Published in Techspace

Aakash Bindal Follow
Feb 10, 2019 · 7 min read · Listen

Normalization Techniques in Deep Neural Networks

> *We are going to study Batch Norm, Weight Norm, Layer Norm, Instance Norm, Group Norm, Batch-Instance Norm, Switchable Norm*

Let's start with the question,

Why do we need Normalization ?

Normalization has always been an active area of research in deep learning. Normalization techniques can decrease your model's training time by a huge factor. Let

be unbiased(to higher value features).

2. It reduces **Internal Covariate Shift**. It is the change in the distribution of network activations due to the change in network parameters during training. To improve the training, we seek to reduce the internal covariate shift.
3. In this paper, authors claims that Batch Norm makes loss surface smoother(i.e. it bounds the magnitude of the gradients much more tightly).
4. It makes the Optimization faster because normalization doesn't allow weights to explode all over the place and restricts them to a certain range.
5. An unintended benefit of Normalization is that it helps network in Regularization(only slightly, not significantly).

From above, we can conclude that getting Normalization right can be a crucial factor in getting your model to train effectively, but this isn't as easy as it sounds. Let me support this by certain questions.

1. How Normalization layers behave in Distributed training ?
2. Which Normalization technique should you use for your task like CNN, RNN, style transfer etc ?
3. What happens when you change the batch size of dataset in your training ?
4. Which norm technique would be the best trade-off for computation and accuracy for your network ?

To answer these questions, Let's dive into details of each normalization technique one by one.

Batch Normalization

Batch normalization is a method that normalizes activations in a network across the mini-batch of definite size. For each feature, batch normalization computes the mean and variance of that feature in the mini-batch. It then subtracts the mean and divides the feature by its mini-batch standard deviation.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

this image is taken from https://arxiv.org/pdf/1502.03167.pdf%27

But wait, what if increasing the magnitude of the weights made the network perform better?

To solve this issue, we can add γ and β as scale and shift learn-able parameters respectively. This all can be summarized as:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

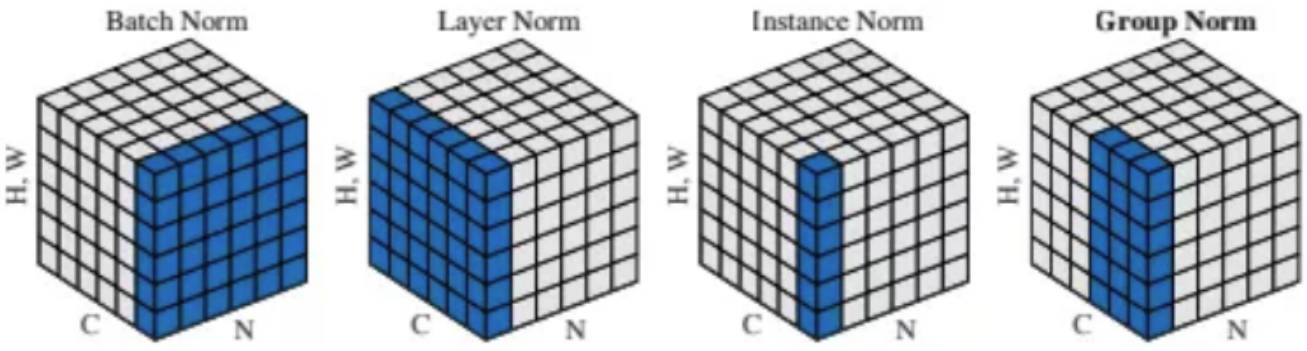
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

ϵ is the stability constant in the equation.

becomes too noisy and training might affect. There would also be a problem in **distributed training**. As, if you are computing in different machines then you have to take same batch size because otherwise γ and β will be different for different systems.

2. **Recurrent Neural Network** → In an RNN, the recurrent activations of each time-step will have a different story to tell(i.e. statistics). This means that we have to fit a separate batch norm layer for each time-step. This makes the model more complicated and space consuming because it forces us to store the statistics for each time-step during training.

> *Batch norm alternatives(or better **norm**s) are discussed below in details but if you only interested in very short description(or revision just by look at an image) look at this :*



A visual comparison of various normalization methods

This image is taken from https://arxiv.org/pdf/1803.08494.pdf

Weight Normalization

Wait, why don't we **normalize weights of a layer** instead of normalizing the activations directly. Well, Weight Normalization does exactly that.

Weight normalization reparameterizes the weights ($\boldsymbol{\omega}$) as :

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

As for the mean, authors of this paper cleverly combine **mean-only batch normalization** and **weight normalization** to get the desired output even in small mini-batches. It means that they subtract out the mean of the minibatch but do not divide by the variance. Finally, they use weight normalization instead of dividing by variance.

> *Note: Mean is less noisy as compared to variance(which above makes mean a good choice over variance) due to the* ***law of large numbers****.*

The paper shows that weight normalization combined with mean-only batch normalization achieves the best results on CIFAR-10.

Layer Normalization

Layer normalization normalizes input across the features instead of normalizing input features across the batch dimension in batch normalization.

A mini-batch consists of multiple examples with the same number of features. Mini-batches are matrices(or tensors) where one axis corresponds to the batch and the other axis(or axes) correspond to the feature dimensions.

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij}$$
$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_{ij} - \mu_i)^2$$
$$\hat{x_{ij}} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

i represents batch and j represents features. $x_{i,j}$ is the i,j-th element of the input data.

The authors of the paper claims that layer normalization performs better than batch norm in case of **RNN**s.

Instance(or Contrast) Normalization

Layer normalization and instance normalization is very similar to each other but the difference between them is that instance normalization normalizes across each channel in each training example instead of normalizing across input features in an training example. Unlike batch normalization, the instance normalization layer is

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

This image is taken from https://arxiv.org/pdf/1607.08022.pdf

Here, $\mathbf{x} \in \mathbb{R}^{T \times C \times W \times H}$ be an input tensor containing a batch of T images. Let $\mathbf{x}_{tijk}$ denote its tijk-th element, where k and j span spatial dimensions(**H**eight and **W**idth of the image), i is the feature channel (color channel if the input is an RGB image), and t is the index of the image in the batch.

This technique is originally devised for **style transfer**, the problem instance normalization tries to address is that the network should be agnostic to the **contrast** of the original image.

Group Normalization

As the name suggests, Group Normalization normalizes over group of channels for each training examples. We can say that, Group Norm is in between Instance Norm and Layer Norm.

∵ When we put all the channels into a single group, group normalization becomes Layer normalization. And, when we put each channel into different groups it becomes Instance normalization.

$$\mu_i = \frac{1}{m}\sum_{k \in \mathcal{S}_i} x_k, \quad \sigma_i = \sqrt{\frac{1}{m}\sum_{k \in \mathcal{S}_i}(x_k - \mu_i)^2 + \epsilon},$$

S_i is defined below.

$$\mathcal{S}_i = \{k \mid k_N = i_N, \lfloor \frac{k_C}{C/G} \rfloor = \lfloor \frac{i_C}{C/G} \rfloor\}.$$

$$y_i = \gamma \hat{x}_i + \beta,$$

Here, **x** is the feature computed by a layer, and i is an index. In the case of 2D images, **i** = (**i**N , **i**C , **i**H, **i**W) is a 4D vector indexing the features in (N, C, H, W) order, where N is the batch axis, **C** is the channel axis, and **H** and **W** are the spatial height and width axes. **G** is the number of groups, w[illegible]ed hyper-parameter. C/G is the number of channels per group. ⌊.⌋ is the floor operation, and "⌊**kC/(C/G)**⌋= ⌊**iC/(C/G)**⌋" means that the indexes **i** and **k** are in the same group of channels, assuming each group of channels are stored in a sequential order along the C axis. GN computes μ and σ along the (H, W) axes and along a group of C/G channels.

Batch-Instance Normalization

The problem with Instance normalization is that it completely erases style information. Though, this has its own merits(such as in style transfer) it can be problematic in those conditions where contrast matters(like in weather classification, brightness of the sky matters). Batch-instance normalization attempts to deal with this by learning how much style information should be used for each channel(**C**).

Batch-Instance Normalization is just an interpolation between batch norm and instance norm.

$$\mathbf{y} = \left(\rho \cdot \hat{\mathbf{x}}^{(B)} + (1-\rho) \cdot \hat{\mathbf{x}}^{(I)}\right) \cdot \gamma + \beta,$$

the value of ρ is in between 0 and 1.

The interesting aspect of batch-instance normalization is that the balancing parameter ρ is learned through gradient descent.

From batch-instance normalization, we can conclude that models could learn to adaptively use different normalization methods using gradient descent.

> *Understanding from above, a question may arise.*

Can we switch the normalization technique whenever needed ?

This paper proposed switchable normalization, a method that uses a weighted average of different mean and variance statistics from batch normalization, instance normalization, and layer normalization.

The authors showed that switch normalization could potentially outperform batch normalization on tasks such as image classification and object detection.

The paper showed that the instance normalization were used more often in earlier layers, batch normalization was preferred in the middle and layer normalization being used in the last more often. Smaller batch sizes lead to a preference towards layer normalization and instance normalization.

References

1. https://arxiv.org/pdf/1502.03167.pdf%27
2. https://arxiv.org/pdf/1607.06450.pdf
3. https://arxiv.org/pdf/1602.07868.pdf
4. https://arxiv.org/pdf/1607.08022.pdf
5. https://arxiv.org/pdf/1803.08494.pdf
6. https://arxiv.org/pdf/1805.07925.pdf
7. https://arxiv.org/pdf/1811.07727v1.pdf


PUBLISHED ON AUGUST 2, 2019
IN DEVELOPERS CORNER (HTTPS://ANALYTICSINDIAMAG.COM/CATEGORY/DEVELOPERS_CORNER/)

Understanding Normalisation Methods In Deep Learning

BY AMBIKA CHOUDHURY(HTTPS://ANALYTICSINDIAMAG.COM/AUTHOR/AMBIKA-CHOUDHURY/)

recognition, computer vision, machine translation, among others. However, training deep learning models [illegible] a complex task as, during the training phase, inputs of each layer keep changing.

(https://analyticsindiamag.com)

Normalization is an approach which is applied during the preparation of data in order to change the values of numeric columns in a dataset to use a common scale when the features in the data have different ranges. In this article, we will discuss the various normalization methods which can be used in deep learning models.

Let us take an example, suppose an input dataset contains data in one column with values ranging from 0 to 10 and the other column with values ranging from 100,000 to 10,00,000. In this case, the input data contains a big difference in the scale of the numbers which will eventually occur as errors while combining the values as features during modelling. These issues can be mitigated by normalization (https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data) by creating new values and maintaining the general or normal distribution in the data.

There are several approaches in normalisation which can be used in deep learning models. They are mentioned below

Batch Normalization

Batch normalization is one of the popular normalization methods used for training deep learning models. It enables faster and stable training of deep neural networks by stabilising the distributions of layer inputs during the training phase. This approach is mainly related to internal covariate shift (ICS) where internal covariate shift means the change in the distribution of layer inputs caused when the preceding layers are updated. In order to improve the training in a model, it is important to reduce the internal co-variant shift. The batch normalization works here to reduce the internal covariate shift by adding network layers which control the means and variances of the layer inputs.

Advantages

The advantages of batch normalization are mentioned below:

- Batch normalization reduces the internal covariate shift (ICS) and accelerates the training of a deep [illegible] network
- This [illegible] reduces the dependence of gradients on the scale of the parameters or of their initial values [illegible] higher learning rates without the risk of divergence
- Batch Normalisation makes it possible to use saturating nonlinearities by preventing the network from getting stuck in the saturated modes

Click here (https://arxiv.org/pdf/1502.03167.pdf) to know more

Weight Normalization

Weight normalization is a process of reparameterization of the weight vectors in a deep neural network which works by decoupling the length of those weight vectors from their direction. In simple terms, we can define weight normalization as a method for improving the optimisability of the weights of a neural network model.

Advantages

The advantages of weight normalization are mentioned below

- Weight normalization improves the conditioning of the optimisation problem as well as speed up the convergence of stochastic gradient descent.
- It can be applied successfully to recurrent models such as LSTMs as well as in deep reinforcement learning or generative models

Click here (https://papers.nips.cc/paper/6114-weight-normalization-a-simple-reparameterization-to-accelerate-training-of-deep-neural-networks.pdf) to know more

Layer Normalization

Layer normalization is a method to improve the training speed for various neural network models. Unlike batch normalization, this method directly estimates the normalisation statistics from the summed inputs to the neurons within a hidden layer. Layer normalization is basically designed to overcome the drawbacks of batch normalization such as dependent on mini batches, etc.

Advantages

The advantages of layer normalization are mentioned below:

- Layer normalization can be easily applied to recurrent neural networks by computing the normalization statistics separately at each time step
- This approach is effective at stabilising the hidden state dynamics in recurrent networks

Click here (https://arxiv.org/abs/1607.06450) to know more

Group Normalization

Group normalization can be said as an alternative to batch normalization. This approach works by dividing the channels into groups and computes within each group the mean and variance for normalization i.e. normalization is done across width, height and each group. Unlike batch normalization, group normalization is independent of batch sizes, and also its accuracy is stable in a wide range of batch sizes.

Advantages

The advantages of group normalization are mentioned below:

- It has the ability to replace batch normalization in a number of deep learning tasks
- It can be easily implemented in modern libraries with just a few lines of codes

Click here (https://arxiv.org/abs/1803.08494) to know more

Instance Normalization

Instance normalization, also known as contrast normalization is almost similar to layer normalization. Unlike batch normalization, instance normalization is applied to a whole batch of images instead for a single one.

Advantages

The advantages of instance normalization are mentioned below

- This normalization simplifies the learning process of a model.
- The instance normalization can be applied at test time.

Click here (https://arxiv.org/abs/1607.08022) to know more

motion-capture-for-animation-using-first-order-motion-model/)

(https://analyticsindiamag.com)

(https://analyticsindiamag.com/author/ambika-choudhury/)

A Technical Journalist who loves writing about Machine Learning and Artificial Intelligence. A lover of music, writing and learning something out of the box.

Our Upcoming Events

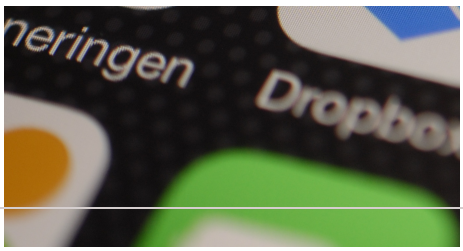
MOST POPULAR

Are Banned Apps Really Banned? (https://analyticsindiamag.com/are-banned-apps-really-banned/)

Bhuvana Kamath

A whopping 348 apps are banned in India to uphold 'the security of the state and public order' but are still available as cracked versions on the web

(https://analyticsindiamag.com)



(https://analyticsindiamag.com/are-ban

Questioning the Hype About AlphaFold (https://analyticsindiamag.com/questioning-the-hype-about-alphafold/)

Poulomi Chatterjee

Protein structures innately shift and transform, sometimes very drastically and sometimes in subtle ways when small-molecule ligands show up.

(https://analyticsindian

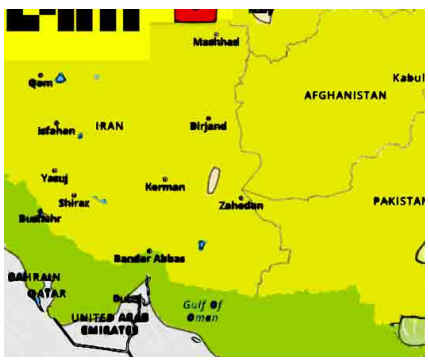
Data Warehouses in Age of Decentralised Architecture (https://analyticsindiamag.com/data-warehouses-in-age-of-decentralised-architecture/)

Zinnia Banerjee

According to Sigmoid, a data engineering and AI solutions company, adoption of cloud data warehouses is growing at a CAGR of 15%.

(https://analyticsindiamag.com/data-

How MapMyIndia Builds Its Maps —and What It Means for the Future (https://analyticsindiamag.com/how-mapmyindia-builds-its-maps-and-what-it-means-for-the-future/)



Tausif Alam

ory of more than two crore data points, (https://analyticsindiamag.com)
inc lematics, and 3D data visualisations

(https://analyticsindiamag.com/how

Will Web3 Change the Monetisation Game for Content Creators? (https://analyticsindiamag.com/will-web3-change-the-monetisation-game-for-content-creators/)

Tausif Alam

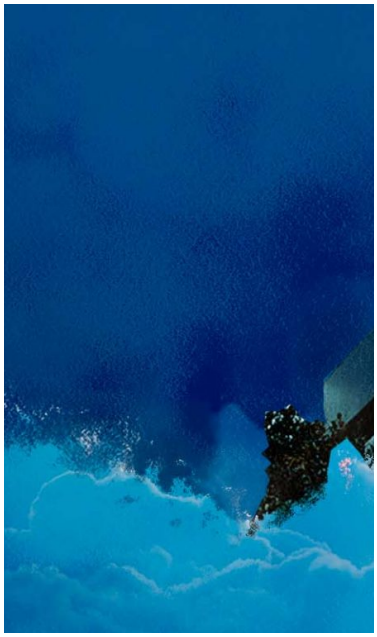
Of all the content creators, only a minority have been able to successfully build the direct audience base

(https://analyticsindiamag.com/will-w

Infosys' AI & Analytics Play (https://analyticsindiamag.com/infosys-ai-analytics-play/)

Amit Raja Naik

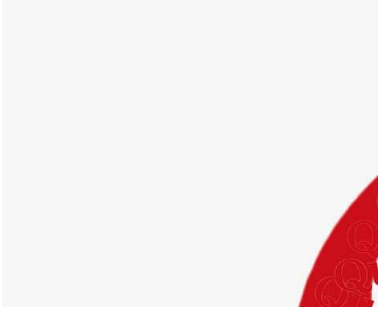
At Infosys, AI and automation are vital deliverables that are offered, including development, maintenance, testing, migration, modernisation, implementation, rollout and more.



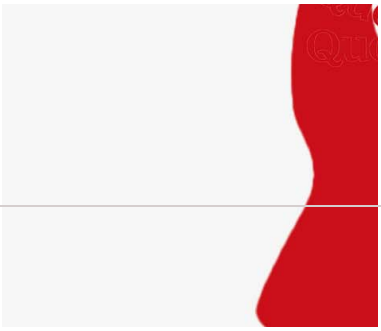
(https://analyticsindiamag.com/

Quora's ML System is a Class Act (https://analyticsindiamag.com/quoras-ml-system-is-a-class-act/)

Shraddha Goled



There are several machine learning algorithms that are running behind the ... tain its position as one of the most popular web... a ... de... s launch. (https://analyticsindiamag.com)



(https://analyticsindiamag.com/c

Behind the Growing (and Huge) IT CEO-Fresher Pay Gap (https://analyticsindiamag.com/behind-the-growing-and-huge-it-ceo-fresher-pay-gap/)

Lokesh Choudhary

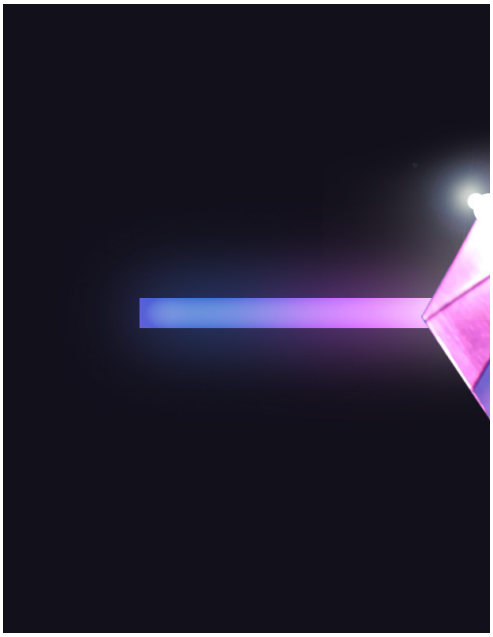
The salaries of top tech CEOs around the world have increased by more than 800% in the last decade, while freshers' salaries rose by a paltry 45%. Does this make you question the status quo?

(https://analyticsindiamag.com/

6 Crazy AI Experiments to Try out Online (https://analyticsindiamag.com/6-crazy-ai-experiments-to-try-out-online/)

Lokesh Choudhary

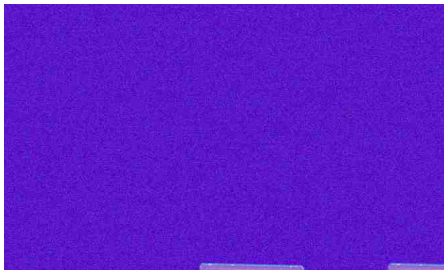
AI is interesting, and it's evolving. Gone are the days when it was only scientists who experimented with it.



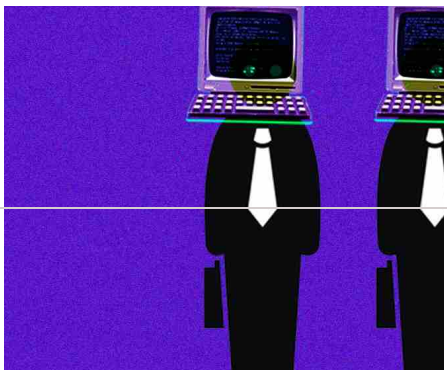
(https://analyticsindiamag.com/6-crazy-ai-

End of Employment Bonds? (https://analyticsindiamag.com/end-of-employment-bonds/)

Shraddha Goled



Food delivery app Swiggy recently announced the 'Moonlight Policy'
oyees to take up second jobs
(https://analyticsindiamag.com)



(https://analyticsindiamag.com/end-of

(https://analyticsindiamag.com)

Data S[illegible]

40 under 40 Data Scientists

Women in AI Leadership

EVENTS

Our Events

AIM Custom Events

AIM Virtual

FOR ML DEVELOPERS

Hackathons

Discussion Forum

Job Portal

Mock Assessments

Practice ML

Free AI Courses

NEWSLETTER

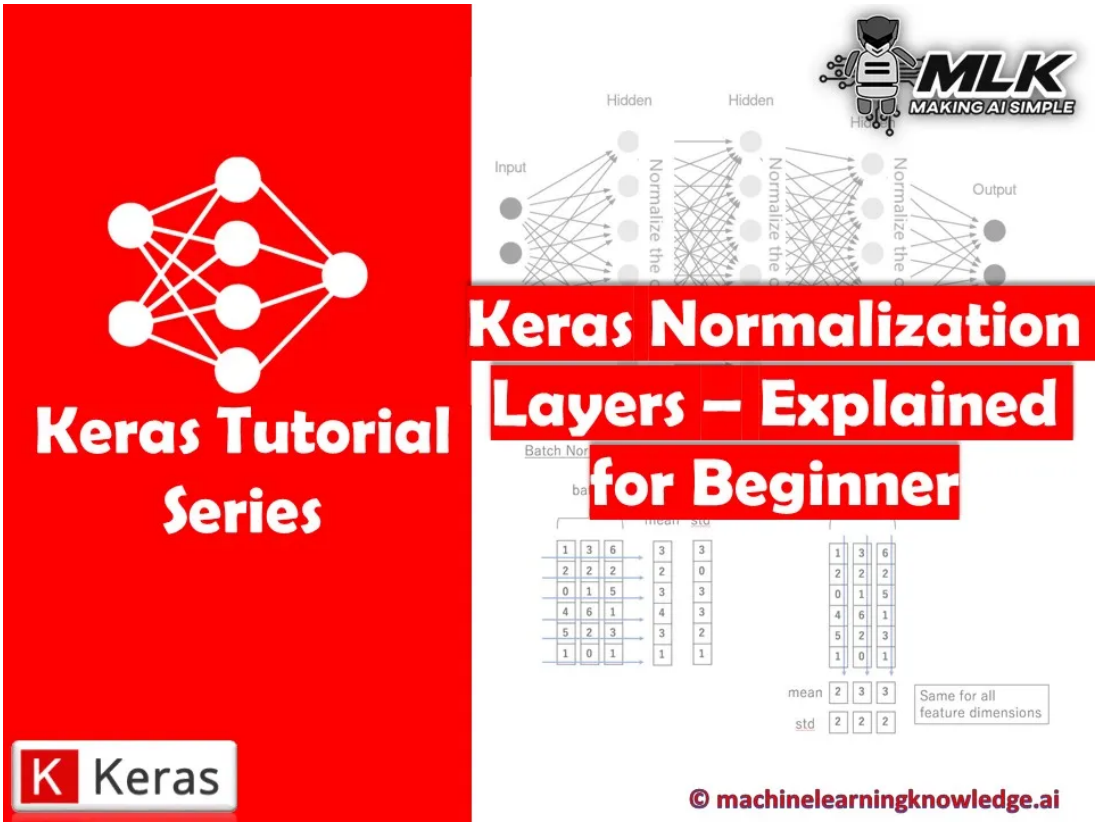
Stay up to date with our latest news, receive exclusive deals, and more.

Enter Your Email Address

SUBSCRIBE →

Keras Normalization Layers- Batch Normalization and Layer Normalization Explained for Beginners

By **Palash Sharma** - December 12, 2020



Keras Normalization Layers- Batch Normalization and Layer Normalization Explained for Beginners

Contents [hide]

Introduction

In this article, we will go through the tutorial for Keras Normalization Layer where will understand why a normalization layer is needed. We will also see what are the two types of normalization layers in Keras – i) Batch Normalization Layer and ii) Layer Normalization Layer and understand them in detail with the help of examples. We will finally do a comparison between batch normalization vs layer normalization with their advantages and disadvantages.

But before going into that let us first understand why Normalization is needed in the first place.

What is Normalization?

Normalization is a method usually used for preparing data before training the model. The main purpose of normalization is to provide a **uniform scale for numerical values**. If the dataset contains numerical data varying in a huge range, it will skew the learning process, resulting in a bad model. The normalization method ensures there is **no loss of information** and even the **range of values** isn't affected.

Normalization is done by the below formula, by subtracting the mean and dividing by the standard deviation.

- **Also Read** – Data Preprocessing in Neural Network for Beginners

In spite of normalizing the input data, the value of activations of certain neurons in the hidden layers can start varying across a wide scale during the training process. This means the input to the neurons to the next hidden layer will also range across the wide range, bringing instability.

To deal with this problem, we use the techniques of "**batch normalization**" layer and "**layer normalization**" layer. Let us see these two techniques in detail along with their implementation examples in Keras.

Batch Normalization Layer

batch normalization (Source)

Batch Normalization Layer is applied for neural networks where the training is done in mini-batches. We divide the data into batches with a certain batch size and then pass it through the network. Batch normalization is applied on the neuron activation for all the samples in the mini-batch such that the mean of output lies close to 0 and the standard deviation lies close to 1. It also introduces two learning parameters gama and beta in its calculation which are all optimized during training.

Advantages of Batch Normalization Layer

1. Batch normalization improves the training time and accuracy of the neural network.
2. It decreases the effect of weight initialization.
3. It also adds a regularization effect on the network.
4. It works better with the fully Connected Neural Network (FCN) and Convolutional Neural Network.

Disadvantages of Batch Normalization Layer

1. Batch normalization is dependent on mini-batch size which means if the mini-batch size is small, it will have little to no effect
2. If there is no batch size involved, like in traditional gradient descent learning, we cannot use it at all.
3. Batch normalization does not work well with Recurrent Neural Networks (RNN)

Syntax for Batch Normalization Layer in Keras

```
 tf.keras.layers.BatchNormalization(
axis=-1,
momentum=0.99,
epsilon=0.001,
center=True,
scale=True,
beta_initializer="zeros",
gamma_initializer="ones",
moving_mean_initializer="zeros",
moving_variance_initializer="ones",
beta_regularizer=None,
gamma_regularizer=None,
beta_constraint=None,
gamma_constraint=None,
renorm=False,
renorm_clipping=None,
renorm_momentum=0.99,
fused=None,
trainable=True,
virtual_batch_size=None,
adjustment=None,
name=None,
**kwargs)
```

Keras Batch Normalization Layer Example

In this example, we'll be looking at how **batch normalization layer** is implemented.

First, we load the libraries and packages that are required. We also import **kmnist dataset** for our implementation.

Install Keras Dataset

In [1]:

```
! pip install extra_keras_datasets
```

```
Collecting extra_keras_datasets
  Downloading https://files.pythonhosted.org/packages/5d/a1/7cf6d48290468aa3fd5a31780c9e44
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from extra
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from extra
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages (from extr
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from py
Installing collected packages: extra-keras-datasets
Successfully installed extra-keras-datasets-1.2.0
```

Import Libraries

In [2]:

```
from extra_keras_datasets import kmnist
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
```

Model Configuration

In the below code snippet we are specifying the **batch size as 250**, the **number of epochs executed is 25**, the data will be classified into **10 different classes**, **20% of the training data** is used as the **validation set** and lastly, **verbosity is set to true**.

In [3]:

```
# Model configuration
batch_size = 250
no_epochs = 25
no_classes = 10
validation_split = 0.2
verbosity = 1
```

Download and Load Dataset

Next, we load the **MNIST** dataset from **Keras datasets module**.

In [4]:

```
# Load KMNIST dataset
(input_train, target_train), (input_test, target_test) = kmnist.load_data(type='kmnist')
```

```
Downloading data from http://codh.rois.ac.jp/kmnist/dataset/kmnist/kmnist-train-imgs.npz
18391040/18384171 [==============================] - 4s 0us/step
Downloading data from http://codh.rois.ac.jp/kmnist/dataset/kmnist/kmnist-train-labels.npz
32768/29700 [================================] - 0s 5us/step
```

```
Downloading data from http://codh.rois.ac.jp/kmnist/dataset/kmnist/kmnist-test-imgs.npz
3080192/3079479 [==============================] - 2s 1us/step
Downloading data from http://codh.rois.ac.jp/kmnist/dataset/kmnist/kmnist-test-labels.npz
8192/5304 [==============================================] - 0s 0us/step
```

Here we process the data, this helps it in preparing data for training.

In [5]:

```
# Shape of the input sets
input_train_shape = input_train.shape
input_test_shape = input_test.shape
```

Now we are obtaining the shape of both training and testing datasets. We also set the shape of keras input data.

In [6]:

```
# Keras layer input shape
input_shape = (input_train_shape[1], input_train_shape[2], 1)
```

To include channels during training, we are using the below code. There are instances when there are no apt channels, so we need to reshape the data for including those channels.

In [7]:

```
# Reshape the training data to include channels
input_train = input_train.reshape(input_train_shape[0], input_train_shape[1], input_train_
input_test = input_test.reshape(input_test_shape[0], input_test_shape[1], input_test_shape
```

The pre-processing of data is almost completed, we just need to convert data type to **float32** format, it definitely makes the **process faster**.

In [8]:

```
# Parse numbers as floats
input_train = input_train.astype('float32')
input_test = input_test.astype('float32')
```

At last, we **normalize the data** for better results.

In [9]:

```
# Normalize input data
input_train = input_train / 255
input_test = input_test / 255
```

Designing the model architecture

In this section, we will be designing our neural network using the Sequential API of Keras. Notice we have used **batch normalization layers** in the design. Along with that, we have also used dense, convolutional, and pooling layers in the architecture.

In [10]:

```
# Create the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(no_classes, activation='softmax'))
```

Model compilation & fitting data

Now we need to compile our model. Model Compilation helps in instantiating the model architecture. In this step we are using sparse categorical crossentropy loss, adam optimizer is used for model optimization.

In [11]:

```
# Compile the model
model.compile(loss=tensorflow.keras.losses.sparse_categorical_crossentropy,
              optimizer=tensorflow.keras.optimizers.Adam(),
              metrics=['accuracy'])
```

Training the Model

At last, we are fitting the data to our model for training.

Here the input training set is provided corresponding targets, and then it is trained to pre-decided batch_size and number of epochs.

In [12]:

```
# Fit data to model
history = model.fit(input_train, target_train,
            batch_size=batch_size,
            epochs=no_epochs,
            verbose=verbosity,
            validation_split=validation_split)
```

Output:

```
Epoch 1/25
192/192 [==============================] - 52s 269ms/step - loss: 0.2153 - accuracy: 0.937
Epoch 2/25
192/192 [==============================] - 52s 270ms/step - loss: 0.0504 - accuracy: 0.987
Epoch 3/25
192/192 [==============================] - 52s 269ms/step - loss: 0.0205 - accuracy: 0.996
Epoch 4/25
192/192 [==============================] - 52s 270ms/step - loss: 0.0083 - accuracy: 0.999
Epoch 5/25
```

```
192/192 [==============================] - 52s 270ms/step - loss: 0.0033 - accuracy: 0.999
Epoch 6/25
192/192 [==============================] - 52s 270ms/step - loss: 0.0015 - accuracy: 1.000
Epoch 7/25
192/192 [==============================] - 52s 269ms/step - loss: 9.4551e-04 - accuracy: 1
Epoch 8/25
192/192 [==============================] - 57s 299ms/step - loss: 6.4900e-04 - accuracy: 1
Epoch 9/25
192/192 [==============================] - 52s 269ms/step - loss: 4.9284e-04 - accuracy: 1
Epoch 10/25
192/192 [==============================] - 52s 269ms/step - loss: 3.9835e-04 - accuracy: 1
Epoch 11/25
192/192 [==============================] - 52s 269ms/step - loss: 3.0890e-04 - accuracy: 1
Epoch 12/25
192/192 [==============================] - 52s 268ms/step - loss: 2.5462e-04 - accuracy: 1
Epoch 13/25
192/192 [==============================] - 52s 269ms/step - loss: 2.2261e-04 - accuracy: 1
Epoch 14/25
192/192 [==============================] - 52s 270ms/step - loss: 1.8208e-04 - accuracy: 1
Epoch 15/25
192/192 [==============================] - 52s 270ms/step - loss: 1.5766e-04 - accuracy: 1
Epoch 16/25
192/192 [==============================] - 52s 270ms/step - loss: 1.3558e-04 - accuracy: 1
Epoch 17/25
192/192 [==============================] - 52s 269ms/step - loss: 1.1254e-04 - accuracy: 1
Epoch 18/25
192/192 [==============================] - 52s 269ms/step - loss: 1.0031e-04 - accuracy: 1
Epoch 19/25
192/192 [==============================] - 51s 268ms/step - loss: 8.5360e-05 - accuracy: 1
Epoch 20/25
192/192 [==============================] - 51s 267ms/step - loss: 7.5436e-05 - accuracy: 1
Epoch 21/25
192/192 [==============================] - 51s 267ms/step - loss: 6.5648e-05 - accuracy: 1
Epoch 22/25
192/192 [==============================] - 51s 267ms/step - loss: 5.8587e-05 - accuracy: 1
Epoch 23/25
192/192 [==============================] - 52s 269ms/step - loss: 5.1551e-05 - accuracy: 1
Epoch 24/25
192/192 [==============================] - 52s 271ms/step - loss: 4.4635e-05 - accuracy: 1
Epoch 25/25
192/192 [==============================] - 52s 270ms/step - loss: 3.8689e-05 - accuracy: 1
```

Generating evaluation metrics

Now we will evaluate the performance of our model.

In [13]:

```
# Generate generalization metric  s
score = model.evaluate(input_test, target_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

Output:

```
Test loss: 0.22573837637901306 / Test accuracy: 0.9553999900817871
```

Layer Normalization Layer

Batch Normalization vs Layer Normalization (Source)

The next type of normalization layer in Keras is **Layer Normalization** which addresses the drawbacks of batch normalization. This technique is not dependent on batches and the normalization is applied on the neuron for a single instance across all features. Here also mean activation remains close to 0 and mean standard deviation remains close to 1.

Advantages of Layer Normalization

1. It is not dependent on any batch sizes during training.
2. It works better with Recurrent Neural Network.

Disadvantages of Layer Normalization

1. It may not produce good results with Convolutional Neural Networks (CNN)

Syntax of Layer Normalization Layer in Keras

```
tf.keras.layers.LayerNormalization(
 axis=-1,
 epsilon=0.001,
 center=True,
 scale=True,
 beta_initializer="zeros",
 gamma_initializer="ones",
 beta_regularizer=None,
 gamma_regularizer=None,
 beta_constraint=None,
 gamma_constraint=None,
 trainable=True,
 name=None,
**kwargs)
```

Keras Normalization Layer Example

Let us see the example of how does LayerNormalization works in Keras. For this, we will be using the same dataset that we had used in the above example of batch normalization. Hence we are skipping the data download and preprocessing part for which you can refer to the above example. We will directly go to designing and training the neural network.

Designing the model architecture

In [15]:

```
import tensorflow as tf
```

In [16]:

```
model_lay = tf.keras.models.Sequential([
  # Reshape into "channels last" setup.
  tf.keras.layers.Reshape((28,28,1), input_shape=(28,28)),
  tf.keras.layers.Conv2D(filters=10, kernel_size=(3,3),data_format="channels_last"),
  # LayerNorm Layer
  tf.keras.layers.LayerNormalization(axis=3 , center=True , scale=True),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

Compiling the Model

In [17]:

```
# Compiling Layer Normalization Model
model_lay.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Fitting the Model

In [19]:

```
# Fitting Layer Normalization Model
his_lay = model_lay.fit(input_train, target_train)
```

Output:

```
1875/1875 [==============================] - 41s 22ms/step - loss: 0.3809 - accuracy: 0.88
```

Evaluating Layer Normalization Model

In [20]:

```
# Generate generalization metric  s
score = model_lay.evaluate(input_test, target_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

Output:

```
Test loss: 0.45526155829429626 / Test accuracy: 0.8698999881744385
```

Comparing the results of two methods

As we look at the accuracy of the two methods on test data, we can see that **batch normalization achieved 96% accuracy** whereas **layer normalization achieved 87% accuracy.** We can see that on CNN, batch normalization produced better results than layer normalization.

Batch Normalization vs Layer Normalization

Before wrapping up, let us summarise the key difference between Batch Normalization and Layer Normalization in deep learning that we discussed above.

1. Batch Normalization depends on mini-batch size and may not work properly for smaller batch sizes. On the other hand, Layer normalization does not depend on mini-batch size.
2. In batch normalization, input values of the same neuron for all the data in the mini-batch are normalized. Whereas in layer normalization, input values for all neurons in the same layer are normalized for each data sample.
3. Batch normalization works better with fully connected layers and convolutional neural network (CNN) but it shows poor results with recurrent neural network (RNN). On the other hand, the main advantage of Layer normalization is that it works really well with RNN.

- **Also Read** – Different Types of Keras Layers Explained for Beginners
- **Also Read** – Keras Dropout Layer Explained for Beginners
- **Also Read** – Keras Dense Layer Explained for Beginners
- **Also Read** – Keras Convolution Layer – A Beginner's Guide
- **Also Read –** Beginners's Guide to Keras Models API
- **Also Read** – Types of Keras Loss Functions Explained for Beginners

Conclusion

In this tutorial, we learned about the Keras normalization layer and its different types i.e. batch normalization and layer normalization. We saw the syntax, examples, and did a detailed comparison of batch normalization vs layer normalization in Keras with their advantages and disadvantages.

Reference Keras Documentation

Related Terms:

- Term: Numerical Data
- Term: Artificial Neuron
- Term: Artificial Neural Network
- Term: Class
- Term: Deep Learning

Palash Sharma

I am Palash Sharma, an undergraduate student who loves to explore and garner in-depth knowledge in the fields like Artificial Intelligence and Machine Learning. I am captivated by the wonders these fields have produced with their novel implementations. With this, I have a desire to share my knowledge with others in all my capacity.