Distributed Systems for Machine Learning  
Neil Agarwal  
April 13, 2022

Fundamentals of Distributed Systems

Fault Tolerance

Massive Parallelization

Synchronization/Consensus

Lecture Goals

• Define systems for machine learning
• Understand challenges and considerations in designing such systems
• Explore a widely deployed system for ML (TensorFlow)
Agenda

• What’s led to the success of machine learning?
• What’s a typical machine learning job?
• Systems for machine learning
  • Definition
  • Challenges:
    • How to handle distributed computation?
    • How to support execution in diverse environments/heterogeneous hardware?
    • Developer interface: Tradeoff between flexibility and efficiency
• Case Study: TensorFlow

The Success of Machine Learning Today

Object detection

Autonomous vehicles

Language Modeling

Game playing

The Ingredients in ML Success

ML Model

ML systems bridge model, data, and hardware

Data

ImageNet, Kaggle, Flickr, NetFlix, ...

Hardware

GPUs, TPUs, Supercomputers, FPGAs

RedNet, Transformers, Graph Neural Networks, Mixture-of-Experts, ...

What is a typical machine learning job?
Note on Training vs Inference

In today’s discussion, will focus on training!

Will briefly discuss inference later!

Machine Learning Training Pipeline

1. Users select a model architecture!
   - Typically Deep Neural Networks (DNNs)
   - Others types/variants: Recurrent Neural Networks, Graph Neural Networks, etc.

Machine Learning Training Pipeline

2. Users provide a large labeled dataset
   - images + classification labels
   - images + captions
   - sentence + sentimental analysis

Machine Learning Training Pipeline

3. Train the model!
   - sequentially process the dataset
   - learn using a form of gradient descent (via backpropagation)
Machine Learning Training Pipeline

System for Machine Learning

- Abstracts away the underlying systems complexities of executing the training of machine learning models
- Design Considerations
  - Main
    - How to handle distributed computation?
    - How to support execution in different environments and on heterogeneous hardware?
    - What’s the right interface for users that still supports customizations?
  - Others
    - How to support different non-deterministic control flows (e.g., recurrent neural networks?)
Design Consideration #1: How to handle distributed computation?

- Why perform distributed machine learning in the first place?
  - Trends
    - Increasingly large datasets: millions/billions of images/samples
    - Increasingly large DNNs: more layers, more parameters
  - For example, GPT-3 is a language model with about 175 billion parameters, trained on 45 Terabytes of text data
  - Too slow to process on a single machine
    - The entirety of a DNN (and its weights/grads) cannot fit on a single machine!

Distributed ML: Data Parallelism

1. Partition training data into batches
2. Compute the gradients of each batch on a GPU
3. Aggregate gradients across GPUs

Challenge: All the workers must communicate with the centralized server for weight updates.

Distributed ML: Model Parallelism

- Split a model into multiple subgraphs and assign them to different devices
- Transfer intermediate results between devices

Challenge: How split model across machines?

Distributed ML Considerations

- Placement of computation across machines
- Communication of intermediate data between machines
- Fault tolerance! What happens if a machine crashes?
- Synchronization
Design Consideration #2: How to support execution in different environments and on heterogeneous hardware?

- Various types of compute settings:
  - datacenter (thousands of CPUs, GPUs)
  - workstation set up (single CPU, few GPUs)
  - laptop

- Heterogeneous Hardware: GPUs, TPUs, FPGAs
  - Each is optimized for different tasks
  - Optimal memory placement/computation configuration depends on type

Define Once + Run Everywhere

Design Consideration #3: What’s the right interface/programming model for users that still supports customizations?

- Support different user requirements
  - novice user: uses several default settings
  - expert user:
    - define new layers
    - try new training algorithms
    - introduce new optimizations

- Want easy-to-use interface while still being customizable

System for Machine Learning Recap

- Abstracts away the underlying systems complexities of executing the training of machine learning models
- Design Considerations
  - How to handle distributed computation?
  - How to support execution in different environments and on heterogeneous hardware?
  - What’s the right interface for users that still supports customizations?

Case Study: TensorFlow
TensorFlow

- Developed by Google Brain
  - successor to DistBelief
- A system widely used in industry/academia for distributed machine learning!
- Main Contributions
  - Support for large-scale distributed training
  - Modular architecture that decouples optimizations of the machine learning model from the infrastructure itself
  - supports diverse compute environments, heterogeneous hardware
  - Very user-friendly: Python interface that enables customizability across the stack

TensorFlow System Design

Data Preprocessing

Distributed DNN Training

Model checkpointing

Phase 1: Define an ML model as a dataflow graph

Phase 2: Execute an optimized version of the graph

TensorFlow: Example

```
# 1. Construct a graph representing the model:
#  y = tf.placeholder(tf.float32, [BATCH_SIZE, 10])  # placeholder for input.
#  y_ = tf.placeholder(tf.float32, [BATCH_SIZE, 1])  # placeholder for labels.
W_1 = tf.Variable(tf.random_uniform([10, 10]))  # 784x100 weight matrix.
h_1 = tf.sigmoid(tf.matmul(y, W_1) + b_1)  # 100-element bias vector.
layer_1 = tf.nn.relu(tf.matmul(y, W_1) + b_1)  # Output of hidden layer.
W_2 = tf.Variable(tf.random_uniform([100, 10]))  # 100x10 weight matrix.
h_2 = tf.sigmoid(tf.matmul(layer_1, W_2) + b_2)  # 10-element bias vector.
U_3 = tf.matmul(layer_1, W_2) + b_2  # Output of linear layer.

# 2. Add nodes that represent the optimization algorithm:
#  loss = tf.nn.softmax_cross_entropy_with_logits(layer_2, y)
#  train_op = tf.train.AdamOptimizer(learning_rate=0.01).minimize(loss)
```

Systems for Machine Learning Inference

- Application/customer facing: stringent latency targets
- Deal with interactions with network
- Caching opportunities
- Model compression/pruning
  - tradeoff between speed and accuracy
- Edge deployments
Active Research Areas in ML+Systems

• Application-specific optimizations for machine learning (e.g., video analytics)
• ML for systems (e.g., learned databases, compilation optimizations)
• New computation models (spot instances, serverless computing, programmable networks)

Takeaways

• **Systems for machine learning** are critical to the success of machine learning
• Handle the systems challenges involved in running large-scale distributed machine learning
  • e.g., fault tolerance, consistency, heterogeneous hardware, communication
• Provide an easy-to-use interface for developers while still enabling significant levels of customizability