1. Is it OK to initialize all the weights to the same value as long as that value is selected randomly using He initialization?
2. Is it OK to initialize the bias terms to 0?
3. Name three advantages of the SELU activation function over 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?
5. What may happen if you set the momentum hyperparameter too close to 1 (e.g., 0.99999) when using an SGD optimizer?
6. Name three ways you can produce 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?
8. Practice training a deep neural network on the CIFAR10 image dataset:
   1. 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.
   2. 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.
   3. 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?
   4. 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.).
   5. Try regularizing the model with alpha dropout. Then, without retraining your model, see if you can achieve better accuracy using MC Dropout.

Answer:

1. Initializing all the weights to the same value using He initialization is not recommended as it can lead to symmetry problems and the neurons in the same layer will always learn the same features. He initialization is used to randomly initialize the weights to avoid these issues and improve the convergence of the network.
2. It is generally OK to initialize the bias terms to 0, as long as the weights are initialized properly. However, in some cases, it may be beneficial to initialize the biases to a small positive value to prevent dead neurons, especially when using activation functions such as ReLU.
3. Three advantages of the SELU activation function over ReLU are:

* SELU can self-normalize, which can help address the vanishing/exploding gradients problem and improve convergence.
* SELU is continuous and differentiable everywhere, which can help improve the stability of the gradients and allow for more efficient optimization.
* SELU has a mean activation close to 0 and a standard deviation close to 1, which can help reduce the covariate shift and improve the generalization performance of the network.

1. In general, the choice of activation function depends on the specific problem and the characteristics of the data. Here are some guidelines:

* SELU: for deep neural networks, especially when using batch normalization and self-normalizing networks.
* Leaky ReLU and variants (e.g. Parametric ReLU, Exponential Linear Units): when avoiding dead neurons and improving the generalization performance of the network.
* ReLU: for most cases, especially when the data is sparse and the network is deep.
* Tanh and logistic: for output layers of binary classifiers and when the data is centered around 0.
* Softmax: for multi-class classification problems, where the output layer should represent a probability distribution over the classes.

1. If the momentum hyperparameter is set too close to 1, the optimizer will rely too much on the previous updates and may overshoot the minimum of the cost function. This can lead to slow convergence, or even divergence in extreme cases.
2. Three ways to produce a sparse model are:

* L1 regularization: penalizes the absolute value of the weights, encouraging some of them to become zero.
* Dropout: randomly sets some of the neurons to zero during training, forcing the network to rely on a smaller subset of the neurons.
* Weight decay: adds a penalty term to the cost function that discourages large weights, leading to some of them becoming very small.

1. Dropout can slow down training as each iteration requires more computations due to the random dropout of some neurons. However, it can help prevent overfitting and improve the generalization performance of the network. Inference (making predictions on new instances) is not affected by dropout, as it is only used during training. MC Dropout (Monte Carlo Dropout) is a technique that uses dropout during inference to estimate the uncertainty of the model predictions, and it can slow down inference.
2. Here is an outline of the steps to train a deep neural network on the CIFAR10 image dataset:

a.

from tensorflow import keras

model = keras.models.Sequential()

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

for \_ in range(20):

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

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

b.

(X\_train\_full, y\_train\_full), (X\_test, y\_test) = keras.datasets.cifar10.load\_data()

X\_train\_full = X\_train\_full / 255.0

X\_test = X\_test / 255.0

X\_valid, X\_train = X\_train\_full[:5000], X\_train\_full[5000:]

y\_valid, y\_train = y\_train\_full[:5000], y\_train\_full[5000:]

early\_stopping\_cb = keras.callbacks.EarlyStopping(patience=20, restore\_best\_weights=True)

optimizer = keras.optimizers.Nadam(lr=5e-5)

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

history = model.fit(X\_train, y\_train, epochs=100, validation\_data=(X\_valid, y\_valid),

callbacks=[early\_stopping\_cb])

c.

from tensorflow import keras

model = keras.models.Sequential()

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

for \_ in range(20):

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

model.add(keras.layers.BatchNormalization())

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

d.

from tensorflow import keras

model = keras.models.Sequential()

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

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

for \_ in range(19):

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

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

e.

from tensorflow import keras

model = keras.models.Sequential()

model.add(k