Deployment on Constrained Devices

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

The "Edge" Challenge

Scenario: You trained a ResNet-50. It's 100MB.

You want to run it on a Raspberry Pi or a Mobile Phone.

Constraints:

1. **Memory**: Device has 2GB RAM, model needs 4GB.
2. **Latency**: Inference takes 5s, user needs <100ms.
3. **Power**: GPU drains battery in 20 mins.
4. **Storage**: App limit is 50MB.

Solution: Model Optimization.

Techniques Overview

graph TD A[Trained Model] --> B[Quantization]; A --> C[Pruning]; A --> D[Knowledge Distillation]; A --> E[Architecture Search]; B --> F[Optimized Model]; C --> F; D --> F; E --> F;

Today's Focus:

1. **Quantization**: Lower precision math.
2. **Pruning**: Removing useless connections.
3. **ONNX**: Efficient Runtime.

Quantization: Theory

Standard Training: Float32 (32-bit floating point).

Quantization: Convert to Int8 (8-bit integer).

Formula:

$$Q(x) = \text{round}\left(\frac{x}{S} + Z\right)$$

- S: Scale
- Z: Zero-point

Impact:

- **Size**: 32 bits -> 8 bits = **4x reduction**.
- **Speed**: Integer math is faster than float math on CPUs.
- **Accuracy**: Minimal drop (<1%) for robust models.

Types of Quantization

1. **Post-Training Quantization (PTQ)**:
   - Train normal Float32 model.
   - Calibrate with small dataset.
   - Convert to Int8.
   - *Easiest*.
2. **Quantization-Aware Training (QAT)**:
   - Simulate quantization *during* training.
   - Model learns to adapt to lower precision.
   - *Best Accuracy*.

Pruning: Theory

Idea: Neural Networks are over-parameterized. Many weights are near zero.
Action: Set small weights to exactly zero.

Structured vs Unstructured:

- **Unstructured**: Random zeros. Good for compression, bad for speed (sparse matrices need hardware support).
- **Structured**: Remove entire channels/filters. Good for speed (smaller matrix).

ONNX: Open Neural Network Exchange

The Universal Bridge

graph LR A[PyTorch] --> D[ONNX Graph]; B[TensorFlow] --> D; C[Scikit-Learn] --> D; D --> E[ONNX Runtime (ORT)]; E --> F[Android]; E --> G[Raspberry Pi]; E --> H[Browser (WASM)];

Why use it?

- **Interoperability**: Train in PyTorch, deploy in C++.
- **Optimization**: ORT applies graph fusions (e.g., Conv+ReLU merging).

Lab Preview

Hands-on Optimization:

1. **Baseline**: Measure size/speed of ResNet-18.
2. **Pruning**: Use `torch.nn.utils.prune` to remove 30% of weights.
3. **Quantization**: Apply PyTorch dynamic quantization.
4. **ONNX Export**: Convert and run with ONNX Runtime.

Goal: Make the model 2x faster and 4x smaller!