
Assignment 2 for E0 270: Machine Learning

Chinmay Jain (SR No: 23006) * ¹

1. Introduction

This report details the implementation of two techniques for improving the efficiency and performance of large language models (LLMs): Low-Rank Adaptation (LoRA) and Knowledge Distillation using RNN(Recurrent Neural Network).

2. CoLA Dataset

The CoLA dataset is a collection of sentences annotated with labels indicating their linguistic acceptability or grammatical correctness. It is commonly used in natural language processing (NLP) tasks, particularly in evaluating the performance of models in grammaticality judgment and language understanding tasks. The Corpus of Linguistic Acceptability (CoLA) in its full form consists of 10657 sentences from 23 linguistics publications, expertly annotated for acceptability (grammaticality) by their original authors. The data is split as:

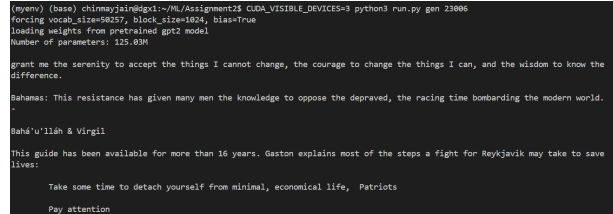
- *in_domain_train.tsv* (8551 lines)
- *in_domain_dev.tsv* (527 lines)
- *out_of_domain_dev.tsv* (516 lines)

2.1. Data Format

The dataset is provided in tsv structured format, having 3 columns. First being the source or origin of the sentence. This code serves as a unique identifier for each sentence and is typically a numerical or alphanumeric value. Second one is the Acceptability Label, it is the binary indicator denoting the acceptability judgment of the sentence. This column contains values 0(Unacceptable) and 1(Acceptable). The last column contains the text of the sentence under evaluation.

3. Problem 0: Generation

We begin with a creative sentence as a prompt, **"grant me the serenity to accept the things I cannot change, the courage to change the things I can, and the wisdom to know the difference."** and pass it through the base GPT2 model for extending the prompt.



```
(myenv) (base) chinmayjain@gs1:~/ML/Assignment2$ CUDA_VISIBLE_DEVICES=3 python3 run.py gen 23006
forcing vocab_size=50257, block_size=1024, bias=True
loading weights from pretrained gpt2 model
number of parameters: 125.83M

grant me the serenity to accept the things I cannot change, the courage to change the things I can, and the wisdom to know the difference.

Bahamas: This resistance has given many men the knowledge to oppose the depraved, the racing time bombarding the modern world.

Bahá'u'lláh & Virgil

This guide has been available for more than 16 years. Gaston explains most of the steps a fight for Reykjavik may take to save lives:

Take some time to detach yourself from minimal, economical life, Patriots

Pay attention
```

Figure 1. Results on running `python run.py gen 23006`

4. Problem 1: Low Rank Adaptation (LoRA)

Low-rank adaptation refers to the modification of matrices or tensors by adjusting their rank to optimize certain objectives or constraints. This concept finds applications in various fields like signal processing, data analysis, and machine learning.

In mathematical terms, low-rank adaptation involves decomposing a given matrix A into two or more matrices of lower ranks, $A \approx UV^T$ or $A \approx U\Sigma V^T$, where U , V are matrices of orthogonal columns, and Σ is a diagonal matrix with singular values. Then, adaptations are made to U , V , or Σ to achieve desired properties.

Technically, low-rank adaptation can be achieved through techniques such as singular value decomposition (SVD), truncated SVD, principal component analysis (PCA), or low-rank matrix completion. These techniques allow for dimensionality reduction, noise reduction, and efficient representation of data while preserving important characteristics.

Low-Rank Adaptation, known as LoRA, involves the freezing of pre-trained model weights and the insertion of trainable rank decomposition matrices into every layer of the Transformer architecture. This process significantly decreases the number of trainable parameters required for subsequent tasks (Hu et al., 2021). In comparison to GPT-2's fine-tuning with Adam, which entails 125.03 million parameters, LoRA can achieve a reduction of 99.5%, resulting in only 0.63 million parameters.

- *Auxiliary Linear Layers:* LoRA injects additional linear layers (called auxiliary LoRA layers) in parallel to the existing linear layers in the pre-trained model.

- **Low-Rank Update Storage:** Instead of directly fine-tuning the original weights, LoRA focuses on capturing the difference between the pre-trained weights and the fine-tuned weights. This difference is assumed to be low-rank, meaning it can be represented by a smaller set of values.
- **Decomposition with L and R Matrices:** This low-rank difference is captured by two matrices, typically denoted by L and R. These matrices have much fewer elements compared to the original weight matrix.
- **Multiplication for Reconstruction:** The original weight matrix after fine-tuning can be reconstructed by multiplying L and R.

```
(myenv) (base) chinmayia@gpt1:~/ML/Assignment2$ CUDA_VISIBLE_DEVICES=3 python3 run.py LoRA 23006
forcing vocab_size=65537, block_size=1024, bias=True
loading weights from pretrained gpt2 model
Some weights of GPT2ForSequenceClassification were not initialized from the model checkpoint at gpt2 and are newly initialized
[["score.weight"]]
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Number of parameters: 125.03M
Reduction: 99.48%
Train Epoch: [1/10], Loss: 0.5926, Accuracy: 70.0503
Val Epoch: [1/10], Loss: 0.5444, Accuracy: 72.1063
Train Epoch: [2/10], Loss: 0.4641, Accuracy: 78.4470
Val Epoch: [2/10], Loss: 0.5400, Accuracy: 72.6755
Train Epoch: [3/10], Loss: 0.3766, Accuracy: 83.2651
Val Epoch: [3/10], Loss: 0.8543, Accuracy: 76.4786
Train Epoch: [4/10], Loss: 0.2878, Accuracy: 87.8952
Val Epoch: [4/10], Loss: 0.5714, Accuracy: 78.7476
Train Epoch: [5/10], Loss: 0.2238, Accuracy: 90.8432
Val Epoch: [5/10], Loss: 0.4647, Accuracy: 77.7949
Train Epoch: [6/10], Loss: 0.2046, Accuracy: 91.5565
Val Epoch: [6/10], Loss: 0.5812, Accuracy: 78.1784
Train Epoch: [7/10], Loss: 0.1667, Accuracy: 93.6266
Val Epoch: [7/10], Loss: 0.6004, Accuracy: 78.9374
Train Epoch: [8/10], Loss: 0.1294, Accuracy: 95.1819
Val Epoch: [8/10], Loss: 0.6772, Accuracy: 78.5579
Train Epoch: [9/10], Loss: 0.1197, Accuracy: 95.8335
Val Epoch: [9/10], Loss: 0.6731, Accuracy: 88.8349
Train Epoch: [10/10], Loss: 0.1345, Accuracy: 94.9713
Val Epoch: [10/10], Loss: 0.6853, Accuracy: 76.1764
```

Figure 2. Results on running `python run.py LoRA 23006`

4.1. Model used as LoRA Auxiliary Linear Layer

The `LoRALinear` class is a PyTorch module designed to implement the LoRA (Low-Rank Adaptation) model. This model is tailored to reduce the number of trainable parameters in a neural network layer while preserving its representational capacity. Let's delve into its details:

- **Input and Output Dimensions:**
 - `in_features`: Specifies the number of input features to the layer.
 - `out_features`: Defines the number of output features from the layer.
 - `rank`: Denotes the rank of the low-rank decomposition.
- **Matrix Decomposition:**
 - **U:** This matrix represents the low-rank decomposition for the input features. It is an instance of the `nn.Linear` module, initialized with the shape `(in_features, rank)`, indicating that it maps from the input feature space to a lower-dimensional space represented by the rank.

- **V:** This matrix represents the low-rank decomposition for the output features. It is also an instance of the `nn.Linear` module, initialized with the shape `(rank, out_features)`, implying that it maps from the lower-dimensional space to the output feature space.

- **Initialization:**

- **Kaiming Uniform for U:** `nn.init.kaiming_uniform_(self.U.weight.data, a=math.sqrt(5))`: This initialization method helps mitigate the vanishing or exploding gradient problem commonly encountered during training. By setting a positive slope parameter to $\sqrt{5}$, Kaiming initialization ensures that the weights are initialized in a way that maintains signal propagation and reduces the likelihood of saturation or collapse in the ReLU-activated layers, leading to more stable and efficient training.
- **Zeros for V:** `nn.init.zeros_(self.V.weight.data)`: The weights of matrix V are initialized to zeros. This initialization simplifies the initialization process, as the low-rank decomposition aims to capture the difference between pre-trained and fine-tuned weights.

- **Scaling Factor:**

- `alpha`: This parameter represents a scaling factor used to control the magnitude of the decomposition. It is set to a predefined value of 64.
- `scale`: Represents the scaling factor applied to the low-rank decomposition. It is calculated as `alpha` divided by `rank`, ensuring that the decomposition is appropriately scaled.

Forward pass in this model for a particular layer is simply a left layer, with the inputs to that layer as `in_feats` to L, which is then fed into R as `out_feats` to obtain right layer. Finally as scaled right layer is the result, where the scaling factor is `Alpha/rank`.

4.2. Results of LoRA for GPT on CoLA

As shown in figure 3, at the end of the Ninth epoch, the training accuracy is 95.44% while validation accuracy is 80.83%.

5. Problem 2: Knowledge Distillation

Knowledge Distillation (KD) is a technique used to transfer knowledge from a large, complex (GPT-2/teacher model) to a smaller, simpler model (RNN/student model). It aims to distill the knowledge contained in the teacher model

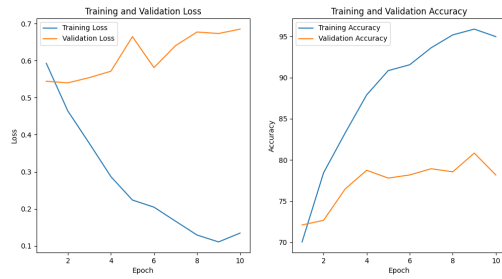


Figure 3. The loss and accuracy plot of the LoRA GPT model on CoLA

into a more compact form, making the student model more efficient while preserving its performance.

5.1. Model Architecture

The student model used for knowledge distillation is called DistilRNN. It consists of an embedding layer, an RNN layer, and a fully connected layer with softmax activation for classification. The architecture is as follows:

1. Embedding Layer: Converts input tokens into dense vectors.
2. RNN Layer: Processes the embedded input sequences to capture temporal dependencies.
3. Fully Connected Layer: Generates output logits for classification using a softmax activation function.

5.2. Implementation:

In our implementation, we utilize a teacher model based on the GPT architecture and a student model called DistilRNN. The DistilRNN model is a simple recurrent neural network (RNN) architecture designed to distill knowledge from the GPT teacher model. We train the DistilRNN model using a combination of distillation loss, calculated using KL divergence, and standard cross-entropy loss. The training procedure involves optimizing the student model's parameters using an Adam optimizer and monitoring the loss and accuracy metrics during training. Finally, we visualize the training progress using loss and accuracy plots to analyze the performance of the DistilRNN model.

5.3. Results of DistilRNN on CoLA

As shown in figure 5, at the end of the third epoch, the training accuracy is 75% while validation accuracy is 69.26%.

```
(myenv) (base) chinmayjain@x1:~/ML/Assignment2$ CUDA_VISIBLE_DEVICES=3 python3 run.py distil 23006
forcing vocab_size=50257, block_size=1024, bias=True
loading weights from pre-trained gpt2 model
Some weights of GPT2ForSequenceClassification were not initialized from the model checkpoint at gpt2 and are newly initialized:
[[score.weight]]
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Number of parameters: 125.03M
Number of trainable parameters: 0.63M
Reduction: 99.16%
/usr/lib/python3.12/site-packages/torch/nn/functional.py:2976: UserWarning: reduction: 'mean' divides the total loss by both the batch size and the support size, 'batchmean' divides only by the batch size, and aligns with the kl div math definition, 'mean' will be changed to behave the same as 'batchmean' in the next major release.
warnings.warn(
Epoch 1/3:
Train Loss: 0.3558
Val Loss: 0.6254
Val Accuracy: 69.07%
Epoch 2/3:
Train Loss: 0.3518
Val Loss: 0.6268
Val Accuracy: 69.07%
Epoch 3/3:
Train Loss: 0.3512
Val Loss: 0.6254
Val Accuracy: 69.26%
```

Figure 4. Results on running python run.py distil 23006



Figure 5. The loss and accuracy plot of the DistilRNN model on CoLA

References

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. Lora: Low-rank adaptation of large language models. *CoRR*, 2021. URL <https://arxiv.org/abs/2106.09685>.