FAQs

- Please sign up for the PA2 demonstration
  https://www.signupgenius.com/go/30E0E4EACA29ABF0- pa2demonstration
- This link is available in today’s canvas announcement as well
- Wednesday 4/3: workshop
- Please submit your review/slides by 2 hours before the class starts

Today’s topics

- Distributed optimization for ML

This material is built based on

- Jeffrey Dean and Greg S. Corrado and Rajat Monga and Kai Chen and Matthieu Devin and Quoc V. Le and Mark Z. Mao and Marc Aurelio Ranzato and Andrew Senior and Paul Tucker and Ke Yang and Andrew Y. Ng, Large Scale Distributed Deep Networks, 2012, NIPS
- Martin Abadi and Paul Barham and Jianmin Chen and Zhifeng Chen and Andy Davis and Jeffrey Dean and Matthieu Devin and Sanjay Ghemawat and Geoffrey Irving and Michael Isard and Manjunath Kudlur and Josh Levenberg and Rajat Monga and Sherry Moore and Derek G. Murray and Benoit Steiner and Paul Tucker and Vijay Vasudevan and Pete Warden and Martin Wicke and Yuan Yu and Xiaoqiang Zheng, TensorFlow: A system for large-scale machine learning, 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016
Revisit Neural Networks with a handwriting recognition example

- Training your model with a large number of handwritten digits to recognize them.

Perceptron

- A perceptron takes several binary inputs, \( x_1, x_2, x_3 \ldots \) and produces a single binary output.
- Weights \( w_1, w_2, w_3 \ldots \), real numbers expressing the importance of the respective inputs:

\[
\text{output} = \begin{cases} 
0 & \text{if } \sum w_i x_i \leq \text{threshold} \\
1 & \text{if } \sum w_i x_i > \text{threshold}
\end{cases}
\]

Perceptron with layers

- First layer of perception.
- Second layer of perception.

Recognizing individual digits

- Suppose we have a multilayer perceptron (MLP).
- “Hidden” layer: neither inputs nor outputs.

Simple example: “Recognize number “9”!”

-Encoding the intensities of the image pixels into the input neurons - e.g., if the image is a 64 by 64 grayscale image, then 4,096 = 64 x 64 input neurons with the intensities scaled between 0 and 1.

-Output layer:
- Less than 0.5: input image is not a “9”
- Greater than 0.5: input image is a “9”
Recognizing individual digits

Training with SGD

- Suppose that our input images are 28 x 28 dimensional vector
- Output will be 10 dimensional vector
- For digit ‘6’, y(x)=(0,0,0,0,0,0,1,0,0,0)\textsuperscript{T}
- The training algorithm should find weights and biases
- So that the output from the network approximates y(x) for all training inputs x
- Cost function

$$C(w,b) = \frac{1}{2n} \sum_{i=1}^{n} ||y_i - \hat{y}_i||^2$$

- Here, w denotes the collection of all weights and b all the biases.
  n is the total number of training inputs and \(\hat{y}_i\) is the vector of outputs from the network when x is input.
GEAR Workshop II | Scalable Computing Models
Distributed Optimization Algorithms
Asynchronous Parallel Optimization
DistBelief: 1. Downpour SGD

Downpour SGD [1/3]
- Objectives of Downpour SGD
  - Applying SGD to extremely large datasets
- A variant of asynchronous stochastic gradient descent
  - Uses multiple replicas of a single DistBelief model
- This approach is asynchronous in two distinct aspects
  - The model replicas run independently of each other
  - The parameter server shards also run independently of one another

Downpour SGD [2/3]
- Divide the training data into a number of subsets and run a copy of the model on each of these subsets
- The models communicate updates through a centralized parameter server
  - Keeps the current state of all parameters for the model
  - Shards parameters across many machines
  - e.g., if we have 10 parameter server shards, each shard is responsible for storing and applying updates to 1/10th of the model parameters

Downpour SGD [3/3]
- Step 1: Before processing each mini-batch, a model replica asks the parameter server service for an updated copy of its model parameters
  - Each machine needs to communicate with just the subset of parameter server shards that hold the model parameters relevant to its partition
- Step 2: After receiving an updated copy of its parameters, the DistBelief model replica processes a mini-batch of data to compute a parameter gradient
- Step 3: Sends the gradient to the parameter server
  - Then, applies the gradient to the current value of the model parameters

Reducing the communication overhead of Downpour SGD
- Limit each model replica to request updated parameters only every $n_{fetch}$ steps and send updated gradient values only every $n_{push}$ steps
  - where $n_{fetch}$ might not be equal to $n_{push}$
- Traditional distributed SGD
  - $n_{fetch} = n_{push} = 1$

Fault tolerance with Downpour SGD
- More robust to machines failures
- Synchronous SGD with failures
  - If one machine fails, entire training process is delayed
- Asynchronous SGD with failures
  - If one machine fails, other model replica will continue processing training and updating model parameters

http://www.cs.colostate.edu/~cs535
Failures vs. Stochasticity
- With DistBelief, no guarantee that:
  - There were the same number of updates in each shard of parameters
  - There were updates applied in the same order
- There are subtle inconsistencies in the timestamps of parameters
- Relaxing consistency requirement is effective to enhance stochasticity

Improving Downpour SGD with Adagrad
- $\eta_i, K = \frac{\gamma \sum \Delta w_i, K^2}{\sqrt{\sum \Delta w_i, K^2} + \epsilon}$
- The value of $\gamma$:
  - The constant scaling factor for all learning rates
  - Generally larger than the best fixed learning rate used without Adagrad
- The use of Adagrad:
  - Extends the maximum number of model replicas that can productively work simultaneously
  - Eliminates stability concerns
  - Combined with a practice of "warmstarting" model training with only a single model replica before unleashing the other replicas
- These learning rates are computed only from the summed squared gradients of each parameter
- Adagrad can be implemented locally within each parameter server shard

Sandblaster L-BFGS
- Batch methods works well in training small deep networks
- Sandblaster
  - Batch optimization framework
  - Implementation of L-BFGS
  - Distributed parameter storage and manipulation

GEAR Workshop II | Scalable Computing Models
Distributed Optimization Algorithms
Asynchronized Parallel Optimization
DistBelief: 2. Sandblaster L-BFGS

Sandblaster L-BFGS: Coordinator
- Coordinator based system architecture
  - No direct access to the model parameters
  - Issues commands drawn from a small set of operations
  - e.g., dot product, scaling, coefficient-wise addition, multiplication
  - Performed by each parameter server shard independently
  - Results are stored locally on the same shard
Sandblaster L-BFGS: Parameter Server
- Performs small set of operations
- Stores results from the local computation
- Stores additional information
- History cache
  - Provides scalability for the large (e.g. billion parameters) scale learning without incurring the communication overhead
  - To a single central server

Sandblaster L-BFGS: Load balancing
- Typical parallelized implementations of L-BFGS
  - Local machines are responsible for computing the gradient on a subset
  - Results are sent back to the central server
  - Or aggregated via a tree structure
- Performance is limited by the latency of stragglers!

Sandblaster L-BFGS: Load balancing
- Coordinator assigns each of the N model replicas a small portion of work
  - Much smaller than 1/Nth of the total size of a batch
  - Assigns replicas new portions whenever they are free
- Faster model replicas do more work than slower replicas
- At the end of a batch
  - Coordinator schedules multiple copies of the outstanding portions
  - Uses result from whichever model replica finishes first

Sandblaster L-BFGS vs. Downpour SGD
- Downpour SGD
  - Requires relatively high frequency, high bandwidth parameter synchronization with the parameter server
- Sandblaster L-BFGS
  - Workers only fetch parameters at the beginning of each batch
  - Updated by the coordinator
  - Workers only send the gradients every few completed portions
  - To protect against replica failures and restarts

Performance Evaluation
- Left: Training accuracy (on a portion of the training set) for different optimization methods
- Right: Classification accuracy on the hold out test set as a function of training time
  - Downpour and Sandblaster experiments initialized using the same ~10 hour warm-start of simple SGD
TensorFlow: successor to DistBelief
- DistBelief has been used in Google since 2011
  - Parameter server architecture
- Fully connected layer
  - Multiplies its input by a weight matrix, adds a bias vector, and applies a non-linear function (such as a sigmoid) to the result
  - The weight matrix and bias vector are parameters
- A loss function is a scalar function
  - Quantifies the difference between the predicted value (for a given input data point) and the ground truth

TensorFlow: New features
- Defining new layers
  - Implemented DistBelief layers as C++ classes
- Refining the training algorithms
  - SGD update rules
- Defining new training algorithms
  - DistBelief was suitable for the simple feed-forward style NNs
    - Many of Neural Networks were not applicable
      - e.g. RNN, Adversarial networks, reinforcement learning
- Support GPU acceleration
- Support scale down
  - E.g. Single GPU-powered workstation for development

TensorFlow: Design Principles
- A simple dataflow-based programming abstraction
  - Users can deploy applications on distributed clusters, local workstations, mobile devices
- Dataflow graphs of primitive operators
  - Both TensorFlow and DistBelief use a dataflow representation
  - DistBelief model comprises relatively few complex “layers”
  - TensorFlow model represents individual mathematical operators
    - Such as matrix multiplication, convolutions, etc.
    - Users can compose their new application easily

TensorFlow: Deferred execution
- Phase 1: defining the program (e.g. neural network to be trained and update rules)
- Phase 2: executing an optimized version of the program on the set of available devices
- Phase 2 (execution phase) is optimized by deferring this phase until the entire program is available
  - E.g. issues a sequence of kernels to the GPU using the graph’s dependency without waiting for intermediate results
  - High GPU utilization

TensorFlow: Dataflow graph elements
- Operations
  - Computation at vertices
- Tensors
  - Values that flow along edges

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 7
### Tensors

- Data is modeled as tensors (n-dimensional arrays)
- Elements having one of a small number of primitive types
  - Int32, float32, or string
  - E.g. a matrix multiplication with two 2-D tensors, produces a 2-D tensor

- Representing tensors
  - Dense tensor
    - Encoding the data into variable-length string elements
  - Sparse tensor
    - A tuple of dense tensors
    - An n-D sparse tensor with m non-zero elements can be represented in coordinate-list format and a length-m vector of values

### Operations

- Takes m ≥ 0 tensors as input
- Produces n ≥ 0 tensors as output
- An operation has a named “type”
  - May have zero or more compile-time attributes that determine its behavior
  - E.g. Const, MatMul, or AnyOp
  - E.g. Const
    - The simplest operation
    - No inputs and a single output
    - A compile-time attribute
      - E.g. AddN
    - Sums multiple tensors of the same element type, and it has a type attribute and an integer attribute that define its type signature