Neural Networks and Deep Learning Exam

Mohammed Fahmidur Rahman

220401405

Question 1:

a. Deep models, with multiple hidden layers, are essential to neural networks as they allow the network to capture complex features of the input data that would otherwise be difficult to learn. Through hierarchical feature learning, the initial layers allow the model to learn simple patterns but as the data passes through more of the layers, the model can recognise and understand more complex patterns and high-level features. This layered representation of deep neural networks allows the model to capture complex, non-linear relationships in data and capability to handle more complex functions, where even deeper architectures would have lots of layers of data from things such as face scans or image recognition. With proper training, deeper models can outperform and generalise better on unseen data than shallow models with less layers as they have the ability to learn complex patterns that are not necessarily part of the training data, which helps overfitting and ultimately performance.

b. Batch Normalisation is a crucial technique used in CNNs, they allow the training of the model to be done faster and with more stability through normalising the inputs to each layer and then adjusting the data to have a consistent scale, mean of zero and variance of one. In a typical CNN without batch normalisation, the layers inputs would change during training, known as internal covariate shift, where the distribution of layer inputs changes when the model parameters are adjusted; however batch normalisation solves this by computing the mean and variance of each mini-batch which allows data to be scaled accordingly. This makes the network less susceptible to the changes in data as the model learns, allowing for a larger learning rate to speed up training.

Residual connections is another technique used in CNNs which allows deep networks to become more easy to train. Models struggle to learn due to the vanishing gradient problem which occurs when training deep neural networks. Regularly, the gradient is the value that is used to update the model weights however in deep networks with many layers, the gradient can become very small as they are back-propagated through each layer and this causes the gradient to shrink to a point that the weights become no longer updated and therefore the model struggles to train. In residual networks, the output of a single layer is added to its input creating a “Skip Connection” which allows the gradient to flow directly through the network during back-propagation. Because of this, the network now will learn the difference between the input and output of that layer making it easier to train.

Question 2:

1. First Block: 512 filters of size 7*7, applied to the input image of size 3*128*128

   Parameters: (3*7*7) * 512 = 75264

2. Second Block: Feature map split to 8 sub-tensors then reassembled into feature map

   Per sub-tensor: (64*3*3) * (512/8) = 36864
   Parameters: 36864 * 8 = 294912

3. Classifier: Performs spatial average pooling to reduce feature map to a 1x1 dimension

   Parameters: 512*30 = 15360

Total Parameters in the network: 75264 + 294912 + 15360 = **385536**

Question 3:

The model given does not contain any bugs and would run properly, however it is inefficient and has a few issues. The ML engineer has made a model that does not have any pooling layers which is essential in CNNs to reduce the dimensions of feature maps which would then allow the model to be more efficient and prevent overfitting. With the image size being 28*28, the model will be computationally expensive and inefficient due to the absence of pooling layers as well as the ridiculously large amount of filters (512 with kernel size of 3*3) for a tiny image as the network will have too many parameters to train effectively on the small images. Pooling layers in CNNs are usually used to reduce the spatial dimensions of the input's feature map which reduces parameters and prevents overfitting, they also help the network to become more resistant to slight shifts and distortions in the input. Without pooling, the model is prone to overfitting and will struggle to generalise effectively, making it unnecessarily slow to train and execute. For this reason, incorporating pooling layers is a must to improve the model's efficiency, performance and reduce the computational cost.

Question 4:

a. To achieve building a lightweight model with the optimal number of blocks (N) whilst scoring an accuracy of at least 90%, we must begin with a baseline model that has a small number of N and then gradually increase it as we train our model. This ensures that the model can recognise and understand more complex features, however it will also increase the number of parameters we are using. We will apply early stopping to the model to ensure that when the training weights and validation accuracy has plateaued, we are not doing any unnecessary training and overfitting which would otherwise make the model inefficient. We could use data augmentation techniques to manually increase the size of the training dataset to improve the model's robustness. As we re-train the

model each time, the N is gradually increased to which we are assessing the model's accuracy (which should increase) compared to the also increasing number of parameters, finding the balance to which the accuracy continues to increase without disproportionally increasing the model size. Additionally we could also utilise hyper-parameter optimisation, this is where parameters are adjusted, such as the learning rate, batch size, and the number of epochs, to help us achieve the optimal configuration for both accuracy and efficiency. We may also want to use model compression techniques like pruning or quantisation to allow us to reduce the model size without significantly reducing the model accuracy. Through the use of all of these techniques, we look for the optimal number of blocks in the model that consistently achieves 90% accuracy while keeping parameters as low as possible.

b. A desired feature that a recurrent model should have is the ability to handle long-term dependencies effectively. For long and complicated sequences, RNNs struggle to recall important information from earlier parts of the sequence due to the vanishing gradient problem. In basic RNNs, as the sequence length increases, the gradients that are used to update the weights during back-propagation become very small and therefore the models struggle to learn long-term relationships in data. To solve this, advanced RNNs use mechanisms known as Long Short-Term Memory (LSTMs) and Gated Recurrent units (GRU). These mechanisms regulate the flow of information through the network, ensuring important information is retained over long sequences and irrelevant information is forgotten. LSTMs have 3 gates: input to control what goes in, output to determine what goes out, and forget to control what is discarded. GRUs have a single update gate which combines forget and input into one, making them effective whilst also having the ability to capture long-term dependencies.

Bi-directional RNNs enhance the model by processing long sequences in both directions. This allows the network to utilise the future (end) context to help with the beginning (start) of the sequence when processing and understanding information, which also helps with making predictions. By considering both ends of a sequence and processing information from the front and end, bi-directions RNNs are able to make a comprehensive understanding of the sequence which, in turn, improves the model's performance for long and complicated sequences.

Another key feature to improve recurrent models would be attention mechanisms, they allow a model to spend more time and focus on specific parts of the input sequence rather than treating each part of the sequence equally. This allows the models to focus on the most relevant information at each time step. Through assigning weights to different parts of the sequence, attention mechanisms allow the model to learn long-term dependencies more effectively by not spending time on irrelevant bits of data, ultimately reducing the impact of less important data improving the model's performance on long sequences and allowing it to make more accurate predictions.