That's a comprehensive set of deep learning questions! Here's a structured breakdown of the answers:

**1. Weight and Bias Initialization**

1. **Is it OK to initialize all the weights to the same value using He initialization?**  
   No, it's not. If all weights are initialized to the same value, every neuron in the layer will compute the same output and update identically during backpropagation, leading to symmetry and preventing learning. He initialization ensures that the weights are randomly distributed to avoid this issue.
2. **Is it OK to initialize the bias terms to 0?**  
   Yes, initializing biases to 0 is generally fine. Unlike weights, biases do not break symmetry. They only shift the activation function's output. However, some advanced architectures use small random values instead.

**2. SELU vs. ReLU**

1. **Three advantages of the SELU activation function over ReLU:**
   * **Self-normalizing**: Ensures that activations remain standardized through the network, improving training stability.
   * **No dead neurons**: Unlike ReLU, which suffers from dead neurons when weights become negative, SELU does not have this issue.
   * **Works well for deep networks**: SELU maintains a controlled mean and variance across layers, reducing the need for batch normalization.
2. **When to use different activation functions:**
   * **SELU**: When building deep networks without batch normalization, ensuring self-normalization.
   * **Leaky ReLU & Variants**: When addressing the "dying ReLU" problem, where neurons become inactive.
   * **ReLU**: Default choice for most hidden layers due to simplicity and computational efficiency.
   * **Tanh**: When centered activations (-1 to 1) are beneficial, especially in shallow networks.
   * **Logistic (Sigmoid)**: Used in binary classification, but has issues with vanishing gradients.
   * **Softmax**: Used in multi-class classification, converting logits into probabilities.

**3. Optimization Challenges**

1. **What happens if the momentum hyperparameter is set too close to 1 (e.g., 0.99999) in SGD?**  
   The optimizer may overshoot minima, oscillate heavily, and struggle to converge, as the accumulated momentum overpowers new gradients.
2. **Three ways to produce a sparse model:**
   * **L1 regularization**: Encourages weight sparsity by penalizing absolute values.
   * **Pruning**: Removes small-weight connections post-training.
   * **Weight quantization**: Reduces precision, forcing small weights to zero.

**4. Dropout and its Impact**

1. **Does dropout slow down training and inference?**
   * **Training**: Yes, since it randomly deactivates neurons, requiring more epochs to converge.
   * **Inference**: No, dropout is turned off during inference.
   * **MC Dropout**: Slows inference because it requires multiple forward passes to estimate uncertainty.

**5. Practical Deep Learning Exercise on CIFAR10**

1. **Training a deep neural network on CIFAR10** a. **Build a DNN with 20 hidden layers of 100 neurons each**
   * Use He initialization (tf.keras.initializers.he\_normal()).
   * Use the ELU activation function (tf.keras.activations.elu).

b. **Train using Nadam optimization and early stopping**

* + Load dataset: keras.datasets.cifar10.load\_data().
  + Use Nadam optimizer (tf.keras.optimizers.Nadam()).
  + Implement early stopping (tf.keras.callbacks.EarlyStopping).

c. **Add Batch Normalization and compare results**

* + Expect faster convergence due to normalized activations.
  + Should improve model generalization.
  + Might slow down training slightly due to additional computations.

d. **Replace Batch Normalization with SELU**

* + Use LeCun normal initialization (tf.keras.initializers.lecun\_normal()).
  + Ensure inputs are standardized.
  + Avoid batch normalization, dropout, or other normalization layers.

e. **Use Alpha Dropout and apply MC Dropout at inference**

* + Alpha Dropout maintains the SELU activation's self-normalization property.
  + Apply Monte Carlo (MC) Dropout during inference for uncertainty estimation.