even when using He initialization. He initialization initializes weights with small random values drawn from a Gaussian distribution with a mean of 0. Initializing all weights to the same value would result in a lack of diversity among neurons in the same layer, and the network might not learn effectively.**

2. Is it OK to initialize the bias terms to 0?

**Yes, it is generally acceptable to initialize the bias terms to 0. Bias terms are used to shift the activation function to the left or right, allowing the network to model complex relationships. Initializing them to 0 simplifies the initialization process and does not introduce any bias in favor of any particular output.**

3. Name three advantages of the SELU activation function over ReLU.

**Self-Normalization: SELU activations can self-normalize the network, which means that the activations tend to have zero mean and unit variance during training. This helps mitigate the vanishing/exploding gradient problem and accelerates convergence.**

**Smoothness: SELU is a smooth, continuous function that is differentiable everywhere, including at zero. This ensures gradient propagation even when inputs are close to zero.**

**Consistency with Weight Initialization: SELU works well with LeCun initialization, ensuring that weights are initially set in a way that aligns with self-normalization.**

4. In which cases would you want to use each of the following activation functions: SELU, leaky

**ReLU (and its variants), ReLU, tanh, logistic, and softmax?**

**SELU: SELU is particularly useful in deep networks where self-normalization is desirable. It tends to work well when dealing with vanishing/exploding gradient issues and can lead to faster convergence.**

**Leaky ReLU (and variants): Leaky ReLU variants are a good choice when you want to mitigate the dying ReLU problem (i.e., neurons getting stuck during training). They are more resilient to dead neurons and can handle a wider range of inputs.**

**ReLU: ReLU is a popular choice for most deep learning tasks due to its simplicity and efficiency. It performs well when the network architecture and initialization are well-tuned.**

**tanh: Hyperbolic tangent (tanh) is suitable for tasks where inputs need to be centered around zero and outputs should be in the range of -1 to 1. It is often used in recurrent neural networks (RNNs).**

**Logistic (Sigmoid): The logistic (sigmoid) activation is mainly used in binary classification tasks where the output should represent a probability between 0 and 1.**

**Softmax: Softmax is used in the output layer of multi-class classification tasks to produce a probability distribution over multiple classes.**

5. What may happen if you set the momentum hyperparameter too close to 1 (e.g., 0.99999)

when using an SGD optimizer?

**If the momentum hyperparameter is set very close to 1 (e.g., 0.99999) in stochastic gradient descent (SGD), it can lead to extremely slow convergence and potential instability during training. The high momentum causes the optimizer to accumulate too much past gradient information, making the updates sluggish and causing the optimizer to have difficulty escaping local minima. It can also result in large oscillations in the weight updates.**

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

**L1 Regularization (Lasso): Applying L1 regularization to the model's weights encourages many of the weights to become exactly zero, effectively producing a sparse model.**

**Dropout: During training, dropout randomly sets a fraction of neuron activations to zero, which can lead to sparsity in the network's activations and make it more robust.**

**Pruning: Pruning involves removing certain connections or neurons from the network that have little impact on the model's performance. Pruned connections are set to zero, resulting in a sparse model.**

7. Does dropout slow down training? Does it slow down inference (i.e., making predictions on

new instances)? What about MC Dropout?

**Dropout can slow down training because, during each training step, a fraction of neurons is randomly dropped out, which increases the computational cost of forward and backward passes. However, it can help prevent overfitting and improve generalization.**

**During inference (making predictions on new instances), dropout is typically turned off, so it does not slow down inference. The model uses the full architecture for making predictions.**

**MC Dropout (Monte Carlo Dropout) involves using dropout during inference but running multiple forward passes with dropout enabled and averaging the predictions. This can increase inference time compared to standard dropout but often results in improved uncertainty estimation and model performance.**

8. Practice training a deep neural network on the CIFAR10 image dataset:

a. Build a DNN with 20 hidden layers of 100 neurons each (that’s too many, but it’s the

point of this exercise). Use He initialization and the ELU activation function.

b. Using Nadam optimization and early stopping, train the network on the CIFAR10

dataset. You can load it with keras.datasets.cifar10.load\_​data(). The dataset is

composed of 60,000 32 × 32–pixel color images (50,000 for training, 10,000 for

testing) with 10 classes, so you’ll need a softmax output layer with 10 neurons.

Remember to search for the right learning rate each time you change the model’s

architecture or hyperparameters.

c. Now try adding Batch Normalization and compare the learning curves: Is it

converging faster than before? Does it produce a better model? How does it affect

training speed?

d. Try replacing Batch Normalization with SELU, and make the necessary adjustements

to ensure the network self-normalizes (i.e., standardize the input features, use

LeCun normal initialization, make sure the DNN contains only a sequence of dense

layers, etc.).

e. Try regularizing the model with alpha dropout. Then, without retraining your model,

see if you can achieve better accuracy using MC Dropout.

**Training a deep neural network on CIFAR10 as described is a substantial task that involves writing and running code, which cannot be performed within the confines of this text-based format. However, I can provide you with a high-level outline of the steps involved:**

**Import necessary libraries, including TensorFlow or PyTorch.**

**Load the CIFAR10 dataset using the provided function.**

**Preprocess the data, including scaling pixel values and one-hot encoding labels.**

**Build a deep neural network with 20 hidden layers, each containing 100 neurons, using He initialization and ELU activation functions.**

**Use the Nadam optimizer and implement early stopping during training.**

**Train the network on the CIFAR10 dataset, searching for an appropriate learning rate.**

**Add Batch Normalization and compare learning curves.**

**Replace Batch Normalization with SELU and make necessary adjustments to ensure self-normalization.**

**Try regularizing the model with alpha dropout.**

**Optionally, experiment with MC Dropout for improved accuracy and uncertainty estimation.**

**For detailed code and execution, you would need to use a programming environment like TensorFlow or PyTorch.**