FUNDAMENTALS OF DISTRIBUTED SYSTEMS

Fault Tolerance  Synchronization/Consensus  Massive Parallelization

Also apply to systems for machine learning!
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
  – 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 contents are based on Neil Agarwal's lecture at Princeton and Zhihao Jia's ML materials
THE SUCCESS OF MACHINE LEARNING TODAY

Object detection

Autonomous vehicles

Language Modeling

Game playing
THREE KEY INGREDIENTS IN ML SUCCESS

ML Model

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

Systems for ML

ML systems bridge model, data, and hardware

Data

ImageNet, Kaggle, Flickr, Netflix, ...

Hardware

GPUs, TPUs, Supercomputers, FPGAS
What is a typical machine learning job?
NOTE ON TRAINING VS INFEERENCE

In today’s discussion, will focus on training!
Will briefly discuss inference later!
– 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 + sentiment 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 complexity of training machine learning models

Don’t worry about it.

Model & Dataset

Trained Model

Phew.

Synchronization

Graph Optimizations

Tolerance

Hardware Interface

Data + Compute Layout
SYSTEM FOR MACHINE LEARNING

— 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?
DESIGN CONSIDERATION #1: 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
  — Is trained on 45 Terabytes of text data

Too slow to process on a single machine

The entirety of a DNN (and its weights/gradients) cannot fit on a single machine!
DISTRIBUTED ML: DATA PARALLELISM

1. Partition training data into batches

ML Model

Training Dataset

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

- GPU 1
- GPU 2
- GPU N

2. Compute the gradients of each batch on a GPU

3. Aggregate gradients across GPUs

Gradients Aggregation
DISTRIBUTED ML: MODEL PARALLELISM

— Split a model into multiple subgraphs and assign them to different 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: SUPPORT HETEROGENEOUS ENV?

— 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
DESIGN CONSIDERATION #3: INTERFACE FOR 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
TENSORFLOW: EXAMPLE

Phase 1: Define an ML model as a dataflow graph

```python
# 1. Construct a graph representing the model.
x = tf.placeholder(tf.float32, [BATCH_SIZE, 784])  # Placeholder for input.
y = tf.placeholder(tf.float32, [BATCH_SIZE, 10])  # Placeholder for labels.

W_1 = tf.Variable(tf.random_uniform([784, 100]))  # 784x100 weight matrix.
b_1 = tf.Variable(tf.zeros([100]))  # 100-element bias vector.
layer_1 = tf.nn.relu(tf.matmul(x, W_1) + b_1)  # Output of hidden layer.

W_2 = tf.Variable(tf.random_uniform([100, 10]))  # 100x10 weight matrix.
b_2 = tf.Variable(tf.zeros([10]))  # 10-element bias vector.
layer_2 = 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(0.01).minimize(loss)
```

Phase 2: Execute an optimized version of the graph

```python
# 3. Execute the graph on batches of input data.
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())  # Connect to the TF runtime.
    sess.run(tf.initialize_variables())  # Randomly initialize weights.
    for step in range(NUM_STEPS):
        x_data, y_data = ...
        sess.run(train_op, {x: x_data, y: y_data})  # Train iteratively for NUM_STEPS.
```

26
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)
GAME: GANDALF
https://gandalf.lakera.ai/
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

— Next class: Reliability
ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON COURSE MATERIALS SHARED BY PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS, PROF. MICHAEL FREEDMAN, PROF. KYLE JAMIESON, PROF. WYATT LLOYD AND PROF. ROXANA GEAMBASU. MANY APPRECIATIONS FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.