# ML Assignment 2

Yaswanth Kumar Pattapu

22523

## Low Rank Adaptation

### Training Strategy

The training loop iterates over batches of data from the trainloader. In Batch Processing before the backward pass (gradient calculation), the gradients of all model parameters are cleared. Each batch consists of input data (input), a mask (which denotes the padding), and the target labels (target). The model (teacher_model) is then used to predict the output given the input data and mask. Calculates the loss between the predicted output and the actual target labels using Cross-EntropyLoss. Backpropagates the gradients of the loss with respect to the model parameters. Updates the model parameters based on the computed gradients using the optimization algorithm Adam. Accumulates the total loss for logging or monitoring purposes. Predicted labels are obtained by selecting the index with the highest value along the second dimension of the output tensor (assuming a classification task). Counts the number of correct predictions by comparing the predicted labels (pred) with the actual target labels (target). Accumulates the total number of samples processed.

Throughout the training loop, total_loss, c_pred, and c_total are updated to keep track of the total loss and the number of correct predictions.

### LoRA Implementation

Implementation of custom linear layer using the LoRA decomposition technique.

In the constructor of the class. Stores the number of input features and output features. Specified the rank for the LoRA decomposition. Defined a dropout layer with a dropout of 0.1. Initialized a learnable parameter U of shape (in_features, rank) using PyTorch's Parameter class. This matrix is part of the LoRA decomposition for the input space. Initializes a learnable parameter V of shape (out_features, rank) using PyTorch's Parameter class. This matrix is part of the LoRA decomposition for the output space. Calls the reset_parameters method to initialize the values of U and V.

In reset_parameters(), Initializes the values of U and V using Xavier uniform initialization, which helps in better training convergence by initializing the weights to be neither too large nor too small.

In forward method, Applies dropout to the input tensor to prevent overfitting. Computes the matrix multiplication of the input tensor with the U matrix. Computes the matrix multiplication of the result with the transpose of the V matrix, effectively performing the LoRA decomposition and producing and returning the output tensor.

## LoRA gpt2

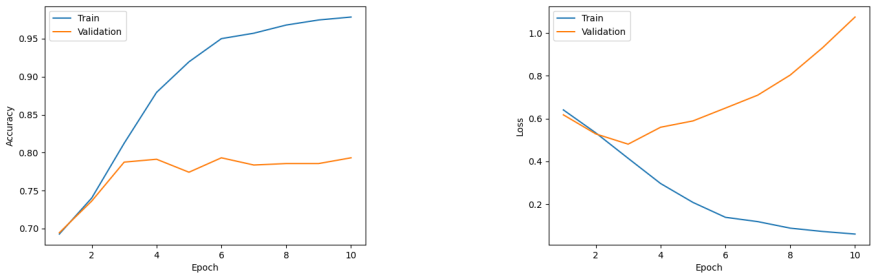
### Hyperparameters

Table 1: Training Hyperparameters

| **Hyperparameter** | **Value** |
|---|---|
| Batch Size | 128 |
| Learning Rate | 0.001 |
| Number of Epochs | 10 |

### Parameters

Here We are using the rank 4.
Total Number of Parameters = 125.03 million
Total Number of Trainable parameters = 0.63 million
Reduction in Parameters = $\frac{Number of Parameters - Number of Trainable parameters}{Number of Parameters}$
$= \frac{25.03M - 0.63M}{25.03M} = 99.50\%$




(a) LoRA gpt2 accuracy

(b) LoRA gpt2 loss

Figure 1: LoRA gpt2 results

The maximum accuracy achieved on Training set when validation set accuracy is maximum is **97.85.%** The maximum accuracy achieved on validation is **79.32%**.

## LoRA gpt2-medium

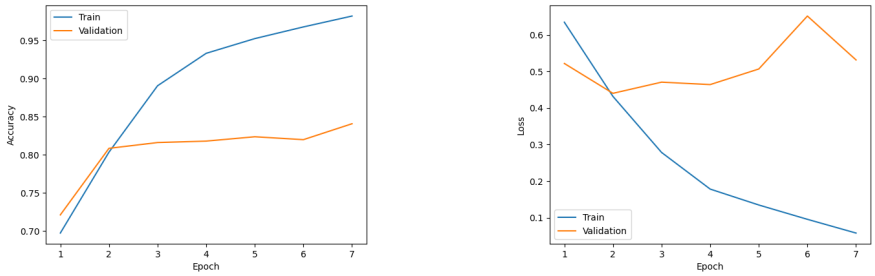
### Hyperparameters

Table 2: Training Hyperparameters

| Hyperparameter | Value |
|---|---|
| Batch Size | 128 |
| Learning Rate | 0.001 |
| Number of Epochs | 7 |

### Parameters

Here We are using the rank 4.
Total Number of Parameters = 356.4 million
Total Number of Trainable parameters = 1.68 million
Reduction in Parameters = $\frac{Numberof Parameters - Numberof Trainable parameters}{Numberof Parameters}$
$= \frac{356.40M - 1.68M}{356.40M} = 99.53\%$




(a) LoRA gpt2-medium accuracy (b) LoRA gpt2-medium loss

Figure 2: LoRA gpt2-medium results

The maximum accuracy achieved on Training set when validation set accuracy is maximum is **98.19.%** The maximum accuracy achieved on validation is **84.06%**.

# Knowledge Distillation

## RNN model

Defined an embedding layer with vocabulary size 50257 and embedding dimension 768. This layer is used to convert input tokens (represented as integers) into dense vector representations. Defined an RNN layer with input size 768,

hidden size 768, 1 recurrent layer. Defines a fully connected (linear) layer that maps the output of the RNN to a two-dimensional output.

In forward method, Defines the forward pass of the module. Applies the embedding layer to the input tensor, converting token indices into dense embeddings. Applies the RNN layer to the embedded input tensor. Takes the output of the RNN from the last time step and applies the fully connected layer to produce the final output. This operation effectively reduces the sequence of RNN outputs to a single vector for each sample in the batch, which is then fed into the linear layer for classification.

## Training

While training we have to put teacher_model in evaluation mode and student_mode in train mode.

Teacher Model Inference performs a forward pass through the teacher model. This ensures the teacher model's weights are not updated during backpropagation (since it's already trained). Stores the predictions made by the teacher model on the current batch.

Student Model Inference performs a forward pass through the student model with the same input and mask used for the teacher. Stores the predictions made by the student model on the current batch.

Distillation Loss Calculation, defines a temperature parameter T, used for softening the teacher's predictions. Higher temperatures lead to softer targets, encouraging the student to learn the overall distribution of the teacher's outputs instead of just the most likely class. Applies softmax on the teacher's output divided by temperature (T). We will apply log-softmax on the student's output divided by temperature (T). This converts the student's raw outputs into log probabilities. We will calculate the **KL divergence** between the student's log probabilities and the softened teacher's target distribution. Finally, it's multiplied by T squared to scale the loss according to the chosen temperature.

Backpropagation and Optimization, Computes the gradients of the loss function with respect to the student model's weights. Updates the student model's weights based on the calculated gradients using the Adam optimizer.

## DistilRNN gpt2

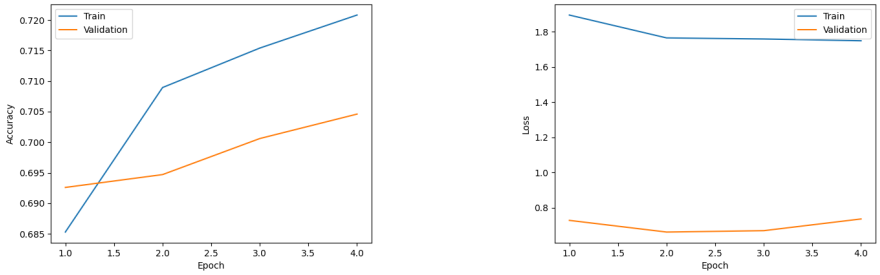
### Hyperparameters

Table 3: Training Hyperparameters

| **Hyperparameter** | **Value** |
|---|---|
| Batch Size | 128 |
| Learning Rate | 0.001 |
| Number of Epochs | 4 |

### Parameters

Here We are using the rank 4.
Total Number of Parameters = 125.03 million
Total Number of Trainable parameters = 0.63 million
Reduction in Parameters = 99.50%




(a) DistilRNN gpt2 accuracy (b) DistilRNN gpt2 loss

Figure 3: DistilRNN gpt2 results

The maximum accuracy achieved on Training set when validation set accuracy is maximum is **71.08.%** The maximum accuracy achieved on validation is **70.06%**.

## DistilRNN gpt2-medium

### Hyperparameters

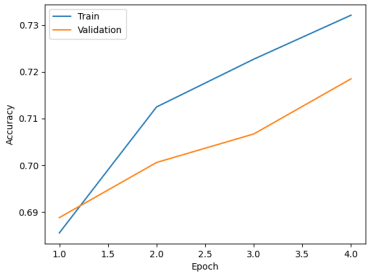
Table 4: Training Hyperparameters

| **Hyperparameter** | **Value** |
|---|---|
| Batch Size | 128 |
| Learning Rate | 0.001 |
| Number of Epochs | 4 |

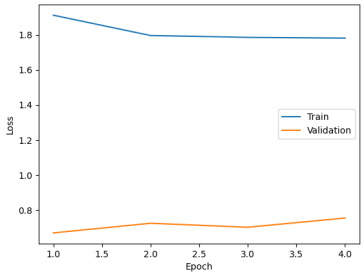
### Parameters

Here We are using the rank 4.
Total Number of Parameters = 356.4 million
Total Number of Trainable parameters = 1.68 million
Reduction in Parameters = 99.53%

The maximum accuracy achieved on Training set when validation set accuracy is maximum is **72.21.%** The maximum accuracy achieved on validation is **71.85%**.




(a) DistilRNN gpt2-medium accuracy




(b) DistilRNN gpt2-medium loss

Figure 4: DistilRNN gpt2-medium results

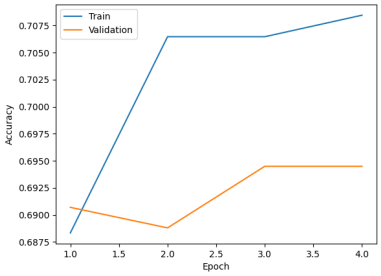
# RNN

## Hyperparameters

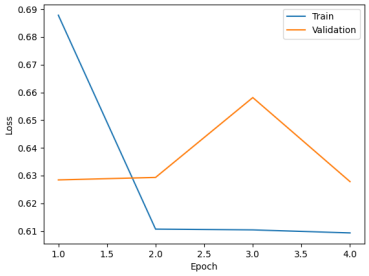
Table 5: Training Hyperparameters

| **Hyperparameter** | **Value** |
|---|---|
| Batch Size | 128 |
| Learning Rate | 0.001 |
| Number of Epochs | 4 |

The maximum accuracy achieved on Training set when validation set accuracy is maximum is **70.85.%** The maximum accuracy achieved on validation is **69.45%**.




(a) RNN accuracy




(b) RNN loss

Figure 5: RNN results

## Conclusion

By using LoRA we can effectively reduce the computational complexity of deep neural networks by approximating the weight matrices with lower-rank matrices. This reduction in complexity leads to faster inference times. Here we achieved reduction of 99.5% with gpt2 without losing much accuracy.

Disitllled RNN will get better accuracy compared to RNN training from scratch. Pre-trained teacher models have already learned meaningful representations from large datasets or tasks, by using the knowledge encoded in the teacher model, the student model can converge faster during training, as it starts with a set of initial parameters that are already close to optimal. Pre-trained teacher models capture valuable knowledge about the task or domain they were trained on. By distilling this knowledge into the student model, it can benefit from insights learned by the teacher, even if the student model has a smaller capacity or is trained on a different dataset.

Training a student model from scratch can be computationally expensive, especially for large-scale models or tasks with limited computational resources. By distilling knowledge from a pre-trained teacher model, the student model requires fewer computational resources and training samples to achieve comparable performance.

We are getting accuracies in the given order $LoRA \geq Distilled\_RNN \geq RNN$.