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. ) It is not advisable to set all weights to the same initial value, even if He initialization is used to choose the weights at random. He initialization is used to set initial weights to values that minimize the risk of gradients exploding or disappearing during training. The goal is defeated if all weights are the same because every neuron will initially learn the same features, making it more difficult for the network to learn a variety of representations.
3. Is it OK to initialize the bias terms to 0?
4. ) In general, bias terms can be initialized to 0. Initializing bias terms to 0 makes the initial learning process simpler. Bias terms are adjusted during training. Initializing biases to non-zero values, however, might be advantageous in some circumstances, particularly when working with particular activation functions or network architectures.
5. Name three advantages of the SELU activation function over ReLU.

A.) The SELU activation function has three advantages over ReLU.

a. Self-normalization: By encouraging self-normalization, SELU helps to lessen the effects of the vanishing/exploding gradient issue without requiring the use of extra methods like batch normalization.

b. Stability: Because SELU permits negative values during training, it is less prone than ReLU to experience dead neurons.

c. Smoothness: The gradient-based optimization process can benefit from SELU's smoothness as a function.

1. 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?

A.) Using  the activation functions :

When constructing deep neural networks, SELU: Prefer, particularly if self-normalization is desired.

Breach ReLU (and its variations): Suitable for deeper networks, effective in preventing dead neurons.

ReLU: Because of its simplicity, it is frequently used; however, it may have dead neurons.

Tanh and Logistic: Fit for hidden layers where outputs that are normalized are required.

Softmax: Usually applied in the multi-class classification problem's output layer.

1. What may happen if you set the momentum hyperparameter too close to 1 (e.g., 0.99999) when using an SGD optimizer?
2. ) If the momentum hyperparameter in SGD is set too close to 1, for example, 0.99999, the optimizer might overshoot the minimum and oscillate around it. This may cause training to converge slowly or become unstable.
3. Name three ways you can produce a sparse model.

A.) Three techniques for creating a sparse model:

a. L1 Regularization: This encourages sparsity by penalizing the weights' absolute values.

b. Dropout: During training, a random portion of neurons are dropped, creating sparsity.

c. Pruning: Eliminating neurons or connections that make insignificant contributions to the functionality of the model.

1. Does dropout slow down training? Does it slow down inference (i.e., making predictions on new instances)? What about MC Dropout?
2. ) Because it introduces randomness and necessitates more iterations for convergence, dropout can slow down training. It can, however, stop overfitting. Dropout is disabled during inference, so predictions are not slowed down. MC Dropout can improve prediction certainty by executing the model several times while keeping dropout enabled during the test and averaging the outcomes.
3. 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.
4. ) I am unable to run code because of restrictions in this text-based environment. The following steps must be completed in your Python environment with the necessary libraries.

**# CODE**

**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"))**

* 1. 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.

1. ) Train with Nadam optimization and early stopping.
   1. 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?
2. ) Add Batch Normalization and compare learning curves.
   1. 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.).
3. ) Replace Batch Normalization with SELU.
   1. Try regularizing the model with alpha dropout. Then, without retraining your model, see if you can achieve better accuracy using MC Dropout.

A.) Regularize with alpha dropout and experiment with MC Dropout for better accuracy.