Weight Normalization for Neural Networks

Presented by Weining Qu

For MATHGR5430 Machine Learning for Finance

Columbia University

Spring 2023

Motivation

- Despite complex deep neural networks, efficient first-order gradient-based optimizations and speedy convergence still matter to major architectures.
- Batch normalization
  - A common regularization technique dealing with various input ranges of original data and make model parameter updates consistent by normalizing each input dimension to a common range.
  - Also can accelerate training of deep neural networks, allowing higher learning rate and less dependence on initialization.
  - *Drawbacks: additional memories and computations, dependency on batch data, adding noise to the gradients.*

Recap

- Batch normalization
  - For each mini-batch train data,

$$X_b = (x_1, \ldots, x_N)^T \in R^{N \times D}, \quad x_i \in R^D$$

  - Standardize batch data by sample batch mean and variance

$$\hat{\mu} = \bar{X}_b = \frac{1}{N} \sum_{i=1}^{N} x_i \in R^D, \quad \hat{\sigma}_2 = S^2(X_b) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{X}_b)^2 \in R^D$$

$$\hat{X}_b = \frac{X_b - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \epsilon}} \in R^{N \times D}$$

  - Transform standardized X -> Y = BN(X, beta, gamma) linearly, beta, gamma are learnable parameters to recover “true” distribution

$$Y_b = \hat{X}_b \odot \gamma + \beta$$

  - Record moving mean/variance, an exponentially decaying sum of the mean/variance from all previous batches and the current batch sample mean/variance.

$$\begin{cases} \mu_b = \alpha \mu_{b-1} + (1 - \alpha) \bar{X}_b \\ \sigma_b^2 = \alpha \sigma_{b-1} + (1 - \alpha) S^2(X_b) \end{cases}$$

  - At test stage, standardization parameters becomes the final moving mean/variance.

$$\hat{\mu} = \mu_B, \quad \hat{\sigma}^2 = \sigma_B^2$$

Weight Normalization

- A reparameterization method inspired by batch normalization.
- More general, less dependency on data, therefore applicable to
  - recurrent models such as LSTMs,
  - and noise-sensitive models such as reinforcement learning or generative models.
- Empirically verified for faster gradient descent convergence.

Weight Normalization

- Operation of each neuron: a weighted sum of input features with bias, followed by a non-linear activation ($\phi(\cdot)$),

$$y = \phi(w \cdot X + b)$$

- Parameters: w, b
- Decompose weight vector w into
  - a scalar parameter g representing its length and
  - a k-dimensional vector parameter $v/||v||$ representing its direction,

$$w = \frac{g}{||v||} v,$$

- New parameters: $g = ||w||, v$.

Weight Normalization

- Gradient:

$$\nabla_g L = \frac{\nabla_w L \cdot v}{||v||}, \nabla_{\mathbf{v}} L = \frac{g}{||\mathbf{v}||} M_{\mathbf{w}} \nabla_{\mathbf{w}} L, \quad \text{with} \quad \mathrm{M}_{\mathbf{w}} = \mathrm{I} - \frac{\mathbf{w}\mathbf{w}'}{||\mathbf{w}||^2},$$

- where M_w is a projection matrix that projects onto the complement of the w vector.
- Two effects of reparameterization:
  - the gradient of w is scaled by $g/||v||$ and
  - the gradient is project away from current w. This can eliminate noise in the original direction, bring robustness to various learning rates and speed up optimization.
- Only need moderate modifications to back-propagation procedure that is independent of input features batch size.

Stableness of optimization with weight normalization

$$\nabla_{\mathbf{v}} L = \frac{g}{||\mathbf{v}||} M_{\mathbf{w}} \nabla_{\mathbf{w}} L, \quad \text{with} \quad \mathrm{M}_{\mathbf{w}} = \mathrm{I} - \frac{\mathbf{w}\mathbf{w}'}{||\mathbf{w}||^2},$$

- Let $v' = v + \Delta v$ denote parameter update, with $\Delta v \propto \nabla_v L$ (steepest ascent/descent), then Δv is orthogonal to w as we project it away.
- v is proportional to w, then update of v is orthogonal to w and the norm of v grows monotonically with the number of updates.
- If $\frac{||\Delta v||}{||v||} = c$, the new weight vector will have norm

$$||\mathbf{v}'|| = \sqrt{||\mathbf{v}||^2 + c^2||\mathbf{v}||^2} = \sqrt{1 + c^2}||\mathbf{v}|| \geq ||\mathbf{v}||$$

- The rate of increase will depend on the variance of the weight gradient.
- If gradients are noisy, c is large, the norm of v will quickly increase, which in turn decreases the scaling factor $g/||v||$.
- If the norm of the gradients is small, we get $\sqrt{1 + c^2} \approx 1$ and the norm of v will stop increasing.
- Using this mechanism, the scaled gradient self-stabilizes its norm.

Data-dependent parameter initialization

- Lack of property of robust parameter initialization from batch normalization
- Initialize in a way that ensure all features have zero mean and unit variance pre-activation [3]
  - V: normal distribution (mean 0 and standard deviation 0.5)
  - g and b: an initial feedforward pass through of a mini-batch

$$t = \frac{v \cdot X}{||v||} \quad \text{and} \quad y = \phi\left(\frac{t - \mu[t]}{\sigma[t]}\right),$$

$$g \leftarrow \frac{1}{\sigma[t]}, \text{ and } b \leftarrow \frac{-\mu[t]}{\sigma[t]}.$$

CNN with weight normalization

Base model: modified ConvPool-CNN-C [2]

```
'''
This function returns a multi-layer keras model with weight normalization wrapper
takes 2 inputs:
input_shape: a list of integers representing the shape of the inputs
nb_classes: an integer representing the number of categories
'''
def WN_model(input_shape, nb_classes):
    inputs = tf.keras.Input(input_shape)
    x = GaussianNoise(stddev = 0.15)(inputs)
    x = WeightNorm(Convolution2D(96, (3, 3), padding='same',activation=lrelu), data_dep_init = True)(x)
    x = WeightNorm(Convolution2D(96, (3, 3),padding='same',activation=lrelu), data_dep_init = True)(x)
    x = WeightNorm(Convolution2D(96, (3, 3),padding='same',activation=lrelu), data_dep_init = True)(x)
    x = MaxPooling2D((2, 2))(x)
    x = Dropout(0.5)(x)

    x = WeightNorm(Convolution2D(192, (3, 3),padding='same',activation=lrelu), data_dep_init = True)(x)
    x = WeightNorm(Convolution2D(192, (3, 3),padding='same',activation=lrelu), data_dep_init = True)(x)
    x = WeightNorm(Convolution2D(192, (3, 3),padding='same',activation=lrelu), data_dep_init = True)(x)
    x = MaxPooling2D((2, 2))(x)
    x = Dropout(0.5)(x)

    x = Flatten()(x)
    x = WeightNorm(Dense(512, activation='relu'), data_dep_init = True)(x)
    x = Dropout(0.5)(x)
    outputs = Dense(nb_classes, activation='softmax')(x)

    model = tf.keras.Model(inputs, outputs, name = 'WN_Model')
    return model
```

Key functions of weight normalization

- No transformation of input features
- A change to the kernel variable (no explicit gradient computation needed)

```
def _get_weight_norm(self):
    """Generate weights with normalization."""
    # Determine the axis to expand for parameter scalar 'g'
    new_axis = -self.filter_axes - 3
    # recover kernel weight vector 'w' from its length 'g' and new direction 'v'
    self.layer.kernel = tf.nn.l2_normalize(
        self.v, axis=self.kernel_norm_axes) * tf.expand_dims(self.g, new_axis)
```

```
self.v = self.layer.kernel
self.layer.kernel = None # to avoid a duplicate `kernel` variable after `build` is called

# initialize length scalar 'g'
self.g = self.add_weight(
    name='g',
    shape=(int(self.v.shape[self.filter_axes]),),
    initializer='ones',
    dtype=self.v.dtype,
    trainable=True)
```

```
def call(self, inputs):
    """Call `Layer`."""
    if not self.initialized:
        if self.data_dep_init:
            # use data-dependent initialization
            self._data_dep_init(inputs)
        else:
            # initialize `g` as the norm of the initialized kernel
            self._data_norm_init()
    self.initialized.assign(True)

    self._get_weight_norm() # reparameterization
    output = self.layer(inputs) # compute new output
    return output
```

Results of original paper

1. Supervised Classification: CIFAR-10
   - Model: the same as before (modified ConvPool-CNN-C)
   - Mean-only Batch Normalization: a computationally cheaper modification of Batch Normalization by subtracting out the mini-batch means.
   - Mean-only B.N. has the effect of centering the gradients that are backpropagated.

   A combination of Weight Normalization and mean-only B.N. achieves best test accuracy.

Model	Test Error
Maxout [6]	11.68%
Network in Network [17]	10.41%
Deeply Supervised [16]	9.6%
ConvPool-CNN-C [26]	9.31%
ALL-CNN-C [26]	9.08%
our CNN, mean-only B.N.	8.52%
our CNN, weight norm.	8.46%
our CNN, normal param.	8.43%
our CNN, batch norm.	8.05%
ours, W.N. + mean-only B.N.	**7.31%**

Figure 2: Classification results on CIFAR-10 without data augmentation.

Results of original paper

2. Generative Modelling:

Convolutional VAE (variational autoencoders) [4] on MINIST and CIFAR-10 data

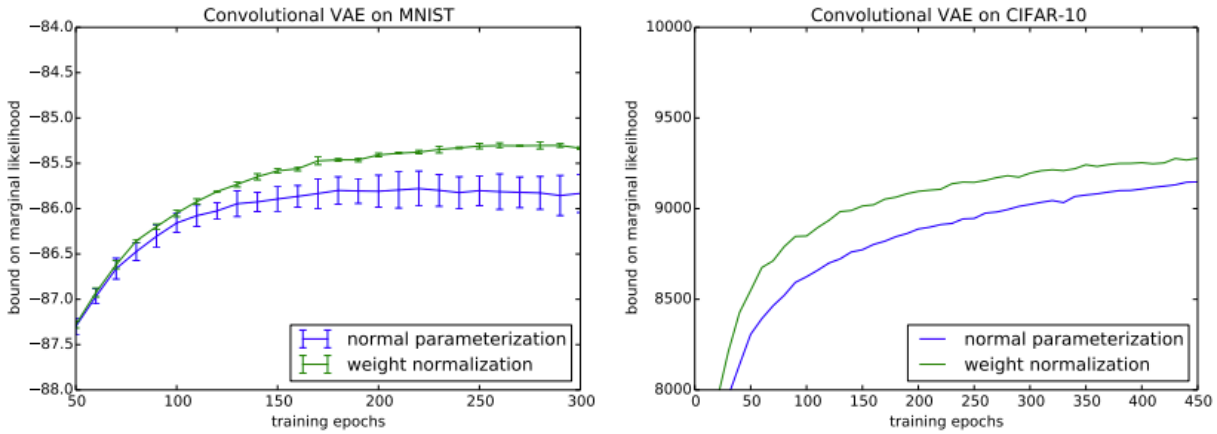

Figure 3: Marginal log likelihood lower bound on the MNIST (top) and CIFAR-10 (bottom) test sets for a convolutional VAE during training, for both the *standard* implementation as well as our modification with *weight normalization*. For MNIST, we provide standard error bars to indicate variance based on different initial random seeds.

Results of original paper

3. Reinforcement Learning: Deep Q-Network (DQN) [5]

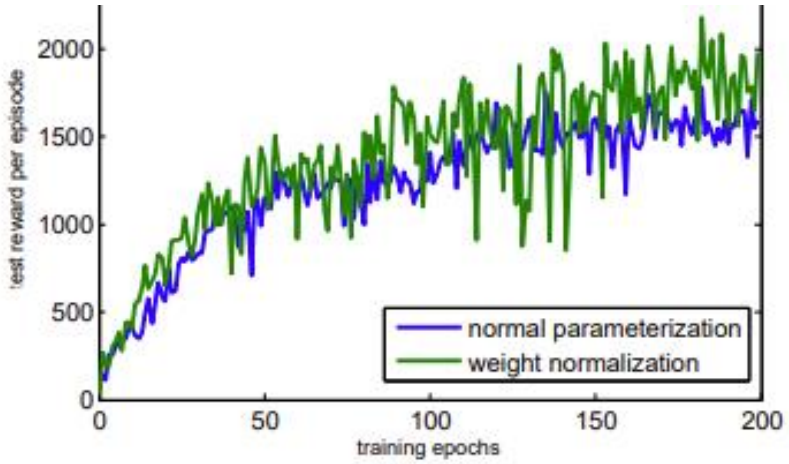

Figure 5: Evaluation scores for Space Invaders obtained by DQN after each epoch of training, for both the standard parameterization and using weight normalization. Learning rates for both cases were selected to maximize the highest achieved test score.

Reference

- [1] T. Salimans and D. Kingma, “Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks,” June. 2016. [Online]. Available: https://arxiv.org/pdf/1602.07868.pdf
- [2] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In ICLR Workshop Track, 2015.
- [3] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell. Data-dependent initializations of convolutional neural networks. arXiv preprint arXiv:1511.06856, 2015.
- [4] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. Proceedings of the 2nd International Conference on Learning Representations, 2013.
- [5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

Questions?