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

A1. Yes, it is generally okay to initialize all the weights to the same value as long as that value is selected randomly using a proper initialization method such as He initialization. He initialization sets the initial weights randomly with a mean of zero and a standard deviation that is based on the number of input and output units in the layer. This ensures that the weights are initialized to appropriate values that allow for efficient learning in the neural network. However, initializing all weights to the same value may limit the exploration of the weight space and can lead to symmetries in the network, which can hinder learning. Therefore, it is generally recommended to use a random initialization method like He initialization rather than initializing all weights to the same value.

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

A2. Yes, it is generally considered OK to initialize the bias terms to 0. This is because the bias term is used to shift the output of the activation function to the left or right, without affecting the shape of the activation function. In most cases, the shape of the activation function is more important than the shift, so initializing the bias to 0 is a reasonable default. However, in some cases, it may be beneficial to initialize the bias to a non-zero value, particularly if the input data has a significant bias or if the activation function is not symmetric around 0.

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

A3. Here are three advantages of the SELU activation function over ReLU:

1. The SELU activation function is a scaled version of the ReLU activation function, which allows for faster and more stable learning. This is because the SELU function has a mean of 0 and a standard deviation of 1, which helps to avoid the problem of vanishing and exploding gradients.
2. The SELU activation function is a self-normalizing function, which means that it can automatically adjust the scale of the activations in the network. This can lead to better convergence and performance, especially in deep networks.
3. The SELU activation function is able to handle negative inputs, unlike the ReLU function which sets negative inputs to 0. This can result in more accurate and expressive representations of the input data.
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?

A4. The choice of activation function in a neural network depends on the specific task and the characteristics of the data. Here are some guidelines for when to use each of the mentioned activation functions:

1. SELU (Scaled Exponential Linear Unit): SELU is a newer activation function that has shown to perform well in deep neural networks. It has the property of self-normalization, which means that the output of each layer has zero mean and unit variance, leading to faster and more stable convergence. SELU should be used in deep neural networks with dense connections, where it has been shown to outperform other activation functions.
2. Leaky ReLU and its variants: Leaky ReLU is a modified version of the ReLU activation function that prevents the “dying ReLU” problem, where neurons can permanently become inactive during training. It has a small non-zero gradient for negative input values, allowing neurons to recover from negative inputs. Leaky ReLU and its variants (such as Parametric ReLU and Randomized ReLU) can be used in neural networks where ReLU does not perform well, such as in deep neural networks with sparse connections or with a large number of layers.
3. ReLU (Rectified Linear Unit): ReLU is the most commonly used activation function in deep learning. It is simple to compute, and its sparsity property helps in reducing overfitting. ReLU should be used in neural networks where there are many layers or when computational efficiency is important.
4. Tanh (Hyperbolic Tangent): Tanh is a symmetric activation function that outputs values between -1 and 1. It is useful when the input data is normalized and centered around zero, such as in image processing tasks.
5. Logistic (Sigmoid): Logistic function is useful for binary classification tasks where the output of the neural network needs to be between 0 and 1. It can also be used as the output activation function for the last layer of a multi-class classification neural network.
6. Softmax: Softmax is used as the output activation function for multi-class classification tasks. It outputs a probability distribution over the classes, with the sum of probabilities equal to 1.

Top of Form

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

A5. When the momentum hyperparameter is set too close to 1 (e.g., 0.99999) when using an SGD optimizer, it may cause the optimizer to overshoot the minimum point and oscillate back and forth, resulting in slow convergence or even divergence of the training process. This happens because the momentum term accumulates too much of the previous gradients, causing the optimizer to take excessively large steps in the same direction. As a result, it may fail to find the minimum point and even diverge from it. Therefore, it is important to set an appropriate value for the momentum hyperparameter to balance the trade-off between exploration and exploitation during the optimization process.

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

A6. Here are three ways to produce a sparse model:

1. **L1 Regularization**: Introducing L1 regularization (also known as Lasso regularization) in the training process will push the weights of the neural network to zero, which leads to a sparse model. This technique encourages the neural network to use a small number of important features and discard the rest.
2. **Dropout**: Dropout is a regularization technique that randomly sets a portion of the neurons in a layer to zero during training. By doing so, the model learns to rely on a subset of neurons for each input and ignores the other neurons, leading to a sparse model.
3. **Weight pruning**: Weight pruning is the process of removing small-weight connections from the neural network. After training the model, we can set a threshold value for the weights, and all the weights below that threshold will be set to zero. This will lead to a sparse model where a large number of connections have been pruned.
4. Does dropout slow down training? Does it slow down inference (i.e., making predictions on new instances)? What about MC Dropout?

A7. Dropout is a regularization technique that randomly drops out (sets to zero) a fraction of the neurons' outputs in a neural network during training. This helps to prevent overfitting and encourages the network to learn more robust features. Dropout can slightly slow down training because the network is forced to learn redundancy and become more robust.

During inference (i.e., making predictions on new instances), dropout is typically turned off, so it does not affect the speed of inference. However, the predictions may be less accurate because the network may have become too reliant on the dropped-out neurons during training.

MC Dropout (Monte Carlo Dropout) is an extension of the dropout technique that provides a way to estimate the uncertainty of the model's predictions. MC Dropout randomly drops out neurons during inference as well, but instead of turning them off completely, it scales their outputs by the probability of keeping them. This effectively creates multiple "thinned-out" versions of the network, and the predictions are made by averaging the outputs of these different versions. This process is repeated several times, and the variance of the predictions across these runs can be used to estimate the uncertainty of the model's predictions. MC Dropout can slow down inference because it involves running the network multiple times, but it can provide more accurate predictions and better estimates of uncertainty.

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.