CS/ECE 528: Embedded Systems and Machine Learning
Fall 2023

Homework/Lab 3: Software Model Optimization

Assigned: 28 September 2023
Due: 5 October 2023

Instructions:

- Submit your solutions via Canvas.
- Submissions should include your jupyter notebooks in a zip file, with notebooks names q1.ipynb, q2.ipynb, etc. in a single folder. You can include comments in your notebooks to explain your design choices.
- **"Save and checkpoint" your notebook after running your notebook, so that cell outputs are preserved.** If you are using Colab, make sure to 'Save' your notebook after running it, before downloading it and submitting.

Q1. (50 points) Create a model (from scratch without any transfer learning) for the MNIST American Sign Language dataset. The dataset can be found (and downloaded from) here: https://www.kaggle.com/datamunge/sign-language-mnist. The American Sign Language letter database of hand gestures represent a multi-class problem with 24 classes of letters (excluding J and Z which require motion). It is a drop-in replacement for the conventional MNIST model but is more challenging to achieve high accuracy on. Download the training and testing csv files from the Kaggle link and import them into your notebook to work with this dataset. Aim for an accuracy of at least 92% on the test images with your baseline model. Include your notebook file. Note: do not use any customized layers in your model, as these are not supported by some of the optimizations you will explore in later questions. Also if you use BatchNormalization, make sure that it comes immediately before an activation layer, otherwise some of the optimizations may create issues.

Q2. (150 points) Embedded and IoT devices often have limited memory or computational power. Various optimizations can be applied to models so that they can be run within these constraints. Optimizations can potentially result in changes in model accuracy, which must be considered during the application development process. The accuracy changes depend on the individual model being optimized, and are difficult to predict ahead of time. Generally, models that are optimized for size or latency will lose a small amount of accuracy. Depending on your application, this may or may not impact your users' experience. In rare cases, certain models may gain some accuracy as a result of the optimization process. Tensorflow supports many different types of post-training quantization optimizations (https://www.tensorflow.org/lite/performance/post_training_quantization) as well as quantization during training (https://www.tensorflow.org/model_optimization/guide/quantization/training). In this question you will explore the impacts of these optimizations on model size and accuracy.

(a) TensorFlow Lite supports dynamic range quantization, where it statically quantizes only the weights from 32-bit floating point to 8-bits of precision. At inference, weights are converted from 8-bits of precision to 32-bit floating point and computed using floating-point kernels. This conversion is done once and cached to reduce latency. To further improve latency, "dynamic-range" operators dynamically quantize activations (inputs) based on their range to 8-bits and perform computations with 8-bit weights and activations. For the model from Q1, use the TFLiteConverter to convert your trained model into a TensorFlow Lite model, and save it as a "s_mnist.tflite" file. Then perform post training dynamic quantization, and save the resulting file as "s_mnist_quant_dyn.tflite". Determine the accuracy of both models on the test set, and comment on the model sizes of the files. Include your notebook and tflite files.

(b) TensorFlow Lite additionally supports converting activations to 16-bit integer values and weights to 8-bit integer values during model conversion from TensorFlow to TensorFlow Lite's flat buffer format. This mode is called the

"16x8 quantization mode". It can improve accuracy of the quantized model significantly, when activations are sensitive to the quantization, while still achieving reduction in model size. Moreover, this fully quantized model can be consumed by integer-only hardware accelerators. For the model from Q1, now perform 16x8 quantization and save the resulting file as “s_mnist_quant_int16x8.tflite”. Comment on the model size and accuracy, relative to the original model, as well as the dynamic quantized models. Include your notebook and tflite files.

(c) To improve accuracy over post-training quantization, Tensorflow also supports quantization-aware training. For the model from Q1, now perform quantization-aware training and save the resulting file as “s_mnist_quant_aware_training.tflite”. Comment on the model size and accuracy, relative to the original model, as well as the dynamic and int16x8 quantized models. Include your notebook and tflite files.

Q3. (75 points) Clustering, or weight sharing, is another model compression technique. This technique reduces the number of unique weight values in a model, leading to benefits for deployment. It first groups the weights of each layer into N clusters, then shares the cluster's centroid value for all the weights belonging to the cluster. By applying a compression algorithm to the clustered weights, it is possible to significantly reduce memory footprint. The approach can further be combined with quantization to achieve even greater improvements. For more details, refer to the tutorial here: https://www.tensorflow.org/model_optimization/guide/clustering. Starting with your baseline MNIST sign language model from Q1, use weight sharing and quantization to minimize the size of the generated Tensorflow Lite model while still maintaining at least 90% test accuracy for the model. Your score will depend on the size of your final model. You can use any of the Tensorflow Lite quantization approaches from Q1 together with weight sharing to achieve your goal. Considering a different value of ‘number of clusters’ (beyond the value of 16 used in the clustering example) can also provide you with beneficial tradeoffs. Include your notebook, best tflite file, and the gzipped version of the tflite file.

Q4. (75 points) Another approach for model optimization involves weight pruning. Tensorflow provides a prune_low_magnitude() API to train models with removed connections. At a high level, the technique works by iteratively removing (i.e. zeroing out) connections between layers, given a schedule and a target sparsity. The Keras-based API can be applied at the level of individual layers, or the entire model. The approach can further be combined with quantization to achieve even greater improvements. For more details, refer to the tutorial here: https://www.tensorflow.org/model_optimization/guide/pruning. Starting with your baseline MNIST sign language model from Q1, use weight pruning and quantization to minimize the size of the generated Tensorflow Lite model while still maintaining at least 90% test accuracy for the model. Your score will depend on the size of your final model. You can use any of the Tensorflow Lite quantization approaches from Q1 together with weight pruning to achieve your goal. Include your notebook, best tflite file, and the gzipped version of the tflite file.

Q5. (75 BONUS POINTS) It is possible to combine post-training quantization, weight sharing, and pruning optimizations (https://www.tensorflow.org/model_optimization/guide/combine/pcqat_example). Achieve the lowest file size possible for your MNIST sign language model with this approach, while maintaining at least 90% test accuracy for the model. Include your notebook, best tflite file, and the gzipped version of the tflite file.