1. Is it OK to initialize all the weights to the same value as long as that value is selected randomly using He initialization?

**Initializing Weights with He Initialization**: It is generally a good practice to initialize weights randomly using techniques like He initialization. Initializing all weights to the same value (even if randomly chosen) can lead to symmetry problems and hinder training

1. Is it OK to initialize the bias terms to 0?

**Initializing Bias Terms**: Initializing bias terms to 0 is a common practice and is generally acceptable. The main reason is that bias terms are responsible for shifting the activation function, and initializing them to 0 ensures that the network starts with a neutral bias.

1. Name three advantages of the SELU activation function over ReLU.

**Advantages of SELU over ReLU**:

* 1. Self-normalization: SELU activation encourages the network's activations to self-normalize during training, addressing vanishing/exploding gradient issues.
  2. Improved vanishing gradient: SELU has a derivative that avoids vanishing gradients in deep networks, making it suitable for very deep architectures.
  3. Outperforms ReLU: SELU often achieves better performance than ReLU on deep neural networks.

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?

**Activation Functions for Different Cases**:

* 1. **SELU**: Useful for deep networks where self-normalization is beneficial.
  2. **Leaky ReLU and Variants**: Suitable when you want to mitigate the dying ReLU problem (ReLU neurons becoming inactive).
  3. **ReLU**: A widely used default choice, especially for convolutional layers.
  4. **Tanh**: Suitable for hidden layers in feedforward neural networks.
  5. **Logistic (Sigmoid)**: Commonly used in binary classification tasks.
  6. **Softmax**: Primarily used in the output layer for multi-class classification problems.

1. What may happen if you set the momentum hyperparameter too close to 1 (e.g., 0.99999) when using an SGD optimizer?

**Momentum Hyperparameter in SGD Optimizer**:

* 1. Setting the momentum hyperparameter too close to 1 (e.g., 0.99999) can lead to very slow convergence and even instability in training.
  2. This is because momentum accumulates historical gradients, and a high value makes the optimizer overly reliant on past gradients, slowing down updates.

1. Name three ways you can produce a sparse model.

**Producing Sparse Models**:

* 1. Weight Pruning: Remove small-weight connections from the neural network.
  2. L1 Regularization (Lasso): Add a regularization term to the loss function that encourages small weights, leading to sparsity.
  3. DropConnect: A variant of dropout where instead of dropping neuron activations, connections are dropped randomly.

1. Does dropout slow down training? Does it slow down inference (i.e., making predictions on new instances)? What about MC Dropout?

**Dropout and Training Speed**:

* 1. Dropout can slow down training because it introduces randomness and forces the network to be more robust, requiring longer training to converge.
  2. During inference (making predictions), dropout is typically turned off, so it doesn't slow down the prediction process.
  3. MC Dropout (Monte Carlo Dropout) involves running inference with dropout enabled multiple times and averaging the results, which can be slower than a single prediction but provides better uncertainty estimates.

1. 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.

DeepNeuralNetwork on the CIFAR10 image dataset.ipynb