QGT Tutorial in TF-Keras

ResNet50 on ImageNet, (FP32 activations quantization)

Andrey Melnikov, Latent AI, Princeton, NJ,
December 2020. Draft 1.0

Table of contents

Overview

Summary

We provide an example tutorial using Quantization Guided Training (QGT) to fine-tune and quantize a pretrained ResNet50 model (ImageNet 1000 classes). The purpose is to show how to use the QGT API using hyperparameters to achieve the best accuracy while reducing bit-precision to reduce model size.

In this tutorial, you will

1. Train a ResNet50 model from a pre initialized model
2. Fine tune the training by applying the QGT API

3. Use QGT hyperparameters to further guide the training. The training is of “local” nature, i.e. we are using small learning rates to modify the weights locally, avoiding expensive training from scratch
4. Evaluate results and iterate to continue the fine-tuning process

Introduction

QGT helps modify existing networks with quantization in mind. As part of the training procedure we use a few hyper-parameters: standard ones and QGT related. For example, we use batch-size equal to 16 to increase the number of updates per epoch. We are scheduling a learning rate decrease ten-folds from epoch to epoch to speed up the training process. Along with it, we increase λ_2 parameter tenfold to increase quantization-like characteristics of the weights (imposed through a regularizer).

A successful training is summarized next in two tables for two experiments. The tables contain learning rates and quantization error coefficient, λ_2, along with the number of epochs required to successfully accomplish the training task. The initial model for all the trainings is tf.keras.applications.resnet50.

4 bits per channel QGT training

(here val stands for “validation”, deq for “dequant”, and cr. en. for “cross entropy”)

learn. rate	λ_2	epochs	val loss	val cr.en.	val. acc	deq loss	deq acc
baseline					**74.45**		
1e-3	0.1	1	1.2355	1.1161	73.16	1.3998	67.1
1e-4	1	1	1.1551	1.1208	72.86	1.28	69.28
1e-6	10	1	1.2279	1.1319	72.60	1.1966	71.02
1e-7	100	1	2.8315	1.1003	72.99	1.1296	72.18

2 bits per channel QGT training (First/last layers at FP32)

learn. rate	λ_2	epochs	val loss	val cr.en.	val. acc	deq loss	deq acc
baseline					**74.45**		
1e-3	1	3	1.9301	1.5039	64.63	1.9016	58.14
1e-4	10	3	1.6127	1.541	63.98	1.6166	62.68

We can see that QGT training achieves 72.18% and 62.68% accuracies for models with quantized weights. How this is achieved using the LatentAI SDK is detailed in the following sections.

Setup

To set up a training procedure we build the dataset pipeline and prepare the QGT Docker container with the LatentAI SDK:

- A construction of tensorflow dataset from ImageNet is a standard process using examples from TensorFlow's official site. One can use https://www.tensorflow.org/datasets/api_docs/python/tfds/core/DatasetBuilder, for example. Remember to insert data augmentation into the pipeline, otherwise high validation accuracy (>74%) may not be achieved.
- Run LEIP-SDK docker as per the instructions provided in the LEIP SDK. Additionally, mount the ImageNet tensorflow dataset folder in the container with the -v option. The final Docker image command would look as follows:

```
$ docker start -v <location-of-source-code>:/home -v \
                <location-of-ImageNet-folder>:/data latentaiorg/leip-sdk
```

Training procedures

Once a Docker container with LEIP SDK is installed we can start training. In order to get a high performing 4 bit QGT (per tensor) trained model a scheduling of parameters is required. The parameters are changed according to two directives:

1. Choose a λ_2 value such that the initial quantization-error loss is about 10 times the cross-entropy loss.
2. Choose parameters keeping the initial accuracy value, i.e. at the beginning of each cycle the accuracy is equal to the last step value. The accuracy might decrease during the training to accommodate quantization restrictions.

The initial model might not be trained to the best accuracy. As a result, the first choice of parameters can increase accuracy during the first training cycle. The results are presented in the next table in which each line is obtained using the following command:

```
$ python3 qgt_train.py --epochs 1 --bits 4 --quantize_per_channel --take_batches 500
--dataset_path /data --learning rate l. rate --lambda2 λ_2
```

First, we establish the maximal learning rate so that the training accuracy is preserved.

l. rate	λ_2	#batches	tr loss	tr cr.en.	tr. acc	Remarks
Choosing learning rate						
0	0	500	0.9107	0.7632	79.8	Initial model

1e-1	0	500	4.5933	4.5933	**14.25**	Too **low** tr acc
1e-2	0	500	1.0659	1.0659	72.54	Too **low** tr acc
1e-3	**0**	**500**	0.8578	0.8578	78.47	**Chosen**
1e-4	0	500	0.9219	0.9219	77.42	1e-3 faster
Establishing initial quantization error coefficient λ_2						
1e-3	1	500	**154.063**	0.9317	76.02	Too **high** loss
1e-3	**1e-1**	**500**	22.9989	0.8746	77.49	**Chosen**
1e-3	1e-2	500	**3.7857**	0.8814	77.65	Too **low** loss

At the second step, for the learning rate equal to 0.001, the highest value of λ_2 is searched, which keeps the initial error approximately 10 times the cross-entropy to preserve the accuracy.

Once the parameters learning rate=1e-3 and λ_2=1e-1 are established we use the following command:

```
$ python3 qgt_train.py --epochs 2 --bits 4 --quantize_per_channel --dataset_path /data --learning_rate 1e-3 --lambda2 1e-1
```

The outcome of this training is summarized in the following table for convenience.

train loss	train cross-entropy	train accuracy	val loss	val cross-entropy	val accuracy	dequant loss	dequant accuracy
1.9652	0.7526	80.58	1.2355	1.1161	73.16	1.3998	67.1
0.8586	0.7405	80.83	1.2430	1.1273	73.04	1.4955	65.54

Notice that the training loss decreases significantly from 22 to 1.96 (and further decreases to 0.86 after the second epoch). Also, at the second epoch the difference between the loss and the cross entropy is very small, suggesting that the quantization loss is already very small for this

choice of parameters. Consequently, we can switch to the next step and increase the value of λ_2.

This process is continued until validation and dequant accuracies are similar. The results are summarized in the following table (presented in Overview).

learn. rate	λ_2	epochs	val loss	val cr.en.	val. acc	deq loss	deq acc
baseline					**74.45**		
1e-3	0.1	1	1.2355	1.1161	73.16	1.3998	67.1
1e-4	1	1	1.1551	1.1208	72.86	1.28	69.28
1e-6	10	1	1.2279	1.1319	72.60	1.1966	71.02
1e-7	100	1	2.8315	1.1003	72.99	1.1296	72.18

The corresponding commands are:

```
$ python3 qgt_train.py --epochs 1 --bits 4 --quantize_per_channel --dataset_path /data --learning_rate 1e-3 --lambda2 1e-1
$ python3 qgt_train.py --epochs 1 --bits 4 --quantize_per_channel --dataset_path /data --learning_rate 1e-4 --lambda2 1
$ python3 qgt_train.py --epochs 1 --bits 4 --quantize_per_channel --dataset_path /data --learning_rate 1e-6 --lambda2 10
$ python3 qgt_train.py --epochs 1 --bits 4 --quantize_per_channel --dataset_path /data --learning_rate 1e-7 --lambda2 100
```

Condition for Termination of the Training

The training is stopped once we achieve accuracies of the dequant and the floating point models almost equal on the validation set. In the preceding example those values are 71.18 and 72.99 respectively, and is displayed at the last row of the table.

Training Directives

While training it is important to identify when something is not working properly. Let us summarize a few correct training behaviours, so that if one of them fails during the training, one can modify the hyper parameters.

1. **Total loss decreases** starting from a high value and not increases. If the parameters are chosen correctly (learning rate, λ_2 and especially batch size) the loss will significantly decrease at the first epoch. For example, Resnet 4 bits pc on ImageNet

converges from 22 to 1.96 on the first epoch with batch-size=16, learning rate=0.001 and λ_2=0.01.

2. **Training accuracy is preserved**. For large models, especially while training for a high number of bits (4,8), the training accuracy usually does not decrease more than 1 percent from epoch to epoch. For 2 bits training, the situation is more flexible and a few percentage drop could be expected.
3. **Switch to a higher value of λ_2.** When the training loss is stabilized or when the loss is twice bigger than the cross-entropy one should increase the value of λ_2 and find the highest learning rate (as explained in training procedures).
4. **Decrease learning rate**. If a sharp drop in training accuracy is observed, decrease the learning rate.