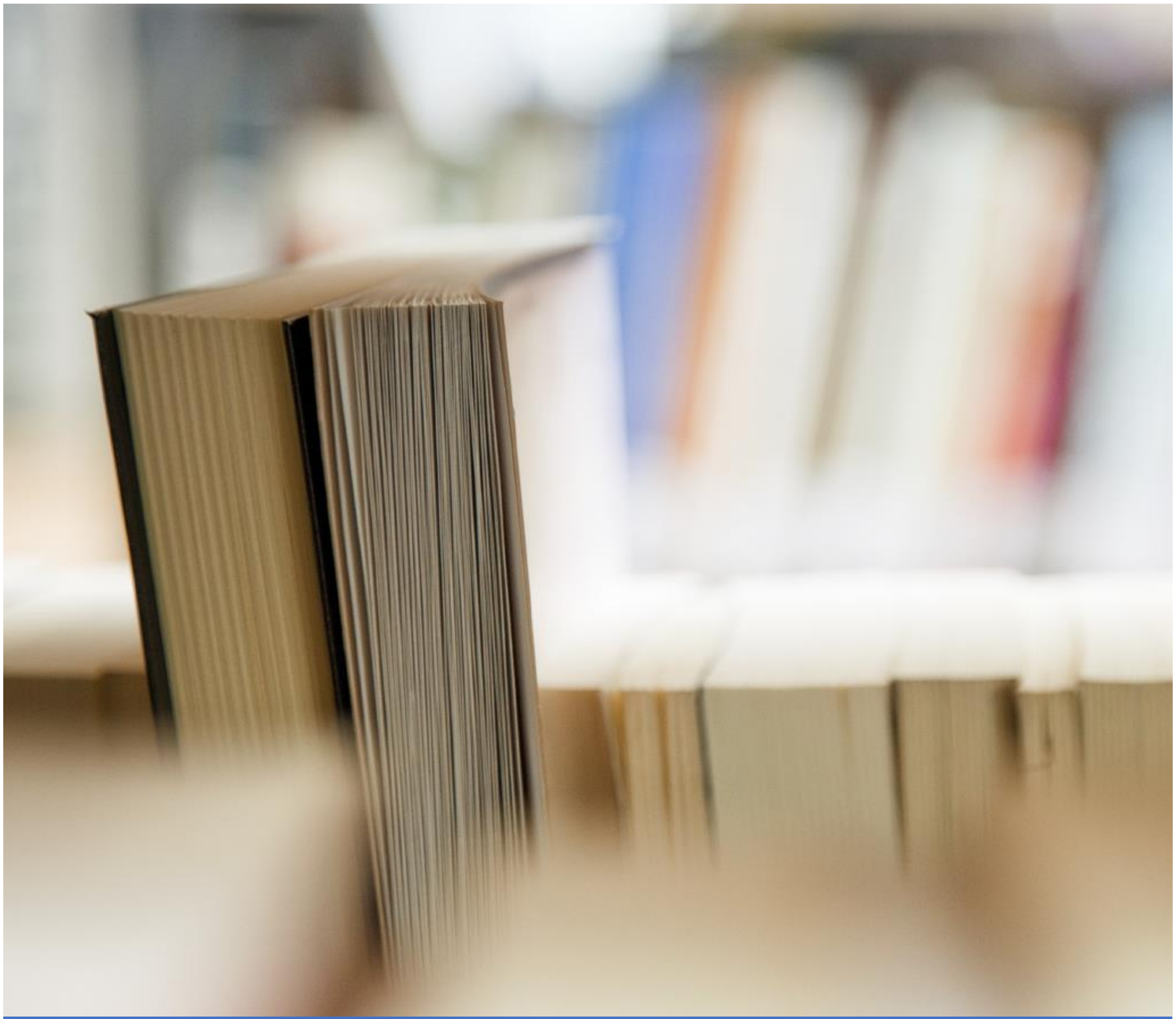


ECCE 635 Deep Learning Systems Design Assignment no 2

Muhammad Muneeb 9/6/20 ID 100052975@ku.ac.ae

Part 1: Conceptual Questions

1. **Why is it necessary to include non-linearities in a neural network?**
Non-linearity is needed in neural network because of the following reasons.
It produces a nonlinear decision boundary via non-linear combinations of the weight and inputs.
It helps to find the non-linear and complex relationships between input features and helps to find a better decision boundary.

2. When the input is 2-dimensional, you can plot the decision boundary of your neural network and clearly see if there is overfitting. How do you check overfitting if the input is 10-dimensional?
There are many ways to check overfitting in 10-D data.
First method.
We can compare the training loss and test loss.
If training loss is very low as compared to test loss it means model is overfit.
Training loss should be almost same as that of test loss if model is not overfit.

Second method
If our model does much better on the training set than on the test set, then we're likely overfitting. For example, if our model has 99% accuracy on the training set but only 55% accuracy on the test set.

Cross-validation is a powerful preventative measure against overfitting.

3. You are designing a deep learning system to detect driver fatigue in cars. It is crucial that that your model detects fatigue, to prevent any accidents. Which of the following is the most appropriate evaluation metric: Accuracy, Precision, Recall, Loss value? Explain your choice.
Precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned.

Well detecting fatigue in car is very important to warn driver so that accidents can be avoided. It is just like detecting cancer disease in which we do not want to lose any positive case. We do not care if person who does not have cancer is detected as positive.
So, **Recall** is the most important evaluation metric in this case because we do not want to miss any positive case even if some negative cases are detected as positive.

4. You want to solve a classification task. You first train your network on 20 samples. Training converges, but the training loss is very high. You then decide to train this network on 10,000 examples. Is your approach to fixing the problem correct? If yes, explain the most likely results of training with 10,000 examples. If not, give a solution to this problem.

If training loss is very high, then it means that model has not learned the proper weights to make accurate predictions. But it is given that our model will converge so it means if we train it more then training loss will reduce.
Training on 10,000, will not work. It is not something that is related to dataset. Even for 10,000 training loss will be high.
Solution
We must increase the number of iterations or epochs so our model weights can be updated properly, and training loss can be reduced.

5. Give benefits of using convolutional layers instead of fully connected ones for visual tasks.
Main feature of CNNs is weight sharing. It reduces the number of parameters that model need to learn for accurate prediction. It saves memory and reduce time.
1 **Spatial Information**
CNN has many different layers to get the spatial information like convolution layer. Convolution layer helps to detect character even if there is translation (position) variation in the character in image.
2 **Connected Component**
In real life objects are composed of many sub components and CNN will detect the sub objects and next layers in CNN will detect the whole object.

6. You are training a neural network model. You initialize the parameters with 0's. Is this a good idea?

Explain your answer.

If parameters are 0 then training will be slow and there is possibility that model does not produce any useful result. We should initialize parameters to random.

Now imagine that you initialize all weights to the same value (e.g. zero or one). In this case, each hidden unit will get the same signal. E.g. if all weights are initialized to 1, each unit gets signal equal to sum of inputs (and outputs sigmoid(sum(inputs))). If all weights are zeros, which is even worse, every hidden unit will get zero signal. If all weights are the same, all units in hidden layer will be the same too.

Consider MNIST dataset.
Parameters are initialized like this

```
initiationmethods = [tf.keras.initializers.Ones(),tf.keras.initializers.Ze
ros(),tf.keras.initializers.RandomNormal(mean=0., stddev=1.)]
```

Accuracy

```
[10.100000351667404, 11.349999904632568, 93.99999976158142]
```

So, we can see that for all 1 and 0 out model has accuracy of about 10 and 11 respectively. So, we should initialize weights to random.

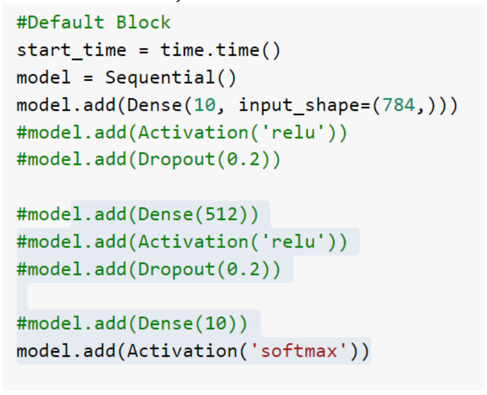
7. You are given a dataset of 28 X28 grayscale images. Your goal is to build a 5-class classifier. You have to adopt one of the following two options:

- **the input is flattened into a 784-dimensional vector, followed by a fully connected layer with 5 neurons.**
- **the input is directly given to a convolutional layer with five 10 X10 filters.**

Explain which one you would choose and why.

Consider first case.
Number of parameters will be 784*5 + 5 = 3925

Consider second case.
Initial size = 28*28*1
Number of parameters for convolution layer will be (10*10+1) *5 = 105
Where as next layer will be 28-(10-1) = 19
19*19*5 = 1805
We have to flatten this layer and directly attach it to fully connected network.
So 1805*5 = 9,025



```
#Default Block
start_time = time.time()
model = Sequential()
model.add(Dense(10, input_shape=(784,)))
#model.add(Activation('relu'))
#model.add(Dropout(0.2))

#model.add(Dense(512))
#model.add(Activation('relu'))
#model.add(Dropout(0.2))

#model.add(Dense(10))
model.add(Activation('softmax'))
```

Total parameter in CNN will be 9025+105 = 9,130

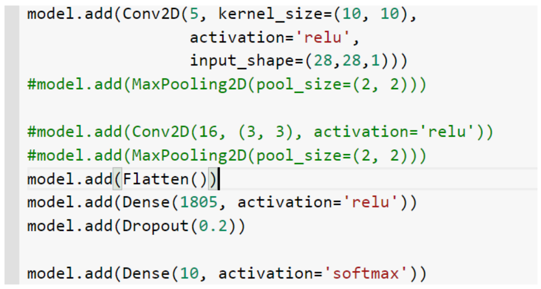
Now have a look at accuracy of the network
For first case

```
Test accuracy: 92.0%

--- 5.081097602844238 seconds ---
```

For second case

```
Test accuracy: 98.3%
--- 259.8927917480469 seconds ---
```



```
model.add(Conv2D(5, kernel_size=(10, 10),
                 activation='relu',
                 input_shape=(28,28,1)))
#model.add(MaxPooling2D(pool_size=(2, 2)))

#model.add(Conv2D(16, (3, 3), activation='relu'))
#model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(1805, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(10, activation='softmax'))
```

For first case time is very less because that network has very few parameters but in second case we have many parameters.
In first case accuracy is low but in second case we get very high accuracy,

8. You would like to train a dog/cat image classifier using mini-batch gradient descent. You have already split your dataset into train, validation, and test sets. The classes are balanced. You realize that within the training set, the images are ordered in such a way that all the dog images come first, and all the cat images come after. A friend tells you: "you absolutely need to shuffle your training set before the training procedure." Is your friend, right? Explain.

Yes, he is right. We must shuffle data in this case.

In mini-batch gradient descent, training dataset is divided into small batches that are used to calculate model error and update model coefficients.

Consider we have 100 points in dataset. 50 dogs and 50 cats with 10 batch size.

If we are using mini-batch then parameters will be updated for all the dogs images and after training on 5 mini-batch (ONLY DOG images) model loss function and parameters will be stable but when next batch will come which contain only cat images the parameter of model will change radically and the loss function value will increase. Model parameters will be more biased towards cat images and there is possibility that they will not classify dog images properly. So, we must shuffle our dataset for proper training.

Shuffling data serves the purpose of reducing variance and making sure that models remain general and overfit less.

Part 2: Programming Assignment

Part 3:
Compare between MLP and Convolutional Neural Network models for character recognition?

For character recognition CNN is better as compared to MLP because of following reasons.
1 Spatial Information
CNN has many different layers to get the spatial information like convolution layer. Convolution layer helps to detect character even if there is translation (position) variation in the character in image. Whereas in NN there is possibility that translations or rotations of original object may be treated as different.
2 Connected Component
In real life objects are composed of many sub components and CNN will detect the sub objects and next layers in CNN will detect the whole object. Whereas in NN it detects the whole object not the sub objects from which object is made of.
3 Max pooling layer in CNN will help us to detect the object even if there is rotation of character in image.
4 CNNs have two non-linearities: pooling layers and ReLU functions. Whereas in NN we have only 1 way of producing non-linearity.
5 Shared Weights
The forward pass becomes the equivalent of convolving a filter over the image to produce a new image. The training of CNNs then becomes the task of learning filters. Whereas in NN in backpropagation weights are updated to make model stable for accurate predictions.