Normalizations in Neural Networks

Batch, Instance, and Layer Nornalizations

Tony Chisenga, Reham Jamal, Jérémie Mabiala

Supervisors: Ms Damaris Ndejabyi and Ms Soona Osman

African Masters in Machine Intelligence
AIMS Sénégal, M'bour

March 29,2024

Outline

Introduction

- Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes, as the weight parameters are updated. For the model to learn and discriminate the features, good precautions can be taken up on the way we initialize the weight parameters.
- For instance, the network suffers from redundant features when the initial weight inputs are the same and constants. This is the symmetry problem. Several others issues that occur depend essentially on the initialization of the weights.
- These issues can decelerate the learning process and must be avoided. The capacity (or the performance) of the model is then essentially related to a good choice of the initial weights. **Normalization is one of the techniques that proved to be effective in solving such problems and has the ability to stabilize and accelerate the learning process.**

Normalization in Neural Network

- **Normalization in machine learning:** In the realm of machine Learning , Normalization generally refers to a set of feature scaling techniques that consists of transforming the data.
- In this context, we can cite two normalization technique, notably:
  1. **Range normalization:** consists of transforming the data in such way that it is bounded in some finite interval [?ref2].
  2. **Z-score normalization:** transforms the data in such a way that it has zero-mean and unit variance.

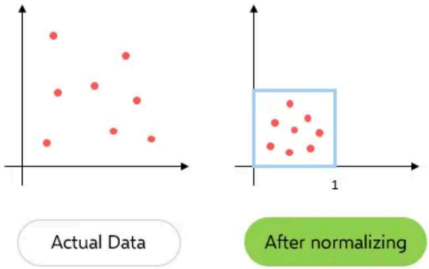

Figure 1: Geometric illustration of normalization (From the authors)

An example of **range normalization**, the actual data is then bounded in the unit square after normalization.

Why normalization for Neural Networks?

Several reasons motivate the use of normalization for neural networks, notably:

- **Reduction of the effects of the difference in scale of inputs**. This makes the features to have the same units and the same importance during the learning process.
  Consequence: Increase the performance of the model.
- **Improve the convergence**:The process keeps the weights in the same units and/or in some reasonable interval.
  Consequence: Avoid issues like vanishing and exploding gradient. Speed up the learning process.

We discuss three types of normalization:

- **Batch normalization** [1]
- **Instance normalization** [3]
- **Layer normalization** [2]

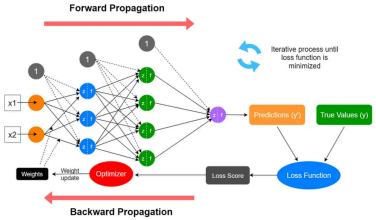

Figure 2: Neural Network with 2 hidden layer and two inputs.

Batch Normalization

- Batch normalization is one of the most commonly used normalization techniques in neural networks.
- It refers to the scenario where the distribution of internal activations or features within a neural network changes as the network's parameters are updated during the training. **Internal covariate shift** is a common challenge in training deep neural networks, especially those with many layers, and can lead to issues such as vanishing or exploding gradient.
- **Internal Covariate Shift**: refers to the change of the distribution of the distribution of the activations during the learning process [1].

Batch normalization aims to reduce/eliminate the Internal Covariate Shift [1].

- **How does it work?**. It works by **normalizing the activations of each layer within a mini-batch, thus reducing internal covariate shift and stabilizing the training process**.This normalization step is applied before the activation function of each layer.
- The scalar features are normalized independently to have the mean zero and variance 1. For a $d-$ dimensional layer $x = (x^{(1)}, \ldots, x^{(d)})$, we use:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}x^{(k)}}{\sqrt{Var\ x^{(k)}}},\ \forall k. \tag{1}$$

The expectation and the variance are computed over the training data.

- Normalizing each input of a layer can alter its representational capacity. To address this, scaling and shifting parameters $\gamma^{(k)}$ and $\beta^{(k)}$ are introduced for each activation $x^{(k)}$. These parameters allow the network to learn the optimal normalization while restoring the original activations if needed. The simple way to get these parameters is to set:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \tag{2}$$

$$\gamma^{(k)} = \sqrt{Var\ x^{(k)}} \tag{3}$$

$$\beta^{(k)} = \mathbb{E}x^{(k)} \tag{4}$$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1, \ldots, x_m\}$.
Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$$
\begin{array}{lll}
\mu_{\mathcal{B}} & \longleftarrow & \frac{1}{m}\sum_{i=1}^{m} x_i \qquad\qquad //\text{min-batch mean} \\
\sigma^2_{\mathcal{B}} & \longleftarrow & \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad //\text{mini-batch variance} \\
\hat{x}_i & \longleftarrow & \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{{\sigma_{\mathcal{B}}}^2 + \epsilon}} \qquad\qquad //\text{normalize} \\
y_i & \longleftarrow & \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i) \qquad //\text{scale and shift}
\end{array}
$$

where $\mathcal{B}$ a batch of size m. With the BN transformation, the derivatives in the backpropagation algorithm become as follows:

$$\frac{\partial L}{\partial \hat{x}_i} = \frac{\partial L}{\partial y_i} \cdot \gamma \tag{5}$$

$$\frac{\partial L}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-3/2} \tag{6}$$

$$\frac{\partial L}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial L}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{-2}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}}) \tag{7}$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial L}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2\,(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial L}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m} \tag{8}$$

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \cdot \hat{x}_i \quad \text{and} \quad \frac{\partial L}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \tag{9}$$

Batch Normalization

The training and the inference with Batch-Normalization is made as follow:
Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$.
Output: Batch-normalized network for inference $=$, N_{BN}^{inf}.

1. $N_{BN}^{tr} \longleftarrow N$ // Training BN network
2. for $k = 1, \ldots, K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to N_{BN}^{tr} , using algorithm 1.
4. Modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$ instead.
5. end for
6. Train N_{BN}^{tr} to optimize the parameters $\Theta \bigcup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$.
7. $N_{BN}^{inf} \longleftarrow N_{BN}^{tr}$ // Inference BN network with frozen parameters.

Batch Normalization

1. for $k = 1, \ldots, K$ do
2. // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
3. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathbf{E}[x] \longleftarrow \mathbf{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$Var[x] \longleftarrow \frac{m}{m-1}\mathbf{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

4. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with

$$y = \frac{\gamma}{\sqrt{Var[x] + \epsilon}} \cdot x + \left(\beta - \frac{\gamma \mathbf{E}[x]}{\sqrt{Vr[x] + \epsilon}}\right)$$

5. end for

Instance Normalization

- It is a normalization technique applied to individual samples in a dataset, which normalizes each sample independently across its channels (features).
- In neural network, when data is passed from one layer to an other, each sample undergoes transformations.
- Instance normalization normalizes the activations of each sample across the channels of each layer of the network, ensures that the statistics (mean and variance) of each sample are preserved independently.

Motivation for Instance Normalization

- Address limitations of batch normalization, particularly when it comes to tasks in which preserving the statistics of individual images is crucial, for instance image stylization

Mathematical Formulation

- Let x be the input, a tensor consisting of a batch of N images. Each of these images has C (channels or features) with height (H) and weight (W). Then, $x \in \mathbb{R}^{N \times C \times H \times W}$ is a four-dimensional tensor.

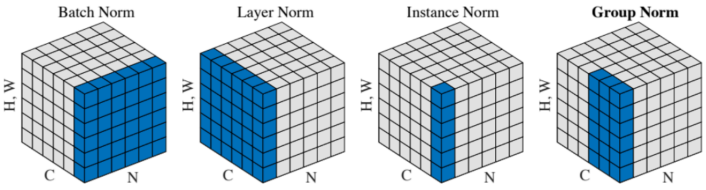

Figure 3: Example of 4− dimensional tensor (from [3]).

- The mathematical formulation involves considering one training sample and feature (in blue in the third image), taking its mean and variance over its Height (H) and weight (W). Then the sample is normalized using the computed statistics. Let us denote
  - x as input tensor
  - t as the number of samples in the tensor
  - i as the number of channels in the tensor
  - H and W as the height and width of the tensor, respectively
  - μ_{ti} as the mean at the ti^{th} channel across the spatial dimensions.
  - σ_{ti} as the standard deviation at the ti^{th} channel across the spatial dimensions.
  - x_{t_lm} represents the pixel value at position (l, m) in the t_i^{th} channel.

We can compute the mean, variance and normalize the sample using the formulation below:

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}},\ \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm},\ \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - m\mu_{ti})^2 \quad (10)$$

Layer Normalization

- Layer normalization (LN) stems from the idea to overcome the limitation of the Batch normalization and again to reduce the learning time of the network.

- A feed-forward network can be considered as a non-linear mapping from a input pattern $\boldsymbol{x}$ to an output vector y, namely

$$a_i^l = w_i^{l\,T} h^l \qquad h_i^{l+1} = f(a_i^l + b_i^l) \tag{11}$$

  where $f(\cdot)$ is an element-wise non linear function, w_i^l is the incoming weights to the i^{th} hidden units, b_i^l is scalar bias parameters, and l stands for the l^{th} layer.
- The parameters of the networks are learned using gradient descent or its variants.

- The sequential nature of Deep Neural Networks makes the gradient w.r.t the weights substantially depends on the outputs of the neurons in the previous layers.
- **The Batch Normalization was proposed to reduce the covariate shift, which normalizes input to each hidden unit over the training cases**. Notably, for the i^th summed input in the l^th layer, this rescales the data under the data distribution

$$\overline{a}_i^l = \frac{g_i^l}{\sigma_i^l}\left(a_i^l - \mu_i^l\right) \qquad \mu_i^l = \underset{\boldsymbol{x}\sim P(\boldsymbol{x})}{\boldsymbol{E}}[a_i^l] \qquad \sigma_i^l = \sqrt{\underset{\boldsymbol{x}\sim P(\boldsymbol{x})}{\boldsymbol{E}}\left[\left(a_i^l - \mu_i^l\right)^2\right]} \tag{12}$$

where $\overline{a}_i^l$ is the normalized summed inputs to the i^{th} hidden unit in the l^{th} layer and g_i is a gain parameter scaling the normalized activation.

- In (12), the expectation is taken over all the training data and it is impractical to compute.
- This requires forward pass through the whole training dataset with the current set of weights. Hence, the statics are estimated using the mini-batch samples.

- To overcome the drawbacks of the batch normalization.
- With the ReLu units, the changes in the output in our layer can entail highly correlated changes in the summed inputs. This covariate shift is fixed by fixing the mean and the variance of the summed inputs within each layer. This motives the layer normalization statistics defined as

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2} \tag{13}$$

  where H is the number of hidden units in a layer, μ^l the mean of the layer.
- In this equation, all the hidden units share the same normalization statistics, μ and σ, unlike (12) the batch normalization.

According to [2], for a standard Recurrent Neural Network(RNN), the summed inputs in the recurrent layer are computed from the current input $\boldsymbol{x}^t$ and the previous vector of hidden states $\boldsymbol{h}^{t-1}$, and are defined as

$$\boldsymbol{a}^t = W_{hh} h^{t-1} + W_{xh} \boldsymbol{x}^t \tag{14}$$

The equation (13) becomes

$$\boldsymbol{h}^t = f\left[\frac{\boldsymbol{g}}{\sigma^t} \circ \left(\boldsymbol{a}^t - \mu^t\right) + \boldsymbol{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2} \tag{15}$$

where W_{hh} is the current hidden to hidden weights and W_{xh} the bottom up inputs to hidden weights and $\circ$ is the Hadamard product between two vectors, $\boldsymbol{b}$ and $\boldsymbol{g}$ are bias and gain parameters, with the same dimension as $\boldsymbol{h}^t$.

$$LN : \mathbb{R}^D \longrightarrow \mathbb{R}^D, \quad LN(\boldsymbol{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{(\boldsymbol{z} - \mu)}{\sigma} \circ \boldsymbol{\alpha} + \boldsymbol{\beta} \tag{16}$$

$$\mu = \frac{1}{D} \sum_{i=1}^{D} z_i, \quad \sigma = \sqrt{\frac{1}{D} \sum_{i=1}^{D} (z_i - \mu)^2} \tag{17}$$

where, z_i is the *ith* element of the vector $\boldsymbol{z}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are adaptive parameters, the gradient being computed by backpropagation. $\boldsymbol{z}$ is the summed input of a given layer.

Advantageous of Normalization

Batch normalization:

- Reduce the overfitting
  - **Regularization effect:** The network becomes less prone to overfitting, allowing it to better generalize to unseen data.
  - **Comparison to the dropout:** in batch-normalized networks, the need for Dropout may be reduced or even eliminated. This is because batch normalization already introduces stochasticity through the mini-batch processing, which serves a similar purpose to Dropout in promoting generalization and reducing overfitting.
- **Speeds up learning :** By reducing internal covariate shift, it helps the model train faster.
- **Allows higher learning rates:** Gradient descent usually requires small learning rates for the network to converge. Batch normalization helps us use much larger learning rates, speeding up the training process.

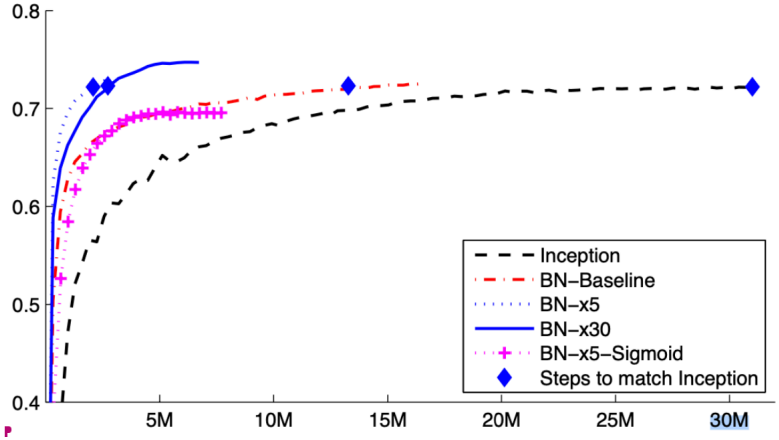

Figure 4: Comparing the performance of the Batch normalized neural network to un-normalized neural network [1].

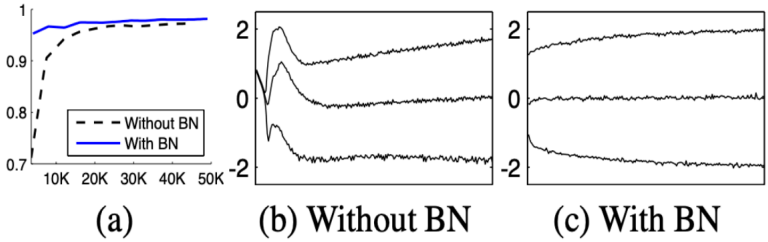

Figure 5: Comparing the performance of the Batch normalized neural network to un-normalized neural network [1].

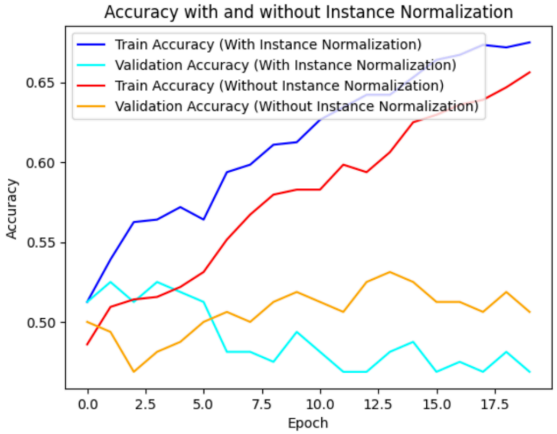

Figure 6: Comparing the performance of the batch-normalized network to un-normalized networks (from authors).

- Test Accuracy (No Normalization): 0.47999998927116394
- Test Accuracy (With Normalization): 0.574999988079071.

Instance Normalization

- Individual samples are normalized independently, the statistics, mean and variance are computed separately.
- Less dependent on batch size; more robust to changes in batch size.
- Suitable for transfer tasks, as it preserves the statistics of individual images.
- Lower computation cost compared to Batch Normalization (BN), it computes the mean and variance separately.
- Utilizes instance statistics to normalize activations, preserving the characteristics of each individual sample separately

Layer Normalization

- Reduce the overfitting, the same as the previous normalization.
- Eliminate the internal covariate shift.
- Speeds up learning, the same as the previous normalization.
- The layers share the same statistics.
- Applicable to RNN.
- No constraints on the size of the batch [2].

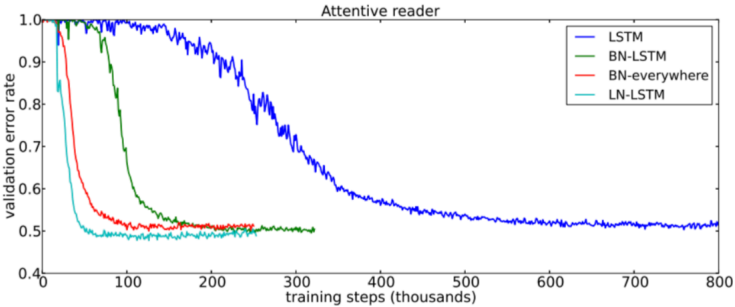

Figure 7: Comparing the performance of BN and LN, for the LSTM .Figure from [2].

- LSTM:

Drawbacks

Batch normalization

- Statistics estimates depend highly on the size of the batch.
- Normalizes activations across the entire mini-batch, computes the mean and the variance over all the training sample in the batch.

Layer Normalization

- Does not perform well for ConvNets

Experimental results

- see notebooks
- https://colab.research.google.com/drive/1mUQEQLWYrV7wv531a5VObPM2I00kBdht?usp=sharing

Conclusion

In this talk, we discussed normalization techniques for neural networks and saw that these techniques are capable to boost up the learning process and avoid on-desirable issues like vanishing and exploding gradient.

References

[1] Sergey and Szegedy Ioffe Christian, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, International conference on machine learning (2015), 448-456. `https://proceedings.mlr.press/v37/ioffe15.pdf`.

[2] Jimmy Lei and Kiros Ba Jamie Ryan and Hinton, *Layer normalization*, arXiv preprint arXiv:1607.06450 (2016).

[3] Dmitry and Vedaldi Ulyanov Andrea and Lempitsky, *Instance normalization: The missing ingredient for fast stylization*, arXiv preprint arXiv:1607.08022 (2016).

Q & A

Thank you!