Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches

Yeming Wen†‡, Paul Vicol†‡, Jimmy Ba†‡, Dustin Tran‡*, Roger Grosse†‡
†University of Toronto ‡Vector Institute *Google ‡Columbia University

Motivation

- Stochastic weights are used in many settings:
  - Regularization (DropConnect)
  - Training BNNs (Gaussian perturbations)
  - Evolution Strategies
  - Exploration in reinforcement learning
- Due to the large number of weights, it is very expensive to compute and store separate weight perturbations for every example in a mini-batch.
- All examples in a mini-batch typically share the same weight perturbation, thereby limiting the variance reduction effect of large mini-batches.

Summary

- We developed a method called Flipout that allows us to sample pseudo-independent weight perturbations efficiently for each example in a mini-batch.
- Flipout decorrelates the gradients between examples and achieves a $1/N$ variance reduction effect in practice.
- Flipout applies to any perturbation distribution that factorizes by weight and is symmetric around 0.
- Flipout speeds up training neural networks with multiplicative Gaussian perturbations, is effective at regularizing LSTMs, and enables us to vectorize evolution strategies.

Theoretical Results

- Flipout gives unbiased stochastic gradients.
- Flipout is guaranteed to have smaller variance than shared perturbations.

\[
\alpha = \frac{1}{N}, \quad \beta = \frac{1}{N} + \gamma 
\]

\[
\gamma = \text{covariance from sampling } \Delta W 
\]

\[
\beta = \text{covariance from sampling } r, s 
\]

\[
\alpha = \frac{1}{N} \text{ variance of gradients on individual examples} 
\]

Method

- One shared perturbation matrix... one sign matrices...
  - ...multiplied by independent rank
  - ...yields pseudo-independent weight perturbations.

\[
\Delta W = \Delta W \circ r_1 s_1^T 
\]

\[
\Delta W = \Delta W \circ r_2 s_2^T 
\]

- To vectorize these computations, we define matrices $R$ and $S$ whose rows correspond to the random sign vectors $r_i$ and $s_i$ for all examples in the mini-batch. Let $X$ denote the batch activations in one layer of a neural net. The next layer’s activations are given by:

\[
Y = \phi \left( X W + \left( (X \circ S) \Delta W \right) \circ R \right) 
\]

where $\phi$ denotes the activation function.

Variance Reduction

- Flipout achieves the ideal linear variance reduction with increasing mini-batch size for FC-NNs, CNNs, and RNNs.

\[
\text{Train/uniError} \quad \text{Validation/uniError} 
\]

\[
\text{Train/Loss} \quad \text{Validation/Loss} 
\]

LSTM Regularization

- Character-level Penn Treebank: Flipout achieves the best reported results for a 1-layer, 1000 hidden unit architecture.

\[
\text{Model} \quad \text{Valid} \quad \text{Test} 
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Valid</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unregularized LSTM</td>
<td>1.468</td>
<td>1.423</td>
</tr>
<tr>
<td>Semeniuta (2016)</td>
<td>1.337</td>
<td>1.300</td>
</tr>
<tr>
<td>Zoneout (2016)</td>
<td>1.306</td>
<td>1.270</td>
</tr>
<tr>
<td>Gal (2016)</td>
<td>1.277</td>
<td>1.245</td>
</tr>
<tr>
<td>Mult. Gauss. (ours)</td>
<td>1.257</td>
<td>1.230</td>
</tr>
<tr>
<td>Mult. Gauss + Flipout (ours)</td>
<td>1.256</td>
<td>1.227</td>
</tr>
</tbody>
</table>

Large Batch Training

- Flipout converges in $\sim3x$ fewer iterations than shared perturbations and is $\sim2x$ as expensive, yielding a 1.5x speedup overall.

\[
\text{Train Loss (FC)} \quad \text{Train Loss (Conv)} 
\]

\[
\text{Iterations} 
\]

Figure: MNIST training using Bayes By Backprop with batch size 8192

Vectorizing Evolution Strategies

- FlipES is as sample-efficient as using fully-independent perturbations. One GPU with Flipout can handle the same throughput as at least 40 CPU cores using existing methods.