Overview

Batch normalization

\[
\begin{align*}
\text{BN}(z) &= \frac{z - E[z]}{\sqrt{\text{Var}[z]}} \\
&= u' = \frac{x - E[x]}{\sqrt{\text{Var}[x]}},
\end{align*}
\]

where \(u = w/|w|\) and \(R_{xx}u\) is the covariance matrix of \(x\).

- forward pass is scale invariant:
  \[
  \text{BN}(w^T x) = \text{BN}(u^T x)
  \]
- backward pass is scale invariant:
  \[
  \frac{\partial \text{BN}(w^T x)}{\partial x} = \frac{\partial \text{BN}(u^T x)}{\partial x}
  \]
- weight update is not scale invariant
  \[
  \frac{\partial \text{BN}(z)}{\partial w} = \frac{1}{|w|} \frac{\partial \text{BN}(z)}{\partial u}
  \]

Things we observe …

- Two networks, with the same forward pass but different weight scaling, may fall into different local minima
- \(L_2\) regularization of the weights is known to be indispensable for the performance, but \(|w|\) does not affect the output of the network. Why is it important then?
- The scale of \(|w|\) affects the learning rate, so \(L_2\) regularization functions as a learning rate control unexpectedly.

What we did …

- Give scale invariance to the weight update step

How?

- Utilize the geometry of the space of scale invariant vectors
- Derive learning rules for this space
- Derive a new regularization method in this space

Operators on a Grassmann manifold \(G(1,n)\)

Gradient of a function

- \(\nabla f(y) = T_p \mathbb{M} (\nabla f(y))\)
- \(\nabla f(x)\) is a tangent vector to the manifold
- \(\nabla f = \nabla g \circ g(y)\) where \(g = \delta f/\delta y\)

Exponential map

\[
\exp_p(h) = \cos ||h|| + \frac{||h||}{||h||} \sin ||h||
\]

Parallel translation

\[
p_t(x; h, \Delta h, k) = \cos ||h|| - \frac{||h||}{||h||} \sin ||h||
\]

\(R^2\) vs \(G(1,2)\)

- \(R^2\) : Only the tangent direction \(\hat{h}\) contributes to minimizing cost, and the normal direction disturbs the learning rate
- \(G(1,2)\) : Move the point by \(|h|\) (rad)

- \(y = \exp_y(x; \hat{h}, ||h||)\)
- It returns to the original point after moving by \(2\pi\)

Proposed algorithms

We derive iterative algorithms to solve the unconstrained optimization on \(G(1,n)\):

\[
\min_{y \in G(1,n)} \ell(y, f(y))
\]

- Optimizations algorithms in Euclidean space can be easily extended to those on manifolds by properly using the operators defined on the manifolds
- A weight vector is a point on the manifold, so we can move it by the exponential map
- Gradient is a tangent vector to the manifold, so the momentum (accumulation of gradient) can be moved by the parallel translation

Experiments

- Classification error rate on CIFAR (median of five runs): WRN-28-10 denotes a wide residual network that has \(d\) convolutional layers and a widening factor \(k\)
- Classification error rate on SVHN (median of five runs):

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>SGGD-G</td>
<td>SGGD-G</td>
</tr>
<tr>
<td>VGG-13</td>
<td>5.88</td>
<td>6.05</td>
</tr>
<tr>
<td>VGG-19</td>
<td>6.49</td>
<td>6.02</td>
</tr>
<tr>
<td>WRN-28-10</td>
<td>3.89</td>
<td>3.78</td>
</tr>
<tr>
<td>WRN-40-10</td>
<td>3.72</td>
<td>3.86</td>
</tr>
</tbody>
</table>

- The proposed algorithms suffer less from a plateau after each learning rate drop
- The proposed algorithms achieve lower training loss than baseline SGD

Space of scale invariant vectors

We want a space where all the scaled versions of a vector collapse to a point

1) 1-d Grassmann manifold \(G(1,n)\):
   - \(y\) and \(\hat{y}\) are equivalent if and only if \(y = \hat{y} + a\) for \(a \in \mathbb{R}\)\[0\]
   - \(g_0(\hat{a}, \hat{a}) = 2\hat{a}/\hat{y}^2\hat{y}\)
   - \(g_0(\hat{a}, \hat{b}) = \hat{a}\hat{b}\)

2) 1-d Stiefel manifold \(V(1,n)\):
   - \(y\) and \(\hat{y}\) are equivalent if \(y = \hat{y}\) for \(a \in \mathbb{R}\)\[0\]
   - \(g_0(\hat{a}, \hat{a}) = 2\hat{a}/\hat{y}^2\hat{y}\)
   - \(g_0(\hat{a}, \hat{b}) = \hat{a}\hat{b}\)

If we choose a representation \(y\) with \(y^2 = 1\) for \(G(1,n)\) and \(V(1,n)\) offer the same metric and resulting operators

Source code for the experiments is available at https://github.com/MiHyungCho/riemannian-batch-normalization