ECS659P: NEURAL NETWORKS & DEEP LEARNING (2022/23 – Semester 2)

Report submitted by: Anushree Vishnoi_220756349

The Problem: CIFAR-10 image classification with CNN
The Task: To build a model on the training set & evaluate it on the test set using Pytorch
The Pipeline: 1. Read the dataset and create the appropriate dataloaders
2. Create and initialise the model
3. Create the loss and optimizer
4. Training the model
5. Final model accuracy on CIFAR-10 Validation (test) Set

The **Goal:** To get the highest possible model accuracy on CIFAR-10 Validation(test) Set.

1. *Read the dataset and create the appropriate dataloaders*

We need 2 thing (a) A way to access the dataset and (b) A way to iterate through it.

- For (a), PyTorch has abstract Dataset class. A Dataset can be anything that has a _len_ function and a _getitem_ function as a way of indexing it into it.
- PyTorch's Tensor Dataset is a Dataset wrapper for tensors. By defining a length and way of indexing , this also gives us a way to iterate, index, slice along the first dimension of a tensor.
- We have created Dataset objects are using "torchvision.datasets.CIFAR10()" , function downloading, transform the data before passing in the model. CIFAR-10 is not provided in tensor form by default, its provided as images hence its pre-processed and converted it into tensors.
- For (b), we use "data.DataLoader( )" ,function to iterate through the dataset, this also provides built-in functionality for: 1. Batching the data, 2.Shuffling the data and Load
- Once the dataloaders are created, they are used to iterate through the training and testing data in batches

2. *Create and initialise the model*

The model architecture consists of a backbone and a classifier/MLP:

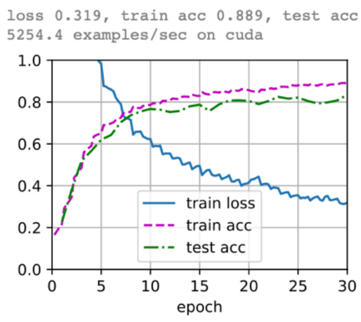
- The backbone consists 5 building blocks - each Nth_block comprising of a linear layer (fully connected layer) and three convolutional layers.
- Classifier - multi-layer perceptron composed of 2 fully connected linear layers.
  - The _init_ method initializes the Nth_Block class from nn.Module class by defining the layers. Each of the Nth block takes x (batch size, channels, height and width) as an input tensor applies adaptive average pooling, flattening, linear transformation and ReLU activation to reduce the input channels to 3 resulting in a vector a.
  - The same input tensor is passed as input to the three convolutional layers and activation function ReLU resulting in conv_1, conv_2 and conv_3.
  - In the forward function, output of the linear layer vector a is split in to a1, a2, a3 and multiplied elementwise (weights of the resulting tensor) with the outputs of the convolutional layers to sum up to obtain single final output as a1Conv_1 + a2Conv_2 + a3Conv_3.The resulting final output tensor acts as the input for the next subsequent block and so on for remaining 4 blocks.
  - The Backbone_ class inherits from the nn.Module base class and defines the architecture of the network comprising Nth_Block class in _init_ method as the building blocks. Each building block is defined by an instance of the 'Nth_Block' class with specific parameters.
  - After each 'Nth_Block', batch normalisation and max pooling are applied to the output feature maps (taking most important feature map).
  - The final output tensor of the last (5th block) acts as the input for the classifier/MLP with ReLU and dropout regularisation which goes through Adaptive Average pooling, flattening to be fed into the MLP

- In the forward method after passing input through all the 5 batches and then the final classifier, the final output logits are obtained of dimension (batch size and 10 classes)

3. ***Create the loss and optimizer***
   - PyTotrch's implementation of Softmax-Cross Entropy loss is used to avoid numerical instabilities
   - 'nn.CrossEntropyLoss( )' is commonly used for multiclass classification problems.It combines 'nn.LogSoftmax( )' and negative log-likelihood loss 'nn.NLLLoss( )' into a single function
   - 'torch.optim.Adam' is an optimization algorithm in PyTorch that is commonly used to train neural network models. It is an adaptive learning rate optimization algorithm that is well suited for large datasets and complex models.
   - It combines momentum with RMS prop
   - The 'Adam' optimizer updates the model parameters based on the gradients of the loss function with respect to the parameters (learning rate= 0.001, weight decay=0.0001, num of epochs=30), similar to other optimization algorithms like stochastic gradient descent(SGD). However, it also includes adaptive learning rate adjustment based on the first and second moment of the gradients.

4. ***Training the model*** **–**

   The model is trained by implementing a 'trainf( )' function. The function is used to train and evaluate the model 'net' using the provided data loaders (train_loader – the data loader for the training dataset and test_loader- the data loader for the test dataset) and given parameters - ('loss' the loss function to be used for training, 'num_epochs' – the number of epochs to train the model for, optimizer – the optimizer used for training and device- the device to use for training - cpu or cuda).
   - During the training process, the function initializes an 'Animator' object for visualizing the training progress.



```
loss 0.319, train acc 0.889, test acc 0.835
5254.4 examples/sec on cuda
```

The 'Animator' object has three curves:

1. Curve for the evolution of training loss – depict the change in the loss function value over the course of training. During the training process, the model is updated by minimising the loss function with respect to the model parameters. As the model makes progress towards minimizing the loss function , the value of the loss function decreases to 0.319

- Each point on the curve represents the average loss value for a batch of training data during one epoch of training. The x-axis of the plot represents the epoch number, while the y-axis represents the value of loss function.
- The plot depicts the loss of change over time and the progress towards minimizing the loss function

    - We can visualize & infer from the above plot that the model is well trained because the training loss curve starts high and decreases over time as the model learns to fit the training data. However it is important to monitor the validation loss as well to make sure that the model is not overfitting to the training data.

      2. For the evolution of training and validation (test) accuracies – depict the change in accuracy of the model over the course of training. During training , the model is updated to improve its ability to classify the input data.The accuracy of the model measures the proportion of correctly classified instances over the total number of instances.
    - Each point on the curve represents the average accuracy value for a batch of training or validation data during one epoch of training. The x-axis of the plot represents the epoch number, while the y-axis represents the value of the accuracy metrics.
    - We can visualise in the above graph that the training accuracy curve starts low from 0.19 (approx.) and increases over time to 0.889 (89% approx.) highlighting the fact that the model learns well to fit the training data.
    - The validation(test) accuracy also shows improvement by increasing from 0.2 (approx.) to 0.835 (84% approx.) and does not level off or decrease at any stages of training to ensure that the model is making progress towards accurately classifying the input data without overfitting to the training data.

      3. All training details including hyper – parameters used.
    - The learning rate is a hyper parameter that determines the step size at each iteration while updating the parameters of the model. A high learning rate may cause divergence and a low learning rate may result in slow convergence and may get stuck in a local minimum.
    - Batch size is another hyperparameter that defines the number of samples that are used to update the parameers of the model in each iteration. A small batch size may result in a noisy estimate of the gradint and slower convergence, while a large batch size may lead to slower training, overfitting and high memory requirements. Therefore by trying different values and monitoring the performance is the only way finding the optimal combination

- Initializes a 'Timer' object for measuring the time it takes to train the model
- Iterates over the training dataset for each epoch and for each iteration:
    - Starts the timer to measure the time it takes to train the current batch of data
    - Sets the model to training mode
    - Clears the gradients of the optimizer
    - Moves the input data and labels to the specified device
    - Computes the forward pass of the model on the input data
    - Computes the loss between the model's predictions and ground truth labels
    - Computes the gradient of the loss with respect to the model parameters using backpropagation
    - Updates the model parameters using the optimizer
    - Computes the training loss and accuracy for the current batch of data
    - Stops the timer and adds the current training loss and accuracy to the 'metric' accumulator
    - If the current iteration is a multiple of 50, adds the current training loss and accuracy to the 'Animator' object for visualisation

5. ***Final model accuracy on CIFAR-10 validation Set is 83.5%***