1) Is it ok to initialize all the weights to the same value as long as that value selected randomly using initialization ?

Ans : -No, all weights should be sampled independently; they should not all have the same initial value.

-One important goal of sampling weights randomly is to break symmetry: if all the weights have the same initial value, even if that value is not zero, then symmetry is not broken and backpropagation will be unable to break it.this means that all the neurons in any given layer will always have the same weights. It’s like having just one neuron per layer, and much slower.

-It is virtually impossible for such a configuration to converge to a good solution.

2) Is it ok to initialize the bias term to 0 ?

Ans : It is perfectly fine to initialize the bias terms to zero. Some people like to initialize them just like weights and that’s okay too it does not make much difference.

3) Name three advantages of the SELU activation over ReLU.

Ans : - It can take on negative values, so the average output of the neurons in any given layer is typically closer to zero than when using the ReLU activation function This helps alleviate the vanishing gradients problem.

-It always has a nonzero derivative, which avoids the dying units issue that can affect ReLU units.

-When the conditions are right then the SELU activation function ensures the model is self-normalized, which solves the exploding/vanishing gradients problems.

4)In which cases would u want to use each of the following activation function : SELU ,leaky ReLU,ReLU,tanh,logistic, and softmax ?

Ans : T-he SELU activation function is a good default.

-If you need the neural network to be as fast as possible, you can use one of the leaky ReLU variants instead (e.g., a simple leaky ReLU using the default hyperparameter value).

-simplicity of the ReLU activation function makes it many people’s preferred option, despite the fact that it is generally outperformed by SELU and leaky ReLU. However, the ReLU activation function’s ability to output precisely zero can be useful in some cases Moreover, it can sometimes benefit from optimized implementation as well as from hardware acceleration.

-hyperbolic tangent (tanh) can be useful in the output layer if you need to output a number between –1 and 1, but nowadays it is not used much in hidden layers (except in recurrent nets).

-logistic activation function is also useful in the output layer when you need to estimate a probability rare in hidden layer

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

Ans : If you set the momentum hyperparameter too close to 1 (e.g., 0.99999) when using an SGD optimizer, then the algorithm will likely pick up a lot of speed,moving roughly toward the global minimum, but its momentum will carry it right past the minimum. Then it will slow down and come back, accelerate again and so on. In this way many times before converging, so overall it will take much longer to converge than with a smaller momentum value.

6) name three ways you can produce a sparse model.

Ans : One way to produce a sparse model (i.e., with most weights equal to zero) is to train the model normally, then zero out tiny weights.

-For more sparsity, you can apply ℓ1 regularization during training, which pushes the optimizer toward sparsity.

-A third option is to use the TensorFlow Model Optimization Toolkit.

7) Does dropout slow down training ? Does it slow down inference ? What about MC dropout ?

Ans : Yes, dropout does slow down training, in general roughly by a factor of two. However, it has no impact on inference speed since it is only turned on during training. MC Dropout is exactly like dropout during training, but it is still active during inference, so each inference is slowed down slightly. More importantly, when using MC Dropout you generally want to run inference 10 times or more to get better predictions. This means that making predictions is slowed down by a factor of 10 or more.

8)practice training a deep neural network on the CIFAR10 image dataset.

a.Build a CNN with 20 hidden layers of 100 neurons each.use he initialization

And the ELU activation function.

ans :

CIFAR stands for the Canadian Institute For Advanced Research and the CIFAR-10 dataset was developed along with the CIFAR-100 dataset by researchers at the CIFAR institute.

b.using nadam optimization and early stopping train network on the CIFAR10 dataset.you can load it with keras.dataset.cifar10.load\_data().the dataset is composed of 60,000 32\*32 pixel color image with 10 classes so you will need a softmax output layer with 10 neuron remember to search for the right learning rate each time you change the models architecture or hyperparameters.

Ans : img, label = dataset[0]

img\_shape = img.shape

img\_shape

torch.size([3,32,32])

num\_classes = len(dataset.classes)

num\_classes

10

classes = dataset.classes

classes

['airplane',

'automobile',

'bird',

'cat',

'deer',

'dog',

'frog',

'horse',

'ship',

'truck']

dataset = CIFAR10(root='data/', download=True, transform=ToTensor())

test\_dataset = CIFAR10(root='data/', train=False, transform=ToTensor())

C.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 effect training speed ?

ans : The principle of normalization of training data can also be applied to the values of neurons in a neural network, which can greatly improve the training of a neural network.With batch normalization, we reduce the internal covariance shift. Or in other words, the amount by which the values of the neurons in the hidden layers shift.

One difference from normalizing data is that in batch normalization, we normalize the neuron values over a so-called mini-batch of m instances, rather than over all instances of the training dataset.

Stable gradients allow the use of higher learning rates. In gradient descent, small learning rates are usually required for the mesh to converge. And as the meshes get deeper, the gradients during gradient descent get smaller, so the training time increases with the depth of the mesh. With batch normalization, we can use much higher learning rates, further increasing the training speed of neural networks.

D.try replacing batch Normalization with SELU and make the necessary Adjustment to ensure the network self normalization.

Ans : Batch normalization is a technique for training very deep neural networks that normalizes the contributions to a layer for every mini-batch. This has the impact of settling the learning process and drastically decreasing the number of training epochs required to train deep neural networks.

E.try Regularizing the model with alpha dropout then without retraining your model see if you can achieve better Accuracy using MC dropout.

Ans : With dropout (dropout rate less than some small value), the accuracy will gradually increase and loss will gradually decrease first(That is what is happening in your case). When you increase dropout beyond a certain threshold, it results in the model not being able to fit properly.

Dropout regularization is a technique to prevent neural networks from overfitting. Dropout works by randomly disabling neurons and their corresponding connections. This prevents the network from relying too much on single neurons and forces all neurons to learn to generalize better.Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks.