Lab 5: The One Where You Learn to Compress

Introduction

In this laboratory, you will explore model compression techniques, specifically quantization, pruning, and model distillation, by applying these methods to an already pretrained model. This laboratory practice is designed to give you practical experience with model compression, allowing you to reduce the memory footprint and computational requirements of a pre-trained model without significantly impacting its performance. You will compare the effectiveness of different compression techniques and observe how they affect both quantitative results and inference speed.

Throughout this practice, you will:

- Apply various quantization methods to reduce the precision of model weights, including 8-bit, fp16 and 1.58-bit quantization formats.
- Implement pruning to eliminate less significant weights in the model.
- Use model distillation to transfer knowledge from a larger teacher model to a smaller student model, retaining essential patterns while reducing model size.
- Evaluate the compressed models on the `wikitext` dataset for language modeling, comparing perplexity and inference time across the different techniques.

By the end of this lab, you will have a deeper understanding of model compression techniques, including quantization, pruning, and distillation. You will also gain insights into how these methods impact the efficiency and effectiveness of NLP models, preparing you to apply model compression for real-world deployment on resource-limited devices.

Learning Outcomes

Upon completion of this laboratory exercise, you will be able to:

- Implement quantization and distillation techniques to a pretrained model.
- Understand how different model compression techniques affect model performance.
- Understand how different model compression techniques can be applied to light model memory requirements.

Deliverables

1. `quantization.py` - This file must implement the different compression methods.
2. `distillation.py` - This file must implement the distillation losses and learning algorithm.
3. `pruning.py` - This file must implement the structured and unstructured pruning techniques.

4. `model compression and pruning.ipynb` - This file must use a `BERT` model from Google to check for model compression and pruning techniques effectiveness.

These files should be uploaded to Github maintaining the format handled in the laboratory practice.

Exercises

Exercise 1: Honey, I shrunk the model!

In this exercise, you will modify the code to implement different quantization methods for the `QuantizedVector` class, allowing you to quantize vectors and apply them in a mixed-precision matrix multiplication function. This will reduce the memory footprint of the model while maintaining accuracy.

First, you need to implement the quantization scheme for each format (`int8` , `fp16` , `1bit` , and `1.58bit`) in the `quantize()` function of the `QuantizedVector` class:

- For `int8` quantization, compute a scaling factor to transform float values to an `int8` range, allowing the values to retain relative magnitudes. You have already seen this in theory class, referred to as `AbsMax` quantization.
- For `fp16` quantization, convert the weights to half-precision floating-point format.
- For `1bit` quantization, map all positive values to 1 and all negative values to -1.
- For `1.58bit` quantization, use a ternary scheme where weights are mapped to {-1, 0, 1} based on a threshold derived from the mean. Check this link for more information about this quantization.

Your goal is to implement each quantization format within the `QuantizedVector` class and ensure that each format accurately represents the vector while reducing memory usage. Next, complete the `mixed_precision_forward()` function. This function performs matrix multiplication with outlier handling and mixed precision to optimize computation. You have seen this in theory class, referred to as `LLM.int8()` quantization. Follow these steps:

- Identify and extract columns in the input containing outlier values (values above a specified threshold) and isolate the rows in the weight matrix corresponding to these columns. Store the maximum absolute values of each row of x and each column of w.
- Quantize the non-outlier rows of the input separately, and quantize the columns of the weight matrix separately using the `QuantizedVector` class with `int8` format. This allows row-wise and column-wise scaling, supporting mixed precision quantization.
- Perform matrix multiplication: use `INT8` x `INT8` for non-outliers (storing sums in `INT32`), and use `FP16` for the outliers.
- Dequantize the results of the non-outlier computation back to `FP16` and combine them with the outlier results.

Exercise 2: Thanos was Right

In this exercise, you will apply magnitude pruning to reduce the number of weights in a neural network model. Pruning allows you to remove less important weights, which can help make the model more efficient in terms of memory usage and computation while maintaining its performance.

To complete this exercise, you need to:

- Implement unstructured pruning in the `apply_weight_pruning()` function, which removes individual weights based on their magnitude. You will:
  - Identify the smallest magnitude weights in each layer specified by `layer_type` (e.g., `nn.Linear`).
  - Zero out a given percentage of weights based on the `pruning_percentage` parameter.
  - Create a mask to set the identified weights to zero and apply it to the model's weights.

Later on, in the notebook exercise, you will experiment with different pruning percentages to analyze how each pruning setup affects model performance and efficiency.

Exercise 3: Wax On, Wax Off.

In this exercise, you will implement model distillation, a technique where a smaller, student model learns to mimic the behavior of a larger, teacher model. Model distillation allows you to transfer the knowledge from a complex model to a simpler one, helping to reduce the model's size and computational requirements without significantly impacting its performance.

To complete this exercise, you need to:

- Implement the distillation process in the `distill_model()` function. You will:
  - Perform a forward pass through both the teacher and student models for each batch of training data.
  - Calculate two loss components:
    - The cross-entropy loss (`loss_ce`) between the student's predictions and the ground truth labels.
    - The distillation loss (`loss_kl`) between the softened predictions of the student and teacher models, using the `compute_distillation_loss()` function.
  - Combine the losses with a weighting factor, `alpha`, to balance the two loss terms.
  - Backpropagate the combined loss and update the student model's parameters using an optimizer.
- Implement the `compute_distillation_loss()` function to calculate the knowledge distillation loss. This function should:
  - Apply softmax with a temperature to the logits of both student and teacher models to create softened probability distributions.
  - Compute the KL divergence between these distributions, scaled by the temperature squared.

Your goal is to train the student model to closely approximate the predictions of the teacher model, allowing it to learn from the teacher's softened outputs rather than just the ground truth labels.

Exercise 4: Compression Assemble!1

In this exercise, you will explore the impact of various model compression techniques, including quantization and pruning, on a neural network's memory usage, inference speed, and accuracy. By executing the code cells provided, you will observe how different methods reduce model size and affect performance metrics, such as perplexity and inference time.

- Carefully analyze the output of each cell. Pay attention to how the model size and inference time vary with different quantization formats (e.g., FP16, INT8, INT4, NF4) and pruning levels (e.g., 20% and 40% for both unstructured and structured pruning).
- Answer the analysis questions at the end of each section based on the results you observe.

Important: Due to a known issue with the `bitsandbytes` package, the FP16 quantization format may cause significantly slower inference times. This is unexpected behavior as FP16 is generally faster, but the current bug affects its performance here. Take this into consideration when analyzing the results for FP16 quantization, or even avoid executing that part of the code and analyze the rests of the quantization formats results.

Test your work!

Specific test functions have been developed to test your work. These tests are designed to automatically check the correctness of your implementations.

Useful Links

1. Quantization

- **"Differentiable Joint Pruning and Quantization for Hardware Efficiency"**
  This paper presents a method that combines pruning and quantization to optimize neural networks for hardware efficiency.
  `https://arxiv.org/abs/2007.10463`
- **"Automatic Mixed-Precision Quantization Search of BERT"**
  The authors propose an automatic framework for mixed-precision quantization of BERT models, balancing model size and performance.
  `https://arxiv.org/abs/2112.14938`
- **Quantization Recipe by PyTorch**
  Demonstrates how to quantize a PyTorch model to reduce size and improve inference speed while maintaining accuracy.
  `https://pytorch.org/tutorials/recipes/quantization.html`
- **Awesome-Model-Quantization**
  A curated list of papers and resources related to model quantization, providing a broad overview of the field.
  `https://github.com/Efficient-ML/Awesome-Model-Quantization`

2. Knowledge Distillation

- **"Model Compression via Distillation and Quantization"**
  This work explores combining knowledge distillation with quantization to compress models effectively.
  `https://arxiv.org/abs/1802.05668`
- **"Combining Compressions for Multiplicative Size Scaling on Natural Language Tasks"**
  The study investigates the synergistic effects of combining quantization, knowledge distillation, and pruning for model compression in NLP tasks.
  `https://arxiv.org/abs/2208.09684`
- **Knowledge Distillation Tutorial by PyTorch**
  Instructions on modifying model classes to extract hidden representations and enhance lightweight models using complex models as teachers.
  `https://pytorch.org/tutorials/beginner/knowledge_distillation_tutorial.html`
- **KD-Lib: A PyTorch Library for Knowledge Distillation, Pruning, and Quantization**
  An open-source library offering modular implementations of various compression algorithms.
  `https://arxiv.org/abs/2011.14691`

3. Pruning

- **"Combining Weight Pruning and Knowledge Distillation for CNN Compression"**
  This paper discusses a method that integrates weight pruning with knowledge distillation to compress convolutional neural networks.
  `https://openaccess.thecvf.com/content/CVPR2021W/EVW/papers/Aghli_Combining_Weight_Pruning_and_Knowledge_Distillation_for_CNN_Compression_CVPRW_2021_paper.pdf`
- **"Matching the Ideal Pruning Method with Knowledge Distillation for Efficient Model Compression"**
  The authors examine the combined effects of knowledge distillation and different pruning strategies on model compression efficiency.
  `https://www.mdpi.com/2571-5577/7/4/56`
- **Pruning Tutorial by PyTorch**
  A guide on using `torch.nn.utils.prune` to sparsify neural networks and implement custom pruning techniques.
  `https://pytorch.org/tutorials/intermediate/pruning_tutorial.html`
- **Pruning and Quantization in PyTorch Lightning**
  Discusses enabling pruning during training using PyTorch's native pruning implementation.
  `https://lightning.ai/docs/pytorch/stable/advanced/pruning_quantization.html`

4. Comprehensive Resources and Tutorials

- **Neural Network Distiller by Intel Labs**
  A PyTorch environment for prototyping and analyzing compression algorithms, including sparsity-inducing methods and low precision arithmetic.
  `https://intellabs.github.io/distiller/index.html`

- **Model Pruning, Distillation, and Quantization, Part 1 by Deepgram**
  Explores principles behind deep model pruning, distillation, and quantization, outlining steps to apply these techniques.
  `https://deepgram.com/learn/model-pruning-distillation-and-quantization-part-1`

- **Pruning in Keras Example by TensorFlow**
  Demonstrates creating sparse models with the TensorFlow Model Optimization Toolkit API and combining pruning with post-training quantization.
  `https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras`

- **Pruning Preserving Quantization Aware Training (PQAT) Keras Example by TensorFlow**
  An end-to-end example showing the usage of the pruning preserving quantization aware training API.
  `https://www.tensorflow.org/model_optimization/guide/combine/pqat_example`

- **"A Survey on Model Compression for Large Language Models"**
  Covers methods like quantization, pruning, and knowledge distillation, highlighting recent advancements.
  `https://arxiv.org/abs/2308.07633`