Techniques and Systems for Training Large Neural Networks Quickly

Jeff Dean
Google

Joint work with many colleagues at Google
How Can We Build More Intelligent Computer Systems?

Need to perceive and understand the world

Basic speech and vision capabilities
  Language understanding
  User behavior prediction

...
How can we do this?

• Cannot write algorithms for each task we want to accomplish separately
• Need to write general algorithms that learn from observations

Can we build systems that:

• Generate understanding from raw data
• Solve difficult problems to improve Google’s products
• Minimize software engineering effort
• Advance state of the art in what is possible
Functions Artificial Neural Nets Can Learn

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixels: [image of a lion]</td>
<td>“lion”</td>
</tr>
<tr>
<td>Audio: [audio spectrum]</td>
<td>“pa lo ah! toe res taur aun ts”</td>
</tr>
<tr>
<td>&lt;query, doc&gt;</td>
<td>P(click on doc)</td>
</tr>
<tr>
<td>“Hello, how are you?”</td>
<td>“Bonjour, comment allez-vous?”</td>
</tr>
<tr>
<td>Pixels: [image of a child]</td>
<td>“A close up of a small child holding a stuffed animal”</td>
</tr>
</tbody>
</table>
Same Underlying System Works for Many Problems

- Nice property: same general approach works for many problems/domains

- Same underlying infrastructure can be used to train and use many kinds of models
Plenty of Data

- **Text**: trillions of words of English + other languages
- **Visual**: billions of images and videos
- **Audio**: thousands of hours of speech per day
- **User activity**: queries, result page clicks, map requests, etc.
- **Knowledge graph**: billions of labelled relation triples
- ...
More Data + Bigger Model = Better Results

• True across wide range of areas

• Many problems have virtually limitless data, so only issue is how big a model we can train
Research Objective: Minimizing Time to Results

• We want results of experiments quickly
• "Patience threshold": No one wants to wait more than a few days or a week for a result
• Significantly affects scale of problems that can be tackled
• We sometimes optimize for experiment turnaround time, rather than absolute minimal system resources for performing the experiment
Train in a day what takes a single GPU card 50 days
Many Different Types of Models Used

Rest of talk:

• Quick sketch of different models that we use and are exploring
• Overview of general infrastructure that makes training large models on large datasets possible
Acoustic Modeling for Speech Recognition

Simple feed-forward neural net

Close collaboration with Google Speech team
2012: Unsupervised Vision Model: QuocNet

Neurons use local receptive fields:
# parameters scales with computation (billions of parameters)
2012 Supervised Vision Model: “AlexNet”

- Softmax to predict object class
- Fully-connected layers (and trained w/ DropOut)
  - Basic architecture developed by Krizhevsky, Sutskever & Hinton
  - Won 2012 ImageNet challenge with 16.4% top-5 error rate

Convolutional layers (same weights used at all spatial locations in layer)

~60M parameters

Some very focused on narrow area (1x1), some focused on wider area (5x5)
Vision models, 2014 edition: GoogLeNet

Module with 6 separate convolutional layers

24 layers deep!

No fully-connected layer, so only ~6M parameters (but ~2B floating point operations per example)

Developed by team of Google Researchers:
Won 2014 ImageNet challenge with 6.66% top-5 error rate
Video models

What's the right architecture?
Sparse Embedding Models
Embeddings allow DNNs to deal with sparse data

~1000-D joint embedding space

Camera

porpoise

dolphin

SeaWorld

Paris
Embedding Models

Skipgram Text Model

Hierarchical softmax classifier

Raw sparse features

Single embedding function

Obama is meeting with Putin

nearby word
Applying Deep Nets to Prediction problems

Logistic regression

Zero or more hidden layers

Floating-point representations

Learned embedding functions

Raw feature categories

words in query

user country

... ...

Google
LSTM for End to End Translation

Source Language:  A B C
Target Language:  W X Y Z

Combining ConvNets and LSTMs
Pretty Wide Variety of Models
But One Underlying System..
Key abstractions of our Software Infrastructure

**Layer**, composed of **Nodes**

Layers implement **ComputeUp** and **ComputeDown** Layers: can be **partitioned** across multiple machines (system manages all the communication automatically)

**Model**: Simple textual description of layers and their connectivity

Parallelizing Deep Network Training
Model Parallelism
Transfer Needed Data Across Machine Boundaries
Model Connectivity and Communication

Dense within-device connections

Sparse and infrequent cross-device connectivity
Concurrent Steps

We often run multiple concurrent steps (perhaps 2 to 5) in the same model.

Effectively pipelines a larger batch, and can hide some communication latency with useful computation.
Model Connectivity and Communication

Device or machine 1
Device or machine 2
Device or machine 3
Data Parallelism

Model Workers

Data Subsets
Synchronize $O(\# \text{ weights})$ Parameters

Parameter Server

Model Workers

Data Subsets
Data Parallelism:
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server

\[ p'' = p' + \Delta p' \]

Model

Data
Data Parallelism:
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server

$p' = p + \Delta p$

Model Workers

$\Delta p$  $p'$

Data Shards
Model Parameters and Reuse

Good for data parallelism scalability if either:

1. Most parameters in model aren’t needed for a particular batch of examples (e.g. sparse embedding models), or
2. Each parameter fetched over the network is reused many times as part of a mini batch (e.g. convolutional vision models, unrolled LSTM models, etc.).
Feasible Number of Replicas

Depends on many things:
(1) Number of parameters needed for each minibatch
(2) Sparsity of updates

Very dense, fully-connected models: 10-50 replicas
Sparse models (mostly embeddings): 500-1000 replicas
Staleness

Parameter Server

Model Workers

Data Shards

Very slow

Too old! Ignore!
Adagrad

Technique for learning a per-parameter learning rate

• Scale update by:

\[ \frac{l}{\sqrt{\sum_{i=1}^{t} \Delta w_{ij}^2}} \]

• Especially useful if not every parameter updated on every

Low-Level Optimizations

Added support for AVX and FMA to Eigen.

Overview  Commits  Activity

Author  Benoit Steiner

Reviewers  No reviewers

Description  This branch adds support for AVX and FMA to Eigen. When combined with the patches attached to bugs 717, 721, and 724 this doubles the speed of the matrix multiplication code on SandyBridge and IvyBridge, and about triples the speed on Haswell.
Low-Level Optimizations

Avoid locking, or use fine-grained locking
Avoid excessive copies for data transferred over the network
Compile versions of important operations specialized to actual sizes
Reduced Precision

Neural nets are very tolerant of reduced precision:

- 8 bits or less for inference
- 12 to 14 bits for training

For sending parameter values over network, we often use lower precision than floats:
- 16-bit “floats”: just truncate the mantissa (don’t even bother with correct rounding)
Large-Scale Batch Methods

We have experimented with large-scale batch methods like L-BFGS

Not really successful for us compared with SGD

Wish they were: very parallelizable
Hyper-parameter Search

Good hyper-parameter selection is critical

Techniques:
(1) Grid search
(2) Grid search, but with heuristics to kill unpromising experiments early
(3) Fancier methods like Gaussian Processes
   e.g. Spearmint: “Practical Bayesian Optimization of Machine Learning Algorithms”, Snoek, Larochelle and Adams, NIPS 2012
Classifications for Many Classes

Many problems need to predict 1 of N classes, and N can be quite large
  • Image models are 10000s of classes
  • Text vocabularies start at 10000s of words and go up to millions
  • Action prediction can be hundreds of millions or billions of actions (e.g. all YouTube videos)
Classifications for Many Classes

For up to a few 10000s of classes, we use a full softmax. For large vocabularies, we use either:

1. Hierarchical softmax or
2. Noise-contrastive estimation
Ensembles

Good news:
(1) Ensembling many models generally improves accuracy significantly (but with diminishing returns after 5 or 10 models)
(2) Completely parallelizable

Bad news:
(1) Makes inference cost much higher
Distillation

Technique for taking large ensemble and using its predictions to train smaller, cheaper model that captures most of the ensemble’s benefits

Have to train a large model or ensemble, but can then distill it into a lighter-weight model so that inference is faster

*Distilling Knowledge in a Neural Network*, Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean, Deep Learning and Representation Learning Workshop, NIPS 2014
Specialists

For large datasets, can train many models in parallel, each specialized for a subset of the classes

Completely parallelizable during training

Only need to consult relevant specialists during inference

*Distilling Knowledge in a Neural Network*, Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean, Deep Learning and Representation Learning Workshop, NIPS 2014
Current Work

Actively building TensorFlow, our next-generation of training and inference software.

Why?
(1) Learned more about the problem in first two years
(2) Want more flexible system, not geared as much towards only SGD training of neural nets
(3) Target a wider range of computational devices
(4) Combine convenience and flexibility of research systems like Theano & Torch with production readiness and scalability of our first system
Thanks!

Questions?