Cheatsheet: Quantization and Precision Tuning for Optimal Inference

Compiled by Ethan Henley, see references for primary sources.

Post-Training Quantization (PTQ)

- Apply INT8 quantization to pre-trained models without retraining.
- Implement per-channel quantization for weights and per-tensor for activations.
- Fine-tune batch normalization statistics post-quantization if accuracy drops.
- Use a small, representative dataset from your target domain for calibration.
- Keep embedding layers and first/last layers in higher precision (FP16) for better accuracy.

Quantization-Aware Fine-Tuning (QAF)

- Start with a pre-trained FP32 model and simulate quantization during fine-tuning.
- Gradually increase quantization noise throughout the fine-tuning process.
- Freeze most of the pre-trained weights and only fine-tune top layers for efficiency.
- Use fake quantization nodes to model INT8 behavior while keeping gradients in FP32.
- Apply learning rate warmup and gradual learning rate decay for stable training.

Mixed Precision for Fine-Tuning

- Use FP16 or BFloat16 for most operations during fine-tuning to speed up the process.
- Implement dynamic loss scaling to prevent gradient underflow.
- Monitor for any accuracy degradation compared to full FP32 fine-tuning.
- Keep master weights and optimizer states in FP32 for stability.
- Use mixed precision libraries (e.g., NVIDIA Apex) for easy integration.

Efficient Inference with Quantized Models

- Deploy fine-tuned and quantized models using inference-optimized runtimes (e.g., ONNX Runtime, TensorRT).
- Use weight caching techniques to speed up repeated inferences.
- Fuse quantized operations (e.g., Conv2D + ReLU) for reduced memory bandwidth.
- Implement efficient int8 GEMM (General Matrix Multiply) operations for faster inference.
- Leverage sparsity in quantized weights for additional speedup if supported by hardware.

Dynamic Quantization for NLP Models

- Quantize weights of pre-trained language models to INT8 offline.
- Keep word embeddings in higher precision (FP16) for better accuracy.
- Implement efficient quantized attention mechanisms for faster inference.
- Dynamically quantize activations during inference to handle varying sequence lengths.
- Use per-token quantization for activations in transformer layers.

Optimizing Attention Mechanisms for Inference

- Quantize query, key, and value projections in transformer layers to INT8.
- Implement efficient int8 matrix multiplication for attention score computation.
- Apply pruning to attention matrices before quantization for sparse inference.
- Keep softmax operations in FP16 to maintain accuracy.
- Use quantized key-value caching for faster autoregressive inference.

Quantization for Transfer Learning

- Quantize the base pre-trained model to INT8 while keeping task-specific layers in higher precision.
- Gradually quantize task-specific layers during later stages of fine-tuning.
- Implement layer-wise adaptive quantization based on sensitivity analysis.
- Fine-tune task-specific layers in FP32 or FP16 on top of the quantized base model.
- Use different quantization granularity for base and task-specific parts of the model.

Calibration Strategies for Pre-Trained Models

- Use a diverse, task-relevant dataset for calibration to capture the dynamic range of activations.
- Apply moving average calibration for models with batch normalization layers
- Perform sensitivity analysis to identify layers that require higher precision.
- Implement per-channel asymmetric quantization for weights to minimize quantization error.
- Use KL divergence or MSE-based methods to determine optimal quantization parameters.

Common Issues & Solutions

My Accuracy Drops After Quantization

- [ ] Check if the calibration dataset is representative of the real-world data
- [ ] Try per-channel quantization instead of per-tensor for weights
- [ ] Increase bit-width for sensitive layers (e.g., first and last layers)
- [ ] Implement fine-tuning after quantization to recover accuracy

My Inference is Slow Despite Quantization

- [ ] Ensure you're using quantization-aware inference engines
- [ ] Check for operator fusion opportunities in your model
- [ ] Profile your model to identify unexpected high-precision operations
- [ ] Use techniques like Quantization Noise to improve generalization

There is Instability in my Mixed Precision Training

- [ ] Ensure proper loss scaling is implemented
- [ ] Monitor for gradient underflow or overflow
- [ ] Start with a lower learning rate and gradually increase
- [ ] Keep a master copy of weights in FP32 for optimizer updates

I Get Poor Generalization of Quantized Models

- [ ] Expand your calibration dataset to cover more diverse cases
- [ ] Implement data augmentation during quantization-aware fine-tuning
- [ ] Try different quantization schemes (symmetric vs asymmetric)

Best Practices

Quantization Workflow

➢ Always establish a strong FP32 baseline before quantization
➢ Start with Post-Training Quantization (PTQ) before moving to Quantization-Aware Training (QAT)
➢ Use gradual quantization: quantize model parts progressively
➢ Keep detailed logs of quantization experiments for comparison

Model Architecture Considerations

➢ Expand your calibration dataset to cover more diverse cases
➢ Implement data augmentation during quantization-aware fine-tuning
➢ Try different quantization schemes (symmetric vs asymmetric)
➢ Use techniques like Quantization Noise to improve generalization

Some Recipes for Your Quantization and Precision-Tuning Cookbook…

Optimizing a Large Language Model for Low-Latency Inference

1. Start with a pre-trained FP32 transformer-based language model
2. Apply mixed precision fine-tuning using FP16 for task-specific adaptation
3. Use Post-Training Quantization (PTQ) to convert the model to INT8
    a. Keep embedding layers and softmax in FP16
    b. Use per-channel quantization for weights, per-tensor for activations
4. Implement efficient INT8 GEMM operations for attention mechanisms
5. Apply quantized key-value caching for faster autoregressive inference
6. Use dynamic quantization for activations to handle varying sequence lengths
7. Deploy using an inference-optimized runtime like ONNX Runtime or TensorRT

Fine-tuning and Quantizing a Vision Transformer for Edge Devices

1. Begin with a pre-trained Vision Transformer (ViT) model
2. Implement mixed precision training (FP16) for initial fine-tuning on target dataset
3. Apply Quantization-Aware Fine-tuning (QAF):
    a. Simulate INT8 quantization during training
    b. Gradually increase quantization noise
    c. Keep first and last layers in FP16
4. Use pruning to reduce model size further (e.g., attention head pruning)
5. Apply layer fusion where possible (e.g., layer normalization folding)
6. Implement efficient quantized attention mechanisms
7. Optimize for target hardware (e.g., mobile GPU, NPU) using tools like ExecuTorch, TorchChat, etc.

Quantizing a BERT-based Model for Question Answering

1. Start with a pre-trained BERT model
2. Fine-tune on your question-answering dataset using mixed precision (FP16)
3. Apply Post-Training Quantization (PTQ):
    a. Use a representative dataset from your QA task for calibration
    b. Quantize to INT8, keeping embeddings and softmax in FP16
4. Implement per-token quantization for activations in transformer layers
5. Use quantized layer normalization for efficiency
6. Apply dynamic quantization for handling variable sequence lengths
7. Optimize the quantized attention mechanism for faster inference
8. Deploy using a quantization-aware inference framework

Transfer Learning and Quantization for Multi-task NLP

1. Begin with a large pre-trained language model (e.g., RoBERTa, T5)
2. Implement task-specific heads for each of your NLP tasks
3. Use mixed precision (FP16) for initial multi-task fine-tuning
4. Apply Quantization-Aware Fine-tuning (QAF) for the shared base:
    a. Quantize the shared transformer layers to INT8
    b. Keep task-specific heads in FP16 initially
5. Gradually quantize task-specific heads during later stages of fine-tuning
6. Use layer-wise adaptive quantization based on sensitivity analysis
7. Implement efficient INT8 inference for the base model
8. Keep a small subset of critical layers (identified by analysis) in higher precision
9. Use quantized knowledge distillation to create a smaller, efficient multi-task model

References

1. *Bhandare, A., et al. (2019). Efficient 8-Bit Quantization of Transformer Neural Machine Language Translation Model. arXiv preprint arXiv:1906.00532.4*
2. *Dong, Z., et al. (2019). HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).*
3. *Gholami, A., et al. (2021). A Survey of Quantization Methods for Efficient Neural Network Inference. arXiv preprint arXiv:2103.13630.*
4. *Jacob, B., et al. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
5. *Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342.*
6. *Nagel, M., et al. (2019). Data-Free Quantization Through Weight Equalization and Bias Correction. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).*
7. *Shen, S., et al. (2020). Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. In Proceedings of the AAAI Conference on Artificial Intelligence.*
8. *Stock, P., et al. (2020). And the Bit Goes Down: Revisiting the Quantization of Neural Networks. In International Conference on Learning Representations (ICLR).*
9. *Wu, H., et al. (2020). Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. arXiv preprint arXiv:2004.09602.*
10. *Zafrir, O., et al. (2019). Q8BERT: Quantized 8Bit BERT. In NeurIPS Workshop on Energy Efficient Machine Learning and Cognitive Computing.*