1. Initializing all weights to the same value is generally not recommended, even if using He initialization. This is because it can lead to all neurons in a layer computing the same function, which can limit the model's ability to learn complex features. It is better to initialize the weights with a small amount of randomness, such as by adding random Gaussian noise to the initial values.

2. Initializing bias terms to 0 is a common practice and generally works well, especially for activation functions that are symmetric around 0 (e.g., tanh). However, in some cases, it may be beneficial to initialize biases to small positive or negative values to encourage certain neurons to activate more or less frequently.

3. The three advantages of the SELU activation function over ReLU are:

- SELU can self-normalize, which can speed up training and improve performance on deep networks.

- SELU has a continuous derivative, which can make optimization easier and more stable.

- SELU is designed to have mean and variance-preserving properties, which can help mitigate the vanishing/exploding gradient problems that can occur in deep networks.

4. The choice of activation function depends on the specific problem being solved and the architecture of the neural network. Generally, ReLU is a good choice for hidden layers, tanh and logistic functions are good for output layers that need to output values between -1 and 1 or 0 and 1 respectively, and softmax is commonly used for multi-class classification problems. Leaky ReLU and its variants can help mitigate the "dying ReLU" problem that can occur in deep networks, while SELU can help with self-normalization and performance on deep networks.

5. If the momentum hyperparameter is set too close to 1, the optimizer will place a greater emphasis on the direction of the previous update and may overshoot the minimum of the cost function. This can lead to slow convergence or even divergence of the optimization process.

6. Three ways to produce a sparse model are:

- L1 regularization: Penalizing the model for having large weight values can encourage some weights to be set to 0.

- Dropout: Randomly setting some activations to 0 during training can encourage the model to learn more robust and distributed representations.

- Weight pruning: Setting small weight values to 0 can reduce the number of parameters in the model and promote sparsity.

7. Dropout can slow down training because it effectively reduces the number of training examples the model sees during each iteration. However, it can help prevent overfitting and improve generalization performance, which can ultimately result in faster convergence to a good solution. Dropout does not slow down inference, as the model uses all of its neurons during inference. MC Dropout, which involves running inference multiple times with dropout enabled and averaging the results, can be slower than regular inference but can provide more accurate uncertainty estimates.

8. Here's an outline of how to train a deep neural network on the CIFAR10 image dataset:

a. Build a DNN with 20 hidden layers of 100 neurons each, using He initialization and ELU activation function:

```

import tensorflow as tf

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Flatten(input\_shape=(32, 32, 3)))

for \_ in range(20):

model.add(tf.keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"))

model.add(tf.keras.layers.Dense(10, activation="softmax"))

```

b. Train the network using Nadam optimization and early stopping:

```

optimizer = tf.keras.optimizers.Nadam(lr=0.001)

model.compile(loss="sparse\_categorical\_crossentropy", optimizer=optimizer, metrics=["accuracy"])

early\_stopping\_cb = tf.keras.callbacks.EarlyStopping(patience=10, restore\_best\_weights=True)