Deep Learning Report

Minor Exam - 2

Take Home Assignment

Ayush Abrol
B20AI052

Question 01

Aim: Train Student-Teacher model with the following details.

1. Following are the details which are variable for each student and my details are as follows:
    - 1. DD/MM/YY - 18/03/02
    - 2. ABC = 052
    - 3. FIRST = Ayush
    - 4. LAST = Abrol
    - 5. PROG - B20AI
2. As ABC is even, dataset is *CIFAR10*.
3. As MM is odd, weight initialization is "*He initialization*".
4. As MM is odd, *MaxPool* to be used.
5. As the last digit o ABC < 6 and ABC is even, the teacher network should have 7 convolutional layers and 1 pooling layer and the first layer should have 6 filters.
6. As DD is even, the student network has 2 conv layers + 1 pool layer.
7. As the sum of digits of ABC is odd, the network should have 1 FC layer with 256 nodes for classification.

We have to report the performance of the student network and compare it with the teacher model. Also, we need to compare the performance with and without EMA (Exponential Moving Average) updates.\

Procedure:

- All the necessary dependencies and libraries were taken care of initially like:
    - numpy, pandas, matplotlib for data manipulation and visualization.

    - Torch, torchvision, torch.nn, PIL for for implementing Deep Learning architectures where necessary using PyTocrch.
- Loading of the training and the testing **CIFAR10 dataset**. **50k training** samples and **10k testing** with 3 channels (RGB) samples are loaded and converted into Torch format Tensors initially.
- **classes** = {0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'truck'}
- Then, we visualized the dataset by displaying a grid of 10 images from a training dataset along with their corresponding labels.

- Then, I created new dataloaders with the permuted shape {50000, 3, 32, 32}.
- And set the environment to GPU.
- Then I defined a PyTorch module class called **TeacherModel**, which is a convolutional neural network (CNN) for image classification.
- The __init__ function defines the architecture of the CNN. Specifically, it consists of 7 convolutional layers, each followed by a ReLU activation function, and a max pooling layer. The last layer of the CNN is a fully connected (linear) layer with 256 output features, followed by another fully connected layer with 10 output features corresponding to the 10 classes in the CIFAR-10 dataset.
- The **nn.init.kaiming_normal_ function** initializes the weights of the convolutional and fully connected layers using the Kaiming/He initialization method.
- The forward function defines how the input data is propagated through the layers of the CNN. Specifically, the input data is passed through each convolutional layer, followed by a ReLU activation function, and then through the max pooling layer.

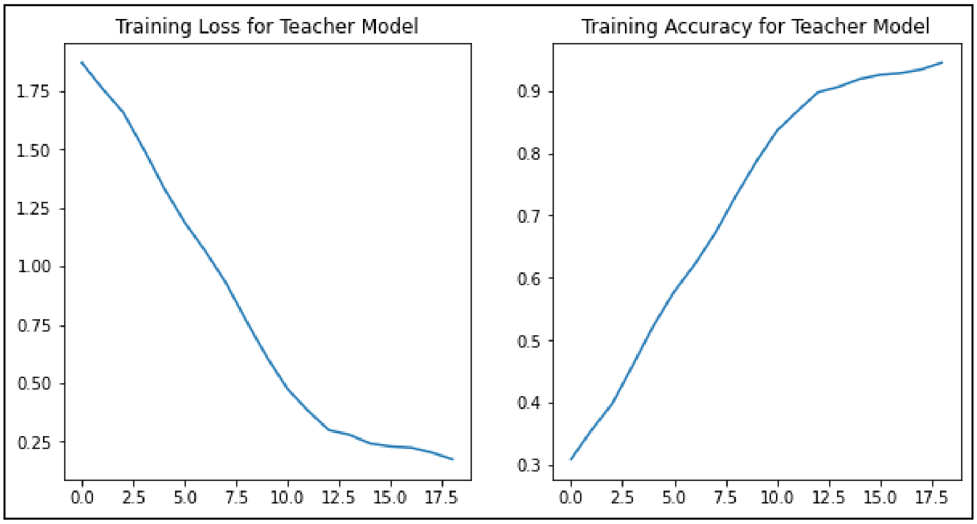
After the final max pooling layer, the output is flattened and passed through the two fully connected layers to obtain the final output of the network.

- Note that the forward function does not include a softmax activation function, which is typically used for multiclass classification tasks. This is because the network is being trained using the knowledge distillation method, which involves using the output of the teacher model as "soft" labels for the student model.
- Then, I instantiated the Teacher model object with Cross Entropy Loss and Adam optimizer to train the Teacher.
- Then, I trained the teacher model for 20 epochs and got the following losses and training accuracies.

```
Epoch: 1 Training Loss:  59.3785232823828       Training Accuracy:  0.24658
Epoch: 2 Training Loss:  1.871787279158297      Training Accuracy:  0.30952
Epoch: 3 Training Loss:  1.7614213542255295     Training Accuracy:  0.3564
Epoch: 4 Training Loss:  1.6606790854802826     Training Accuracy:  0.39844
Epoch: 5 Training Loss:  1.5033126188361126     Training Accuracy:  0.46012
Epoch: 6 Training Loss:  1.334820045534607      Training Accuracy:  0.52328
Epoch: 7 Training Loss:  1.1880343233228035     Training Accuracy:  0.57758
Epoch: 8 Training Loss:  1.0656714410428196     Training Accuracy:  0.62134
Epoch: 9 Training Loss:  0.9298708845892221     Training Accuracy:  0.67156
Epoch: 10 Training Loss:  0.7656422842036733    Training Accuracy:  0.73096
```

```
Training for additional 10 epochs
Epoch: 11 Training Loss:  0.6104397587763989    Training Accuracy:  0.78626
Epoch: 12 Training Loss:  0.47475555257114305   Training Accuracy:  0.83548
Epoch: 13 Training Loss:  0.381651756975352     Training Accuracy:  0.86714
Epoch: 14 Training Loss:  0.29945904153692143   Training Accuracy:  0.89738
Epoch: 15 Training Loss:  0.2795521038610612    Training Accuracy:  0.90584
Epoch: 16 Training Loss:  0.24246648681895508   Training Accuracy:  0.91824
Epoch: 17 Training Loss:  0.22963857193432197   Training Accuracy:  0.92514
Epoch: 18 Training Loss:  0.2243217466318089    Training Accuracy:  0.9278
Epoch: 19 Training Loss:  0.20362323245314687   Training Accuracy:  0.9337
Epoch: 20 Training Loss:  0.17449226268493306   Training Accuracy:  0.94446
Training Complete for Teacher Model with total 20 epochs
```

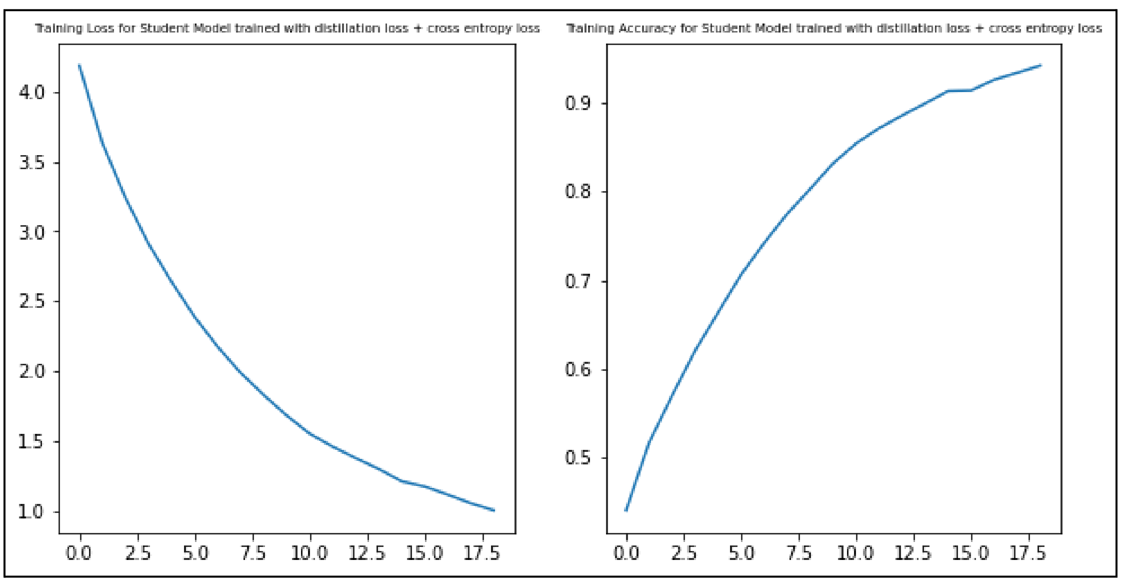
- Then, I saved the Teacher model's trained weights which would then be used to train the Student network using Knowledge distillation.
- Following were the plots obtained when the teacher model was trained.



- Then, I defined a Student PyTorch class for the student model. It inherits from the nn.Module class and defines the architecture of the model in the constructor __init__ method and forward pass in the forward method. The student model consists of 2 convolutional layers with ReLU activation and max pooling, followed by 2 fully connected layers with ReLU activation and the final output layer. The input to the model is a 3-channel image and the output is a 10-dimensional vector, where each element corresponds to a different class.
- Then, instantiated the StudentModel object with Adam optimizer (learning rate = 0.001).
- And defined a distillation loss function where also Cross Entropy is included while back propagation in the student network:
- The function distillation_loss takes in five parameters:
    - y: the predicted outputs of the student model.
    - labels: the true labels for the inputs.
    - teacher_scores: the predicted outputs of the teacher model.
    - T: the temperature used for distillation.
    - alpha: the weighting factor for the distillation loss.
- The function computes the distillation loss using the Kullback-Leibler divergence loss between the softmax outputs of the student and teacher models with the temperature parameter T, multiplied by (T*T * 2.0 * alpha), and added to the cross-entropy loss between the predicted outputs and true labels, multiplied by (1. - alpha). The function returns the total loss.
- Following were the training losses and training accuracies obtained:

```
Epoch: 1 Training Loss:  227.08311822896113      Training Accuracy:  0.27036
Epoch: 2 Training Loss:  4.184126938700371       Training Accuracy:  0.44048
Epoch: 3 Training Loss:  3.6282898958991554      Training Accuracy:  0.51682
Epoch: 4 Training Loss:  3.239944373860079       Training Accuracy:  0.56976
Epoch: 5 Training Loss:  2.910828540392239       Training Accuracy:  0.6205
Epoch: 6 Training Loss:  2.640123134988653       Training Accuracy:  0.66354
Epoch: 7 Training Loss:  2.3888958543157943      Training Accuracy:  0.70624
Epoch: 8 Training Loss:  2.176407312188307       Training Accuracy:  0.74194
Epoch: 9 Training Loss:  1.988972645891292       Training Accuracy:  0.77464
Epoch: 10 Training Loss:  1.8309496119808968     Training Accuracy:  0.80288
Epoch: 11 Training Loss:  1.68374017071541       Training Accuracy:  0.8317
Epoch: 12 Training Loss:  1.5543009984828626     Training Accuracy:  0.8542
Epoch: 13 Training Loss:  1.4613446027726469     Training Accuracy:  0.87136
Epoch: 14 Training Loss:  1.3793550177913187     Training Accuracy:  0.88582
Epoch: 15 Training Loss:  1.3010969735167521     Training Accuracy:  0.89928
Epoch: 16 Training Loss:  1.2129204945491099     Training Accuracy:  0.9133
Epoch: 17 Training Loss:  1.1758446842813126     Training Accuracy:  0.914
Epoch: 18 Training Loss:  1.1178830936741646     Training Accuracy:  0.92602
Epoch: 19 Training Loss:  1.0566234829480692     Training Accuracy:  0.93382
Epoch: 20 Training Loss:  1.005785693750357      Training Accuracy:  0.94192
```

- Then, I plotted the training loss and accuracy for student model trained with distillation loss + cross entropy loss curves.

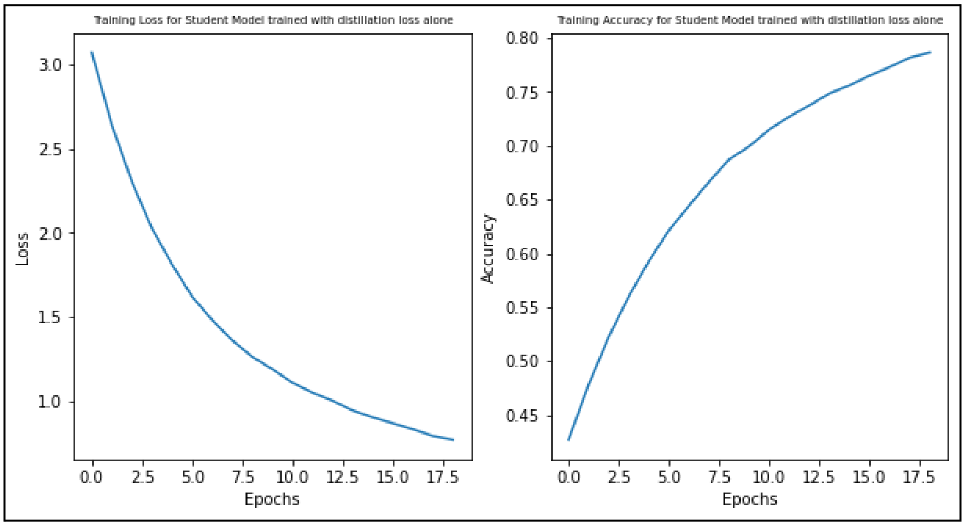


- Then, I saved this trained student model.
- Note that, here we included the Cross Entropy loss as well while the back propagation of the losses.

- Now, we will train a new student model where we would only back propagate the KL Divergence loss between the softmax outputs of the student and teacher models.
- Defined a new distillation loss function.
- The **distillation_loss_ex** function calculates the distillation loss between the student model's predictions y and the teacher model's soft targets teacher_scores, given a temperature parameter T and a mixing coefficient alpha. It applies the KL-divergence loss between the softmax of the predicted outputs and the softmax of the teacher's outputs, multiplied by the temperature squared times 2 times the mixing coefficient. This is done to prioritize the transfer of knowledge from the teacher model to the student model during training. The function doesn't include the cross-entropy loss between the predicted outputs and the true labels, unlike the distillation_loss function.
- Then, I trained the new student model and got the following results.

```
Epoch: 1 Training Loss:  129.23436660290983     Training Accuracy:  0.30932
Epoch: 2 Training Loss:  3.070730686187744      Training Accuracy:  0.42688
Epoch: 3 Training Loss:  2.6408093682945233     Training Accuracy:  0.47808
Epoch: 4 Training Loss:  2.3026988930104637     Training Accuracy:  0.52238
Epoch: 5 Training Loss:  2.024811822130247      Training Accuracy:  0.5601
Epoch: 6 Training Loss:  1.811858680547046      Training Accuracy:  0.59276
Epoch: 7 Training Loss:  1.620711183304067      Training Accuracy:  0.62138
Epoch: 8 Training Loss:  1.4806955406427993     Training Accuracy:  0.64428
Epoch: 9 Training Loss:  1.3605031689719471     Training Accuracy:  0.66652
Epoch: 10 Training Loss:  1.2618382775875003    Training Accuracy:  0.68728
Epoch: 11 Training Loss:  1.1894035380514687    Training Accuracy:  0.6994
Epoch: 12 Training Loss:  1.108454874104551     Training Accuracy:  0.71514
Epoch: 13 Training Loss:  1.049187261735082     Training Accuracy:  0.7271
Epoch: 14 Training Loss:  1.0010833112175201    Training Accuracy:  0.7375
Epoch: 15 Training Loss:  0.943378974562106     Training Accuracy:  0.74862
Epoch: 16 Training Loss:  0.9029414169013957    Training Accuracy:  0.75614
Epoch: 17 Training Loss:  0.8672912651315674    Training Accuracy:  0.7651
Epoch: 18 Training Loss:  0.8319236928849574    Training Accuracy:  0.7731
Epoch: 19 Training Loss:  0.7922762931155427    Training Accuracy:  0.78188
Epoch: 20 Training Loss:  0.7692689586173543    Training Accuracy:  0.78658
Training Complete for Student Model with Knowledge Distillation alone
```

- Again, saved this model’s trained parameters.
- Loss and training accuracy curves:



- Then, I tested all three models (Teacher model, Student 1, and Student 2 models) on the testing dataset.

```
Testing the Teacher Model
Test Loss:  2.8640539434891714   Test Accuracy:  0.5406
```

```
Testing the Student Model with Knowledge Distillation + Cross Entropy Loss
Test Loss:  3.1186087040961543   Test Accuracy:  0.4885
```

```
Testing the Student Model with Knowledge Distillation Loss alone
Test Loss:  3.353041597559482   Test Accuracy:  0.5058
```

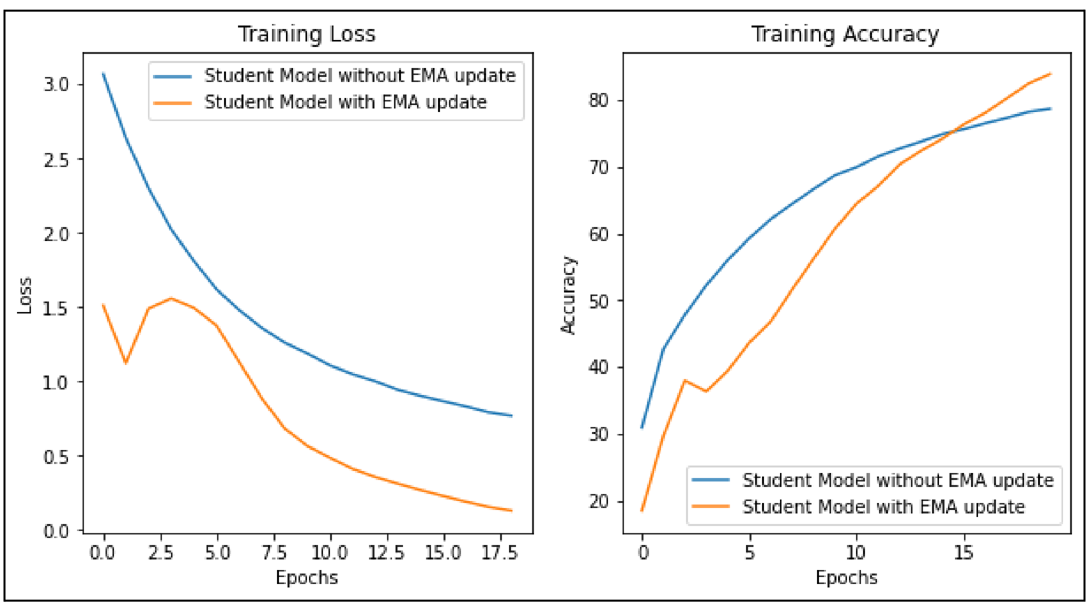
- The reason our test accuracy is low when using distillation_loss_ex function compared to distillation_loss function could be due to the fact that distillation_loss_ex function only applies the distillation loss and does not include the classification loss. In other words, it only encourages the student model to mimic the outputs of the teacher model, but does not explicitly force it to classify the images correctly. On the other hand, distillation_loss function includes both the distillation loss and the classification loss, which helps the student model to both mimic the teacher model and classify the images correctly.
- Therefore, if we use distillation_loss_ex, we may need to increase the number of epochs or adjust the learning rate to allow the student model to learn to classify the images well in addition to mimicking the teacher model. Alternatively, we

could try adjusting the value of the alpha parameter in distillation_loss_ex to balance between the distillation loss and the classification loss.

- Then, I moved on to compare the performance with and without EMA (exponential moving average) updates.
- To compare the impact of ema updates on the performance of the student model trained with knowledge distillation, we can use the same architecture for both the teacher and the student models. For this purpose, we use the architecture of the student model as the common architecture and initialize the parameters of the teacher model to random values.
- Then, I trained two versions of the student model, one with ema updates and the other without ema updates, using the distillation_loss function.
- Then, I defined a **training with EMA function** train_w_EMA that takes in the student network, teacher network, device, trainloader, optimizer, temperature and alpha as inputs. It trains the student network using knowledge distillation with the teacher network's predictions as soft targets, where temperature and alpha are hyperparameters for the distillation loss.
- For each batch of inputs and targets, the function calculates the student network's predictions and the teacher network's predictions. It then calculates the distillation loss using the distillation_loss function, backpropagates the loss and updates the student network's parameters using the optimizer. Additionally, it updates the teacher network's parameters using EMA (Exponential Moving Average) by taking a weighted average of its current parameters and the student network's parameters.
- Then, trained and tested this model and got the following results. (Case where EMA updates were made).

```
Epoch 0: Train Loss=11.563, Train Acc=18.49, Test Loss=2.804, Test Acc=25.52
Epoch 1: Train Loss=1.512, Train Acc=29.69, Test Loss=2.726, Test Acc=30.29
Epoch 2: Train Loss=1.122, Train Acc=37.95, Test Loss=2.525, Test Acc=31.76
Epoch 3: Train Loss=1.490, Train Acc=36.33, Test Loss=3.225, Test Acc=33.08
Epoch 4: Train Loss=1.560, Train Acc=39.44, Test Loss=3.854, Test Acc=37.45
Epoch 5: Train Loss=1.496, Train Acc=43.63, Test Loss=4.452, Test Acc=37.00
Epoch 6: Train Loss=1.377, Train Acc=46.81, Test Loss=5.089, Test Acc=38.09
Epoch 7: Train Loss=1.131, Train Acc=51.61, Test Loss=4.799, Test Acc=38.93
Epoch 8: Train Loss=0.886, Train Acc=56.24, Test Loss=4.680, Test Acc=40.21
Epoch 9: Train Loss=0.685, Train Acc=60.75, Test Loss=4.287, Test Acc=39.18
Epoch 10: Train Loss=0.566, Train Acc=64.48, Test Loss=5.937, Test Acc=42.87
Epoch 11: Train Loss=0.486, Train Acc=67.09, Test Loss=4.692, Test Acc=45.70
Epoch 12: Train Loss=0.411, Train Acc=70.33, Test Loss=5.170, Test Acc=45.55
Epoch 13: Train Loss=0.357, Train Acc=72.38, Test Loss=5.074, Test Acc=45.70
Epoch 14: Train Loss=0.312, Train Acc=74.19, Test Loss=5.605, Test Acc=45.27
Epoch 15: Train Loss=0.268, Train Acc=76.36, Test Loss=5.023, Test Acc=45.92
Epoch 16: Train Loss=0.228, Train Acc=78.07, Test Loss=5.172, Test Acc=46.62
Epoch 17: Train Loss=0.190, Train Acc=80.26, Test Loss=5.399, Test Acc=46.07
Epoch 18: Train Loss=0.155, Train Acc=82.43, Test Loss=6.525, Test Acc=46.30
Epoch 19: Train Loss=0.130, Train Acc=83.87, Test Loss=5.644, Test Acc=47.24
```

- The best accuracy achieved by the student network with EMA updates is 47.24% when trained for 20 epochs.
- And the best accuracy achieved by the student network without EMA updates was 45.76%.
- Comparison between the loss and training accuracy curves (with and without EMA)



- The accuracy and performance are higher in the case of EMA updates because the exponential moving average (EMA) of the student network's parameters is used to update the teacher network's parameters instead of using the student network's parameters directly. This approach helps the teacher network to learn from the more stable and smooth version of the student network's parameters, which can be less sensitive to small, noisy fluctuations in the training process.
- In other words, the EMA updates provide a regularizing effect that helps to prevent overfitting and improve the generalization performance of the teacher network. Additionally, the EMA updates enable the teacher network to adapt to the changes in the student network's parameters gradually, rather than abruptly, which can help to improve the stability and convergence of the training process.

END OF QUESTION 1

—---

Question 01

Aim: Suppose your instructor is working on creating a question set for the DL course. Since we have finished ⅔ of the DL course so far, your instructor wants your help in creating an interesting question.

(a) Based on all the topics we have studied, design 3 interesting and challenging DL questions (Min. 1 question numerical and Min. 1 question theoretical). Simple questions and textbook questions are not allowed.

(b) Run ChatGPT and obtain the answers to your questions. Justify that the answer produced by ChatGPT is correct.

Procedure:

- Question 1: Explain the vanishing and exploding gradient problems in Recurrent Neural Networks (RNNs) and discuss how Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures help mitigate these issues. (Theoretical Question)
- Answer 1 by ChatGPT:

The vanishing and exploding gradient problems are common issues that occur during the training of recurrent neural networks (RNNs), which can prevent the network from effectively learning long-term dependencies.

The vanishing gradient problem occurs when the gradients of the loss function with respect to the weights in the RNN become very small as they propagate backwards through time. This can make it difficult for the network to update the weights in the early layers, as the updates become too small to have a significant impact. As a result, the network may not be able to learn long-term dependencies, and the performance of the model may suffer.

On the other hand, the exploding gradient problem occurs when the gradients become too large, causing the weights to update excessively, leading to numerical instability and poor model performance.

Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures are two types of RNNs that are designed to address these problems. LSTM networks use memory cells, input and output gates, and forget gates to regulate the flow of information through the network. This allows the network to selectively remember or forget information from previous time steps, which can help mitigate the vanishing gradient problem.

GRU networks use fewer parameters than LSTM networks and have fewer gates, making them computationally less expensive. However, they still have the ability to selectively update or forget information from previous time steps, which can help mitigate both the vanishing and exploding gradient problems.

In summary, the vanishing and exploding gradient problems can severely impact the performance of RNNs, but architectures such as LSTM and GRU can help mitigate these issues by selectively remembering or forgetting information from previous time steps, allowing the network to learn long-term dependencies more effectively.

- Is the answer is correct or not?: Yes, the answer given by ChatGPT is correct here, as to solve the very short memory problem of RNNs, LSTM's and GRU's came into place and started getting used widely.

- Question 2: Consider a Convolutional Neural Network (CNN) with the following architecture: Input size is 64x64x3, followed by a convolutional layer with 32 filters of size 5x5, stride 1, and padding 2, then a max-pooling layer of size 2x2, followed by another convolutional layer with 64 filters of size 3x3, stride 1, and padding 1, and finally a fully connected layer with 128 output units. (a) Calculate the total number of trainable parameters in this network. (b) Also, find out the change in the number of parameters if we add a dropout layer with a probability of 0.5 after the max-pooling layer. (Numerical Question)
- Answer 2 by ChatGPT:

  (a) Total number of trainable parameters in the network:

  Convolutional layer 1: 32 filters of size 5x5x3 + 32 bias terms = (5x5x3x32) + 32 = 2400 parameters

  Convolutional layer 2: 64 filters of size 3x3x32 (output from previous layer) + 64 bias terms = (3x3x32x64) + 64 = 18496 parameters

  Fully connected layer: (output from previous layer) x 128 output units + 128 bias terms = (7x7x64x128) + 128 = 3211776 parameters

  Therefore, the total number of trainable parameters in the network is:

  2400 + 18496 + 3211776 = 3238672 parameters

  (b) If a dropout layer with a probability of 0.5 is added after the max-pooling layer, it will not change the number of trainable parameters in the convolutional and fully connected layers. However, it will introduce additional parameters in the form of mask variables that indicate which units are being dropped out.

  The number of mask variables will be equal to the number of units in the max-pooling layer, which is:

  (64/2) x (64/2) x 64 = 16384

  Each mask variable will be a binary value indicating whether the corresponding unit is dropped out or not. Therefore, the total number of additional parameters introduced by the dropout layer will be:

16384 x 1 = 16384 parameters

Hence, the total number of trainable parameters in the network with the added dropout layer will be:

3238672 + 16384 = 3255056 parameters.

- Is the answer is correct or not?: Yes, the answer produced by ChatGPT is correct as it calculated the number of parameters in the CNN and the FC layers correctly and gave the proper reason for “Why dropout layer adds more training parameters rather than reducing the training parameters”.

- Question 3: You are given a dataset of images with varying resolutions and aspect ratios. Explain the steps you would take to preprocess the data for use in a deep learning model, and discuss the trade-offs involved in choosing a specific input size for the model.
- Answer 3 by ChatGPT:

When dealing with a dataset of images with varying resolutions and aspect ratios, it is often necessary to preprocess the data before training a deep learning model. The following steps can be taken for image preprocessing:

  - Resize the images: The images should be resized to a uniform size to ensure that they have the same dimensions. This can be done by cropping the image, adding padding, or using interpolation methods to scale the image. Resizing the images will also reduce the amount of data that needs to be processed during training.
  - Normalize the pixel values: The pixel values of the images should be normalized to a common scale to ensure that they have similar distributions. This can be done by subtracting the mean pixel value of the dataset and dividing by the standard deviation of the pixel values.
  - Data Augmentation: Data augmentation is a technique used to artificially increase the size of the dataset by performing random transformations on the images, such as flipping, rotating, or changing brightness and contrast. This can help prevent overfitting and improve the model's generalization ability.

- Image Preprocessing: Depending on the specific problem, additional preprocessing techniques such as edge detection, noise removal, or color correction may be applied to the images.

When choosing a specific input size for the deep learning model, there are several trade-offs to consider. A larger input size will provide more detailed information to the model, but will also require more computational resources and longer training times. On the other hand, a smaller input size will reduce the amount of data that needs to be processed, but may result in loss of important details and reduce the model's accuracy.

Choosing an appropriate input size for the model depends on the specific problem, the size of the dataset, and the available computational resources. In some cases, it may be beneficial to use multiple input sizes or employ techniques such as image pyramids to process images at different scales.

END OF QUESTION 2

—---