Layer Normalization

Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton

Summary
Layer normalization enhances batch normalization with that it gives a method to use it in recurrent neural networks (NN) and to avoid differences at test and train times. It keeps the same advantages as batch normalization, i.e. better and faster convergence, but now it computes statistics (mean and variance) for all units in a layer using a single example. These statistics are used to normalize all input to neurons. The bias and gain (learnable parameters) are applied before ReLU (or other non-linearity function) to ease the normalization. The work presents experiments with layer normalization on different problems.

1. Introduction.
The problem is that it takes a long time to train deep NN (days, weeks). One approach is to parallelize the work, but that keeps the same amount of computation and increases complexity. Another approach is to normalize input for each neuron using statistics from a summed input. After that, the efficiency of convergence increases. Batch normalization does it, but it is unclear how effectively to apply it for recurrent NN as it uses batches, and also it can't be used in online learning tasks as those batches should be small. Layer normalization addresses these issues.
2. Background.
In NN we have input data *X* and output predictions *y*. If we don't apply any normalization, then there is 'noise' or high correlation between layers ('covariate shift') with neurons (units) when we compute gradients. For computing gradients, feed-forward and backward propagation are used. So normalization changes values of the input by trying to make it zero centered (subtract mean) and similarly scaled (dividing by standard deviation). Now the problem is how to get these statistics. It is inefficient to get it using all examples, so batch normalization uses random mini-batches to get it.
3. Layer normalization.
They compute mean and variance over the sum of input to units in one layer using one example. Unlike batch normalization they don’t depend on batches. So the mean and the variance are

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

where *H* is the number of units in a layer, *a* is input to a unit.
3.1. It is problematic to store statistics for each time step when we use batch normalization in recurrent NN. With layer normalization, it is now clear how to use it in RNN as statistics are per layer not per batch, and we compute it per every time step.
4. Related work. Previous works include the one on how to make batch normalization work with RNN, on weights normalization, on re-parameterization.
5. Analysis. On properties of different normalization.
5.1. Invariance under weight and data transformations. This is about how mean, variance, bias (learned parameter) and gain (learned parameter) are calculated for layer, batch and

weight normalization. Under weights normalization, the mean for the input is zero and the variance is the length of the weights vector.
5.2. Geometry of parameter space during learning. About how the variance (the normalization value) can reduce learning rate and make learning less random.
5.2.1. This is about a metric that shows the parameters space. It will be further used to test how weights change.
5.2.2. This is an analysis of how weights / parameters space changes with respect to the normalization.
6. Experiment results. Six problems were addressed with the focus on RNN including digits classification, language modeling, image-sentence ranking, and others.
6.1. Order embeddings of images and language. This is about how layer normalization performs for image and text entailment (finding out logical order between words or images).
6.2. Teaching machines to read and comprehend. About experiment with layer normalization for the problem of filling blanks in a text.
6.3. Skip-thought vectors. For the problem of generating surrounding sentences for a given sentence.
6.4. Modeling binarized MNIST using DRAW. For the problem of filling missing pieces of an image (digits).
6.5. Handwriting sequence generation. To test longer sentences in RNN.
6.6. Permutation invariant MNIST. To test on feed-forward NN.
6.7. On performance in convolutional NN. Results didn't show much advantage over batch normalization. Further research needed to make layer normalization perform well enough.
7. Conclusion. They proposed layer normalization. They theoretically prove that it works compared to batch normalization and weight normalization. They presented experiments that show that layer normalization works best on RNN.

Judge
Advantages of layer normalization are

1) It can be done “online” , that is we don't need to wait for a full batch of examples to be loaded for the statistics to be calculated.
2) Hence it can be easily used in recurrent NN where there is a complex dependency between layers and batch normalization has no easy way to be applied.
3) There is no dependency on batch size as in batch normalization that gets those statistics as scalars at the test time hence it might be a source of bugs at test time.

Disadvantages of layer normalization are

1) There is no proof that layer normalization works better than batch normalization in convolutional NN.
2) It is unclear how layer normalization works better if there are large differences between examples. Perhaps there will be less use of normalization hence large values for gradients, etc. This needs to be tested.

Deep Residual Learning for Image Recognition

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

About how to train deep neural networks (NN) (100-1000 layers). It introduces residual blocks (residual functions) for training large models. The problem addressed is that it is a worse or even degrading learning if there are many layers. They solve it and prove it works on empirical data. They won several awards in computer vision competitions.

1. Introduction.
This paper addresses the issue in deep convolutional networks when they show worse accuracy than shallow NN, whereas it is expected that deeper networks will perform better. Training deep NN became possible because of the introduction of normalization, i.e. the calculated gradients were not large then. But those NN degraded in accuracy. It was strange because those NN were expected to perform at least at the same level as the same level of accuracy can be achieved if we copy layers of a trained shallow network and add identity mapping layers (they just copy inputs) on top of those. They proposed to add the input that was given to a stack of layers to the output of those layers:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

where *x* is the input vector, *F* is the output vector. This is called a shortcut connection. It proved to fix the problem with degrading accuracy. Intuition behind this is that NN has difficulty learning identity mapping so this shortcut connection adds this identity copying. The awards they won is proof that this is a general solution for deep NN.
2. Related work includes works on residual representations, shortcut connections (works on Perceptron, other).
3. Deep Residual Learning
3.1 Residual Learning. In theory it should be the same to fit *H(x)* and *H(x) - x* , where x is an input for a few stacked layers and *H(x)* is a hypothesis function (with learnable weights), but the ease of learning will be different. They suggest that NN has difficulty training identity mapping (copying data) thus if we add the data on the end of the block it should be different learning.
3.2 Identity mapping by shortcuts. This is that it adds the input at the end of the residual block. Dimensions should match or otherwise additional computation for that required.
3.3 Network Architectures. They call 'Plain network' the one that follows VGG rules: same output feature map sizes, same number of filters; if size halves, the filters doubled. Residual Network is the one with these shortcut connections.
3.4 Implementation. They used implementations similar to the previous ones.
4. Experimentations.
4.1 ImageNet Classification. Experiments with different architectures on ImageNet dataset. Deeper network with extra 16 layers shows 3.51% better accuracy. They tested plain networks, residual networks, identity shortcuts (keeping dimensions), projection shortcuts (changing dimensions). And they construct deeper NN to test on ImageNet. Result was the deeper the network the better accuracy. Comparison with the state-of-the-art models show their better performance on the dataset.

4.2 Cifar-10. Results show that the residual networks perform much better in training and in testing times. They found that too deep a network becomes unnecessarily large for such a simple problem as Cifar-10 hence a need for dropout and other.
4.3 Object detection. They used PASCAL VOC 2004 and COCO datasets. Here they also show good accuracy on recognition tasks.

Judge
Advantages:

1) Can train deep convolutional NN which was not possible before them. So this is a milestone for deep NN.
2) They outperformed all previous architectures because of their much deeper NN.
3) Adding the copy of the input from the previous layer is less demanding on resources, supports backpropagation and doesn't require a state (learnable weights).

Disadvantages:

1) Input and output should be of the same dimensions or otherwise extra computation is needed (projection).
2) Adds complexity to the architecture (source of possible bugs).
3) Care should be taken not to introduce too deep architecture for a simple module. Otherwise the need of dropout or similar method to avoid overfitting.