What is Group Normalization?

An alternative to Batch Normalization

Batch Normalization (BN) has been an important component of many state-of-the-art deep learning models, especially in computer vision. It normalizes the layer inputs by the mean and variance computed within a batch, hence the name. For BN to work the batch size is required to be sufficiently large, usually at least 32. However, there are situations that we have to settle for a small batch size:

- when each data sample is highly memory-consuming, e.g. video or high resolution image
- when we train a very large neural network, which leaves little GPU memory for processing data

Therefore we need alternatives to BN which work well with small batch size. Group Normalization (GN) is one of the latest normalization methods that avoids exploiting the batch dimension, thus is independent of batch size.

Different Normalization Methods

To motivate the formulation of GN we will first look at some of the previous normalization methods.

All of the following normalization methods perform the calculation

$$x_i \leftarrow (x_i - \mu_i) \;/\; \sqrt{(\sigma_i^2 + \varepsilon)}$$

for every coefficient x_i of an input feature x. μ_i and σ_i^2 are the mean and variance computed over a set S_i of coefficients, and ε is a small constant added for numerical stability and to avoid division by zero. The only difference is how the set S_i is chosen.

To illustrate the computation of the normalization methods, we consider a batch of size $N = 3$, with input features a, b, and c. They have channels $C = 4$, height $H = 1$, width $W = 2$:

```
a = [ [[2, 3]], [[5, 7]], [[11, 13]], [[17, 19]] ]
b = [ [[0, 1]], [[1, 2]], [[3, 5]], [[8, 13]] ]
c = [ [[1, 2]], [[3, 4]], [[5, 6]], [[7, 8]] ]
```

Hence the batch will have shape $(N, C, H, W) = (3, 4, 1, 2)$. We take $\varepsilon = 0.00001$.

Batch Normalization

BN normalizes the channels and computes μ_i and σ_i along the (N, H, W) axes. S_i is defined as the set of coefficients in the batch that are in the same channel as x_i.

For the first coefficient $a_i = 2$ of a, where $i = (0, 0, 0)$, the corresponding μ_i and σ_i^2 are computed over the coefficients of a, b, and c that are in the first channel:

```
μᵢ = mean(2, 3, 0, 1, 1, 2) = 1.5
σᵢ² = var(2, 3, 0, 1, 1, 2) = 0.917
```

Plugging these into the normalization formula,

```
aᵢ ← (2 - 1.5) / √(0.917 + 0.00001) = 0.522
```

Computing all the coefficients of a gives

```
a ← [ [[0.522, 1.567]], [[0.676, 1.690]], [[1.071, 1.630]],
[[1.066, 1.492]] ]
```

Layer Normalization

Layer Normalization (LN) is designed to overcome the drawbacks of BN, including its constraints on batch size. It computes μ_i and σ_i along the (C, H, W) axes, with S_i defined as all the coefficients that belong to the same input feature as x_i. As a result, the computation for an input feature is entirely independent of other input features in a batch.

All the coefficients of a are normalized by the same μ_i and σ_i^2

```
μᵢ = mean(2, 3, 5, 7, 11, 13, 17, 19) = 9.625
σᵢ² = var(2, 3, 5, 7, 11, 13, 17, 19) = 35.734
```

Therefore applying LN to a gives

```
a ← [ [[-1.276, -1.108]], [[-0.773, -0.439]], [[0.230, 0.565]],
[[1.234, 1.568]] ]
```

Instance Normalization

Instance Normalization (IN) can be viewed as applying the formula of BN to each input feature (a.k.a. instance) individually as if it is the only member in a batch. More precisely, IN computes μ_i and σ_i along the (H, W) axes, and S_i is defined as the set of coefficients that are in the same input feature and also in the same channel as x_i.

Since the computation of IN is the same as that of BN with batch size = 1, IN actually makes the situation even worse in most cases. However, for style transfer tasks, IN is better at discarding contrast information of an image, and has superior performances than BN.

For the first coefficient $a_i = 2$ of a, where $i = (0, 0, 0)$, the corresponding μ_i and σ_i^2 are simply

```
μᵢ = mean(2, 3) = 2.5
σᵢ² = var(2, 3) = 0.25
```

which gives

```
aᵢ ← (2 - 2.5) / √(0.25 + 0.00001) = -1.000
```

When we apply IN to a, we get

```
a ← [ [[-1.000, 1.000]], [[-1.000, 1.000]], [[-1.000, 1.000]],
[[-1.000, 1.000]] ]
```

Group Normalization

Previously we introduced IN as applying BN to each input feature individually as if batch size = 1. Notice that IN can also be viewed as applying LN to each channel individually as if the number of channels = 1.

Group Normalization (GN) is a middle ground between IN and LN. It organizes the channels into different groups, and computes μ_i and σ_i along the (H, W) axes and along a group of channels. S_i is then the set of coefficients that are in the same input feature and also in the same group of channels as x_i.

The number of groups G is a pre-defined hyperparameter, which is usually required to divide C. For simplicity we group the channels in a sequential order. So channels 1, ..., C / G belong to the 1st group, channels C / G + 1, ..., 2C / G belong to the 2nd group, and so on. When $G = C$, which means each group has only 1 channel, GN becomes IN. On the other hand, when G = 1, GN becomes LN. Therefore G controls the interpolation between IN and LN.

For our example consider $G = 2$. To normalize the first coefficient $a_i = 2$ of a where $i = (0, 0, 0)$, we use the coefficients of a in the first $4 / 2 = 2$ channels

```
μᵢ = mean(2, 3, 5, 7) = 4.25
σᵢ² = var(2, 3, 5, 7) = 3.687
```

Plugging these into the normalization formula,

```
aᵢ ← (2 - 4.25) / √(3.687 + 0.00001) = -1.172
```

For other coefficients of a, the computations are similar:

```
a ← [ [[-1.172, -0.651]], [[0.391, 1.432]], [[-1.265, -0.633]],
[[0.633, 1.265]] ]
```

. . .

Comparison of Normalization Methods

The following figure visualizes the relations among BN, LN, IN, and GN.

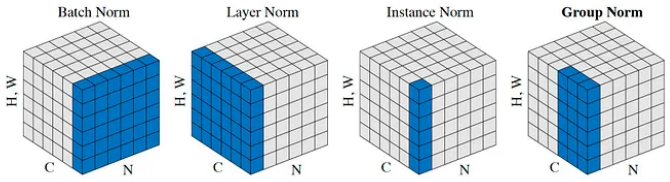

Figure 2 of [4]

The blue regions correspond to the sets S_i for the computation μ_i and σ_i, which are then used to normalize any coefficients in the blue regions.

From this figure we can see how GN interpolates between IN and LN. GN is better than IN as GN can exploit the dependence across the channels. It is also better than LN because it allows different distribution to be learned for each group of channels.

When the batch size is small, GN consistently outperforms BN. However, when the batch size is significantly large, GN does not scale as well as BN and might not be able to match the performance of BN.