Summary

- Training supervised learning models is computationally intensive and difficult to parallelize across multiple compute nodes.
- In particular, batch gradient descent requires memorizing many gradients and potentially broadcasting parameters over a network.
- In this project, we assess the feasibility of Evolution Strategies for performing supervised learning training. Evolution Strategies are a stochastic optimization technique most commonly used in reinforcement learning.
- We found that even for simpler nets, effective Hybrid-ES requires extensive hyperparameter tuning, but its potential memory + data savings mean we should keep investigating it.

The Hybrid-Evolution Algorithm

- Parallel BGD (N worker nodes)
  - Algorithm:
    1. Split training set T into N subsets, T_n
    2. For every iteration i, each worker node:
       3. forward_prop(T_i)
       4. backward_prop(T_i)
       5. \theta_i \leftarrow \theta_i - \Delta \theta_i
       6. transmit(\theta_i)
       7. receive(\bar{\theta}_{set, n}, \alpha_{set, n})
       8. \theta \leftarrow combine(\theta, \alpha)
- Parallel Hybrid-ES (N worker nodes)
  - Algorithm:
    1. If iteration / N \equiv 0:
    2. forward_prop(T, \theta)
    3. backward_prop(T, \theta)
    4. \theta \leftarrow \theta - \Delta \theta
    5. Else:
    6. For K attempts, each worker node:
       7. \Delta \theta \leftarrow d\theta + \mathcal{N}(0, \sigma^2)
       8. \theta_i \leftarrow \theta_i - \alpha \Delta \theta_i
       9. \theta_i \leftarrow \text{argmin forward_prop(T, \theta_i)}
    10. transmit(\text{seed, best_cost})
    11. receive(\text{seed, best_costs})
    12. \theta \leftarrow combine(\text{seed, best_costs})

Hyperparameters + Savings

- Hyperparameters:
  - K, the number of random perturbations each worker node makes (multiply by N)
  - \alpha, the interval for computing the full gradient (as opposed to a stochastic update)
  - \sigma^2, the variance for the random shift matrices

- Model uses the components defined in the algorithms section.
- Runtime doesn’t take into account the cost of sending over network! (So this is a conservative estimate.)
- Network BW – not just data: delay, energy, etc.
- Backprop expected to be costlier, but wasn’t (might be worse for larger nets). Memory + BW benefits increase with net size!

Investigating the Gradient

- We first characterized the behavior of the full gradient, as we want to mimic it stochastically.
- Periodicity of the figure is probably due to cycling over minibatches.
- The norm of the gradients quickly converge, likely due to L2 regularization.

Baseline Network

Most of the algorithm design exploration was done using a multilayer perceptron (MLP) on the MNIST dataset. This allowed for relatively quick iteration and figuring out what worked/what didn’t without having to train a huge net.

The MNIST MLP is has one input layer, one hidden layer (300 hidden units), and a 10-class softmax output layer. Learning rate was 10^-4. It was trained for 30 epochs, each over 50 minibatches of size 1000.

Results + Analysis

<table>
<thead>
<tr>
<th>Reference</th>
<th>Training Set Accuracy</th>
<th>Dev Set Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assisted H-ES (r = 1)</td>
<td>99.54%</td>
<td>96.57%</td>
</tr>
<tr>
<td>H-ES, r = 2</td>
<td>96.80%</td>
<td>94.21%</td>
</tr>
<tr>
<td>H-ES, r = 3</td>
<td>95.34%</td>
<td>93.06%</td>
</tr>
</tbody>
</table>

- Goal isn’t to beat BGD at its own game, but to parallelize BGD in an approximate but much lower-overhead way.
- Hybrid-ES can make forward progress without needing to compute the full gradient.
- May be better for driving training progress in later iterations (once gradient has stabilized).
- Optimal \sigma^2, empirical gradient component var.
- Optimal r with too big, H-ES loses information from the full gradient and can’t make progress.

Future Work

- Try adaptively setting the \sigma^2 variance (shift scaling factor).
- Try stochastically adjusting different components of the gradient.
- Try learning some features of the gradient itself.
- Try sampling random shifts from a non-normal distribution.
- End goal: compress the weights being sent over the network.
- Simulate across a real cluster, using heterogeneous (CPU, GPU, TPU) HW.

References