Things I learned while running neural networks on microcontroller

By
Saradindu Sengupta

ML Engineer @Nunam

When to deploy on the edge

- Low Latency

Average low inference time, generally in critical system such as safety and healthcare environment where latency is crucial with a slight compromise on accuracy

- Privacy

When the user data doesn’t leave the source is the when privacy leakage scenarios are rare

- Network constraints

In a highly network-constraint environment, non-functional requirements (latency, throughput) takes a hit.

- Throughput

When the inference requests per second are high

Patterns

- Training at the edge
  - Federated learning

1. User data never leaves the device
2. In contrast with a centralised model training and serving, the necessary training and model refresh due to data drift happens on the edge
3. Each deployment can have customized model to suit the use-case and data distribution

Patterns

- Inferencing at the edge
  1. Depending on the type of application, this is the most used scenarios where inference only happen at the edge. This introduces several problems over the lifetime starting with data drift and model update.
  2. The model is trained on a large dataset and only inference happens on the edge
  3. Best suited for deployment on microcontrollers

Model quantization

What

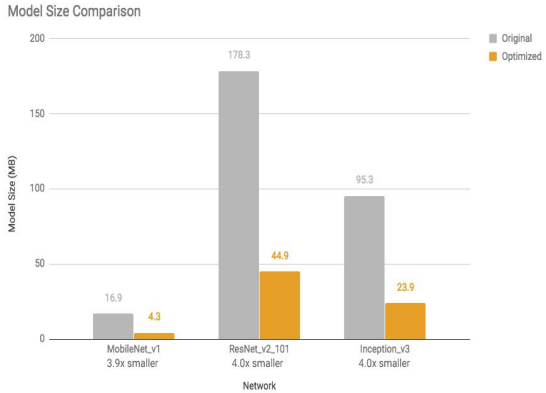
Reduce the floating-point precision to store the model parameters from 32-bit to 16-bit or 8-bit

Why

- Reduce memory footprint
- Reduce memory usage

Drawbacks

- Reduced accuracy dependending on the type of quantization
- (Full integer quantization with int16 activations and int8 weights)[3]
- But how much size is reduced
- All most 4x reduction in models with primarily convolution layers[5]



Types of quantization

- Post-training quantization

1. Quantization of the model parameters from 32-bit floating-point to 16-bit floating-point or 8-bit integer.
2. Different libraries supports different types of post-training quantization for different edge hardwares (mobile,GPU ,microcontrollers and neural-accelerator)
3. For example tflite model optimization toolkit provides 3 different types of post-training quantization methods

Types of Quantization

- Post-training quantization
  1. Post-training float16
     - Suitable for CPU, GPU
     - Accuracy loss: Negligible
  2. Post-training dynamic range
     - Suitable for CPU, GPU
     - Accuracy loss: Least among the rest
  3. Post-training integer
     - Suitable for CPU, GPU and EdgeTPU
     - Accuracy loss: Higher compared to the rest

Types of Quantization

- Quantization-aware training
    1. Happens during model training
    2. 8-bit integers precision
- Which one to choose
    1. Reduction in model size
        a. Post-training float16

    2. Less loss of accuracy

        a. Post-training dynamic range
        b. Post-training integer

Other ways to reduce model size

- Weight Sharing
  1. Cluster weights of each layers into N clusters
  2. Share the centroids of each cluster
  3. Provides 30-50% reduction in model size with negligible loss in accuracy
- Pruning
  1. Remove parameters which are insignificant

Benchmarks

1. Benchmark results of the recent Tiny MLPerf v0.7 [Link][Paper]
2. EdgeTPU benchmark result of different neural net architectures [Link]

References

1. https://www.tensorflow.org/lite/performance/model_optimization#quantization
2. https://www.tensorflow.org/model_optimization/guide/clustering#results
3. https://www.tensorflow.org/lite/performance/model_optimization#full_integer_quantization_with_int16_activations_and_int8_weights
4. https://www.tensorflow.org/model_optimization/guide/quantization/training#results
5. https://blog.tensorflow.org/2018/09/introducing-model-optimization-toolkit.html

Thank You

/in/saradindusengupta

@iamsaradindu

/saradindusengupta