Deep Learning
Normalization

Jian Li
IIIS, Tsinghua
Batch Normalization [Ioffe, Szegedy]

For a layer of input vector \( x \):

**Normalize:**

\[
\hat{x}(k) = \frac{x(k) - \mathbb{E}[x(k)]}{\sqrt{\text{Var}[x(k)]}}
\]

\[
y(k) = \gamma(k) \hat{x}(k) + \beta(k)
\]

- BP needs to be modified to account for the change 反向传播过程需要修正
- Improves gradient flow through the network 可以改善网络的梯度流动
- Allows higher learning rates 允许更高的学习速率
- Reduces the strong dependence on initialization 减少对初始化的依赖
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe 在某种程度上是正则化

**Additional parameters to learn (thru BP)**

Tensorflow Code:

```
tf.nn.batch_normalization(x, mean, variance, offset, scale, variance_epsilon, name=None)
```

Keras Code:

```
keras.layers.normalization.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True)
```
**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1 \ldots m\}$;  
Parameters to be learned: $\gamma, \beta$  
**Output:** $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

\[
\begin{align*}
\mu_\mathcal{B} & \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i & \quad \text{// mini-batch mean} \\
\sigma_\mathcal{B}^2 & \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 & \quad \text{// mini-batch variance} \\
\hat{x}_i & \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} & \quad \text{// normalize} \\
y_i & \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) & \quad \text{// scale and shift}
\end{align*}
\]

**Algorithm 1:** Batch Normalizing Transform, applied to activation $x$ over a mini-batch.

Tensorflow Code:
tf.nn.batch_normalization(x, mean, variance, offset, scale, variance_epsilon, name=None)  
Keras Code:
keras.layers.normalization.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True)
Why BN??

• Internal Covariate Shift

• In statistical learning, one typically assumes that the source distribution (training) and the target distribution (testing) are the same (if not, consider transfer learning)

Covariate shift: 

\[ x \in \mathcal{X}, \quad P_s(Y|X = x) = P_t(Y|X = x) \]

\[ P_s(X) \neq P_t(X) \]

In neural networks, after a few layers’ transformations, the distribution may change a lot.
• Batch Normalization relates to mini-batch size
• Typically, BN requires larger mini-batch size
• Not easy to applied to RNN (possible, but no conclusion)

Previous attempt to do BN on RNN

\[ h_t = \phi(BN(W_h h_{t-1} + W_x x_t)) \]

BN for both horizontal and vertical directions.
Doesn’t work
Horizontal: may need BN with different parameters for different time steps
Layer Normalization
Normalization over a layer (for a single sample)

Normalization over a mini-batch
Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with $N$ as the batch axis, $C$ as the channel axis, and $(H, W)$ as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.
Some details

• Normalization

\[ \bar{a}_i^l = \frac{g_i}{\sigma_i} (a_i^l - \mu_i^l) \]

\( g_i \): a gain parameter scaling the normalized activation before the non-linear activation function

• For a feedforward network, sum over a layer

• In CNN, summation over (C,H,W)

\[ \mu^l = \frac{1}{H} \sum_{i=1}^{H} a_i^l \quad \sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i^l - \mu^l)^2} \]

• In RNN, sum over a layer

\[ a^t = W_{hh} h^{t-1} + W_{xh} x^t. \]

\[ h^t = f \left[ \frac{g}{\sigma} \odot (a^t - \mu^t) + b \right] \quad \mu^t = \frac{1}{H} \sum_{i=1}^{H} a_i^t \quad \sigma^t = \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i^t - \mu^t)^2} \]
Group Normalization
Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with $N$ as the batch axis, $C$ as the channel axis, and $(H, W)$ as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.
Batch Norm:
+ Stable if the batch size is large
+ Robust (in train) to the scale & shift of input data
+ Robust to the scale of weight vector
+ Scale of update decreases while training
- Not good for online learning
- Not good for RNN, LSTM
- Different calculation between train and test

Layer Norm:
+ Effective to small mini batch RNN
+ Robust to the scale of input
+ Robust to the scale and shift of weight matrix
+ Scale of update decreases while training
- Might be not good at CNN?
(Batch Norm is better in some cases)

• Other normalizations: weight normalization
Thanks