**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 OK to initialize all weights to the same value, even if the value is selected randomly. Initializing all weights to the same value can cause symmetry problems, where all neurons in a layer learn the same features during training. He initialization should be applied independently to each weight to break symmetry.

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

* Yes, it is generally OK to initialize bias terms to 0. Unlike weights, biases do not suffer from symmetry issues, and initializing them to 0 is a common practice.

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

* **Self-normalizing**: SELU (Scaled Exponential Linear Unit) ensures that the network self-normalizes, maintaining a mean of 0 and standard deviation of 1 across layers.
* **Avoids vanishing gradients**: SELU helps mitigate the vanishing gradient problem, which can occur with ReLU.
* **Better performance**: SELU often leads to faster convergence and better performance on deep networks compared to ReLU.

**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**: Use in deep networks where self-normalization is desired, especially in fully connected layers.
* **Leaky ReLU**: Use when you want to avoid the "dying ReLU" problem (where neurons get stuck and output 0). Variants like Parametric ReLU (PReLU) can also be used.
* **ReLU**: Use in most hidden layers of deep networks due to its simplicity and effectiveness.
* **Tanh**: Use in hidden layers when you need outputs in the range [-1, 1], but it is less common in deep learning due to the vanishing gradient problem.
* **Logistic (sigmoid)**: Use in the output layer for binary classification or in shallow networks.
* **Softmax**: Use in the output layer for multi-class classification problems.

**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 is set too close to 1, the optimizer may overshoot the optimal solution and fail to converge. The updates may become too large, causing oscillations or divergence in the training process.

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

* **L1 regularization**: Adds a penalty proportional to the absolute value of the weights, encouraging sparsity.
* **Dropout**: Randomly drops neurons during training, which can lead to a sparse network.
* **Pruning**: Removes small weights or neurons after training to create 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?**

* **Training**: Yes, dropout slows down training because it requires additional computations to drop neurons and scale activations.
* **Inference**: Dropout does not slow down inference because it is not applied during prediction.
* **MC Dropout**: Slows down inference because it requires multiple forward passes with dropout enabled to estimate uncertainty.

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

**a. Build a DNN with 20 hidden layers of 100 neurons each. Use He initialization and the ELU activation function.**

* Example code:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.initializers import HeNormal

model = Sequential()

for \_ in range(20):

model.add(Dense(100, activation='elu', kernel\_initializer=HeNormal()))

model.add(Dense(10, activation='softmax'))

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

* Example code:

from tensorflow.keras.optimizers import Nadam

from tensorflow.keras.callbacks import EarlyStopping

model.compile(optimizer=Nadam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

early\_stopping = EarlyStopping(patience=10, restore\_best\_weights=True)

history = model.fit(X\_train, y\_train, epochs=100, validation\_data=(X\_val, y\_val), callbacks=[early\_stopping])

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

* Batch Normalization often speeds up convergence and improves model performance. Add it after each hidden layer:

from tensorflow.keras.layers import BatchNormalization

model = Sequential()

for \_ in range(20):

model.add(Dense(100, activation='elu', kernel\_initializer=HeNormal()))

model.add(BatchNormalization())

model.add(Dense(10, activation='softmax'))

**d. Replace Batch Normalization with SELU and make necessary adjustments.**

* Use SELU activation and LeCun initialization:

model = Sequential()

for \_ in range(20):

model.add(Dense(100, activation='selu', kernel\_initializer='lecun\_normal'))

model.add(Dense(10, activation='softmax'))

**e. Regularize the model with alpha dropout and evaluate using MC Dropout.**

* Add Alpha Dropout (specific to SELU) and evaluate with MC Dropout:

from tensorflow.keras.layers import AlphaDropout

model = Sequential()

for \_ in range(20):

model.add(Dense(100, activation='selu', kernel\_initializer='lecun\_normal'))

model.add(AlphaDropout(rate=0.1))

model.add(Dense(10, activation='softmax'))