Machine Learning With Distributed Training on Blue Waters

Aaron D. Saxton, PhD, Data Scientist
saxton@illinois.edu
Statistics Review

- Simple $y = mx + b$ regression
  - **Goal:** Find $m, b$
    - With data set $\{(x_i, y_i)\}_{i=1,\ldots,n}$
    - Let the error be $R = \sum_{i=1}^{n} [(y_i - (m \cdot x_i + b))^2]$
    - Minimize $R$ with respect to $m$ and $b$.
    - In practice we consider more general $y = f(x)$
Statistics Review

- Regressions with parameterized sets of functions. e.g.
  - \( y = ax^2 + bx + c \) (quadratic)
  - \( y = \sum a_i x^i \) (polynomial)
  - \( y = Ne^{rx} \) (exponential)
  - \( y = \frac{1}{1 + e^{-(a+bx)}} \) (logistic)
Statistics Review

• Rational models of degree ‘n’
  • “degrees of freedom”
    - models capacity

Gradient Decent

- Searching for minimum of

\[ R = \sum_{i=1}^{n} [(y_i - f_{\theta_i}(x_i))^2] \]

- \( \nabla R = \left\langle R_{\theta_0}, R_{\theta_2}, \ldots, R_{\theta_n} \right\rangle \),
  - \( R \) and \( \nabla R \) is a sum over \( i \)

- Update parameters
  - \( R(\vec{\theta}_{t+1}) = R(\vec{\theta}_t + \gamma \nabla R) \)
  - \( \gamma \): Learning Rate
Stochastic Gradient Decent

- Single training example, \((x_i, y_i)\), Sum over only one training example

\[
\nabla R(x_i, y_i) = \left\langle R_{\theta_0}, R_{\theta_2}, \ldots, R_{\theta_n} \right\rangle (x_i, y_i)
\]

- \(R_{(x_i, y_i)}(\hat{\theta}_{t+1}) = R_{(x_i, y_i)}(\hat{\theta}_t + \gamma \nabla R_{(x_i, y_i)})\)

- \(\gamma\): Learning Rate

- Choose next \((x_{i+1}, y_{i+1})\), (Shuffled training set)

- SGD with mini batches

- Many training example, \((x_i, y_i)\), Sum over many training example
  - Batch Size or Mini Batch Size (This gets ambiguous with distributed training)
  - SGD often outperforms traditional GD, want small batches.
    - https://arxiv.org/abs/1711.04325, Extremely Large ... in 15 Minutes
Common ML Models and Datasets

- MNIST
- ImageNet
- Fully connected Neural Network
- Deep Convolutional Neural Network

\[
Z_M = \sigma(\alpha_0 m + \alpha_m X) \\
T_K = \beta_0 k + \beta_k Z \\
f_K(X) = g_k(T)
\]
Modern ML Taxonomy

- Convolution
- Fully connected layers (traditional “Neural Networks”)
- Max Pooling
- Concatenation
- RNN
- GRU
- Transformers

LSTM Cell


https://setosa.io/ev/image-kernels/
Faux Model Example

Image Input

Max Pool, 5x5

Conv2d, 3x3

Conv2d, 3x3

Concat

Conv2d, 5x5

Fully Connected

n: [6,3,2]

SoftMax

Classification Output

Trainable Weights

\( \{ \theta_i : i \in [0, 1, 2, 3, 4] \} \)
Distributed Training, Data Distributed
Distributed Training, Data Distributed
Distributed Training, Data Distributed
Effects of Batch Size on Training, Bad “Generalization”

- Training on MNIST dataset
- Model is a NN with 2 FC layers
- Orange: Batchsize 64  
  Blue: Batchsize 256  
  Purple: Batchsize 1024
- There are ways to mitigate this and scale your model training. Ask me offline: saxton@illinos.edu

https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e
What can we do?

• Regularization!
  • Perturb data
  • Variable learning rate
  • Batch Normalization
• I1, I2 normalization
• Dropout
• Other novel techniques
• No generalized theory that will guarantee what works for you
• Reach out to Aaron Saxton saxton@illinois.edu

[Deep Learning] Batch Normalization
(https://medium.com/@tsengyangyu/batch-normalization-58aac99ee26)
Practical Implementations: Cray ML Plugin

- Cray Optimized MPI Tensor serialization
- Runs concurrently with standard Tensorflow

```python
import ml_comm as mc

tot_model_size = sum([reduce(lambda x, y: x*y, v.get_shape().as_list()) for v in tf.trainable_variables()])
mc.init(1, 1, tot_model_size, "tensorflow")

mc.config_team(0, 0, 100, FLAGS.num_steps, 2, 1)

class BcastTensors(tf.train.SessionRunHook):
    def __init__(self):
        self.bcast = None

    def begin(self):
        new_vars = mc.broadcast(tf.trainable_variables(), 0)
        self.bcast = tf.group([tf.assign(v, new_vars[k]) for k, v in enumerate(tf.trainable_variables())])
        grads_and_vars = optimizer.compute_gradients(total_loss)
        grads = mc.gradients([gv[0] for gv in grads_and_vars], 0)
        gs_and_vs = [(g, v) for (_, v), g in zip(grads_and_vars, grads)]

        train_op = optimizer.apply_gradients(gs_and_vs, global_step=global_step)

        hooks = [tf.train.StopAtStepHook(last_step=FLAGS.num_steps), BcastTensors()]
```

Build Data, Model, and Training Somewhere Here
Practical Implementations: A case study in unsupervised ML on BW

- Unsupervised timeseries classification
  - Novel combination of LSTM, Auto Encoder, K-Means
- Hosting 3TB of time series data in memory
  - Distributed, Indexed, and Queryable: MongoDB
- **Heterogenous Jobs** using XE nodes to host MongoDB cluster XK nodes to run Tensorflow
  - ~3000 XE nodes
  - 64 XK nodes
- Training unsupervised models on data samples that range from ~100MB to ~10GB
- Diminishing return on time to solution at 64 XK nodes
  - Personal observation that most vanilla models give fastest time to solution between with batch size between 16 and 64.
Questions?

Machine Learning With Distributed Training on Blue Waters
Aaron D. Saxton, PhD, Data Scientist
saxton@illinois.edu