Layer Normalization

AbdelAziz Ben Othman

Abstract

Training state-of-the-art, machine learning models is computationally expensive. Layer Normalization is a widely adopted technique that enables faster and stable training of the deep neural networks. It controls the distribution of the input's data by giving each neuron its own adaptive bias and gain which are applied after the normalization. Such technique is straightforward to apply to recurrent neural networks by computing the normalization statistics separately at each time step. We explain on this chapter in details how the layer normalization works and we will show that adding it to our baseline model (Listen-Attend-And-Spell) indeed leads to faster convergence and better generalization.

Keywords: Layer Normalization, recurrent model optimization, Speech recognition

1. Introduction

Over the last two decades, recurrent neural networks have been widely used and they proved to be as performing as the existing state-of-the-art machine learning models in various tasks such as translation, speech-recognition, etc. But state-of-the-art recurrent neural networks often require many days of training. Although that some researchers suggested speeding the learning task by splitting either the data or the model on different machines during training. Such a solution is still not optimal and requires a lot of communication and complex software engineering. Instead, an orthogonal solution is to modify how the data is flowing between the layers during the feed-forward learning phase to make the learning easier. In this context, layer normalization [1] has been proposed by Geoffrey E. Hinton et al. where they demonstrated how estimating the normalization statistics from the summed inputs to the neurons within a hidden layer improves the training phase of recurrent neural networks and leads to a better and faster model. In the next sections, we will detail how the layer normalization is implemented within recurrent neural network in particular the Long-short-term-memory (LSTM) and after that we will explain our experiments and the results that we had when Layer Normalization is added to our baseline speech-recognition model (The Listen-attend-and-spell) [2].

2. Layer Normalization

Training a neural network is a difficult task where you have to take care of different parameters like the role of depth in the architecture, the nature of non-linear activation functions or the initialization of the weights. The combination of these factors favors the change in the data distribution while it flows through the network. This effect is called internal covariate shift [3] and lies at the core of problems such as vanishing/exploding gradients or dead neurons. Covariate shift can be reduced by fixing the mean and the variance of the summed inputs within each layer. We, thus, compute the layer normalization statistics as following:

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

where H is the number of the hidden units and a^l is the vector representation of the summed inputs to the neurons of the l^{th} layer. Within the layer, all the hidden units share the same normalization terms μ and σ.

2.1. Layer Normalization in recurrent neural network

In Recurrent Neural Network, the summed inputs a^t of a layer are computed as following:

$$a^t = W_{hh}h^{t-1} + W_{xh}x^t \tag{2}$$

where W_{hh} is the recurrent hidden to hidden weight and W_{xh} is the bottom up input to hidden weight matrix. Layer-Normalization re-centers and re-scales the input a^t as following:

$$h^t = f\left[\frac{g}{\sigma^t} \odot (a^t - \mu^t) + b\right] \tag{3}$$

μ and σ are computed as explained in equation 1, $\odot$ is the element-wise multiplication between two vectors. Both g (gain) and b (bias) are two trainable variables that have the same shape as h^t and serve respectively to shifting and scaling the normalized output to zero mean and unit variance.

2.2. Layer Normalization in Long-Short-Term Memory

We have seen previously how LN is applied to the Recurrent neural network and now we will specifically focus on how it is applied to the LSTM architecture [4]. Recall that inside LSTM the information is flowing through 4 type of gates: Input, forget, update and output.

$$\begin{pmatrix} f \\ i \\ g \\ o \end{pmatrix} = W_h h_{t-1} + W_x x_t + b \tag{4}$$

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(g_t) \tag{5}$$

$$h_t = \sigma(o_t) \odot c_{t-1} \odot tanh(c_t) \tag{6}$$

c_t and h_t are respectively the state and hidden cell.

Adding layer Normalization to the LSTM will change the basic equations of the information flow as follows:

$$\begin{pmatrix} f \\ i \\ g \\ o \end{pmatrix} = LN(W_h h_{t-1}; \alpha_1, \beta_1) + LN(W_x x_t; \alpha_2, \beta_2) + b \tag{7}$$

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(g_t) \tag{8}$$

$$h_t = \sigma(o_t) \odot tanh(LN(c_t; \alpha_3, \beta_3)) \tag{9}$$

where the multiplicative (α_i) and additive (β_i) are initialized respectively with ones and zeros vectors.

3. Related research questions

Recall that our baseline model, the listen-attend-and-spell, has two main components: the encoder (Listener), which is a Pyramidal deep bidirectional LSTM and the decoder (Speller), which is a recurrent neural network. Since both of them share the recurrent nature of network, we will be adding layer normalization to them under different scenarios and conditions. The aim of our experiments is to answer the following research questions:

- Does the Layer Normalization work as expected within our baseline model?
- Where to apply Layer normalization: to the encoder only? decoder only? or both?
- Does the layer normalization enhance the criteria of evaluation (word-error-rate/phoneme-error-rate) we care about?

4. Experiments

4.1. Experiment 1

First, we applied layer normalization only to the LSTM cells in our encoder and we kept everything else as it is in the baseline model.
We can see from the $Figure 1$ that adding layer normalization indeed makes our LAS model converge faster in term of number of steps.
Recall here that one step is by definition one gradient's update and in one step the model process $batch\_size$ examples of our training data.

$$Number\ of\ steps = \frac{All\_training\_data\_samples}{batch\_size} \tag{10}$$

We tested our method against two datasets: LibriSpeech [5] and TIMIT [6]. Adding layer normalization not only helped the model to converge faster but also to better generalize at test-time and hence reducing the phoneme error rate up to 2% for the TIMIT and the word error rate up to 2% for the Librispeech as we can see in $Table 1$ and $Table 2$.

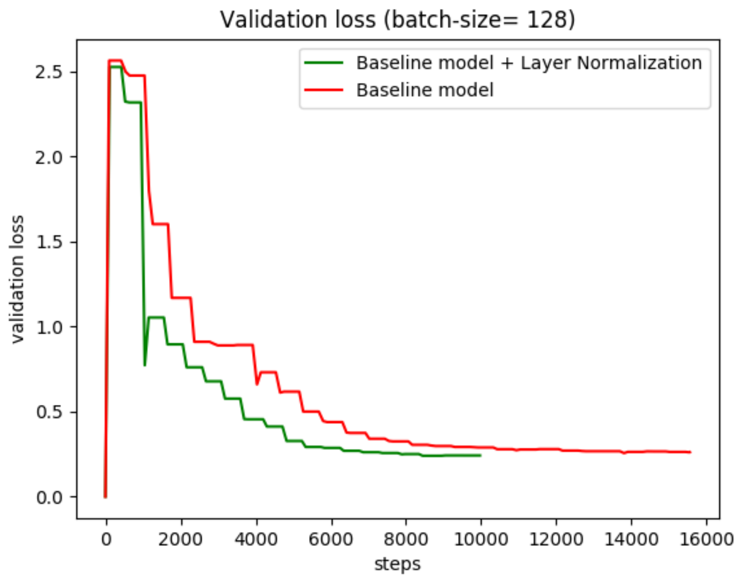


Figure 1: Validation loss for the TIMIT dataset.

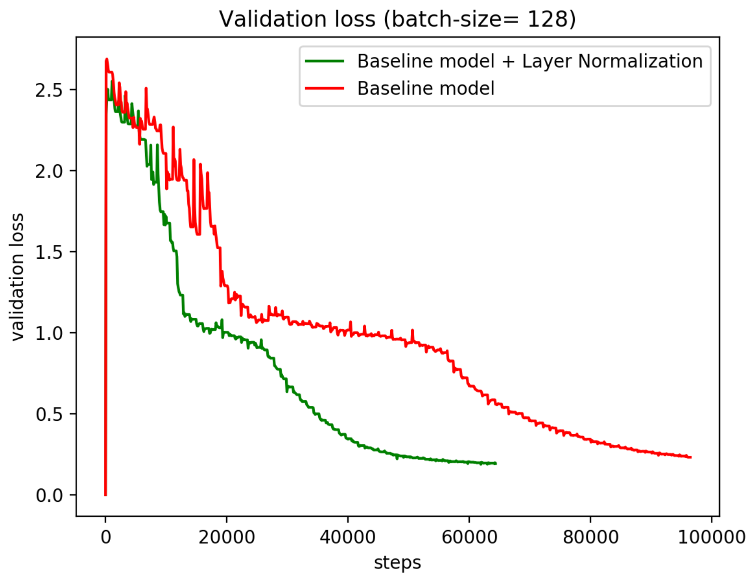


Figure 2: Validation loss for the LibriSpeech dataset.

Model	**Steps to converge**	**Phoneme error rate**
LAS With layer Normalization	9976	0.245
LAS Without layer Normalization	15947	0.268

Table 1: Performance evaluation for the TIMIT dataset

Model	**Steps to converge**	**Word error rate**
LAS With layer Normalization	64354	0.401
LAS Without layer Normalization	96785	0.426

Table 2: Performance evaluation for the LibriSpeech dataset

Recall that for speech-recognition tasks, there are two different metrics to evaluate the performance of the models: Word-error-rate and Phoneme-error-rate and by definition:

$$Word\ Error\ Rate = \frac{S+D+I}{N} = \frac{S+D+I}{S+D+C} \tag{11}$$

Where

- S is the number of substitutions
- D is the number of deletions
- I is the number of insertions
- C is the number of correct words
- N is the number of words in the reference (N=S+D+C)

The Phoneme error rate (PER) is applying the same operation on phonemes instead of words.

4.2. Experiment 2

In this experiment, we want to investigate where does the layer normalization give its best performance. First, we applied layer normalization to both the LSTM cells of our encoder (Listener) and decoder (Speller).
As a proof of concept, we restricted our experiments this time to the TIMIT dataset due to the high time and GPUs consumption required for training the Librispeech dataset.

We can see that the LAS model converges faster when the layer-normalization is only applied to the encoder (Figure3) and generalizes slightly better at test time (Table3).
Second, we applied layer normalization only to the decoder (Speller) and we kept the encoder as it is in the baseline model. Under this configuration, our model performed poorly on the recognition task: the validation loss converges earlier to higher value and the model gave higher phoneme-error-rate on the test-set.

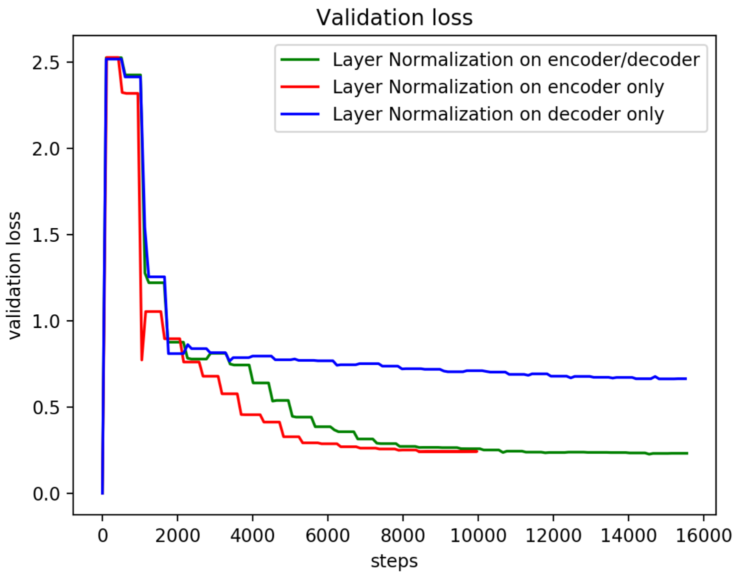


Figure 3: Validation loss.

Model	**Steps to converge**	**Word error rate**
Layer Normalization only on encoder	9976	0.245
Layer Normalization on both encoder/decoder	15552	0.253
Layer Normalization only on decoder	15600	0.666

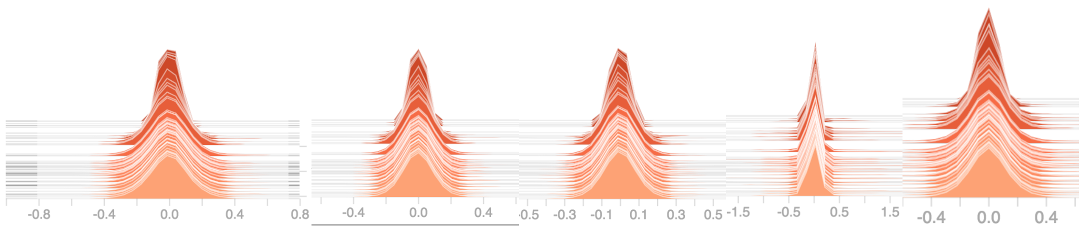
Table 3: Performance evaluation for the TIMIT dataset

4.3. Visualizations

In previous subsections, we showed that layer normalization worked best when it is applied to the encoder. To understand and inspect more what happens to our model during training when the layer normalization is applied, we dive now into some visualizations of two important key features:

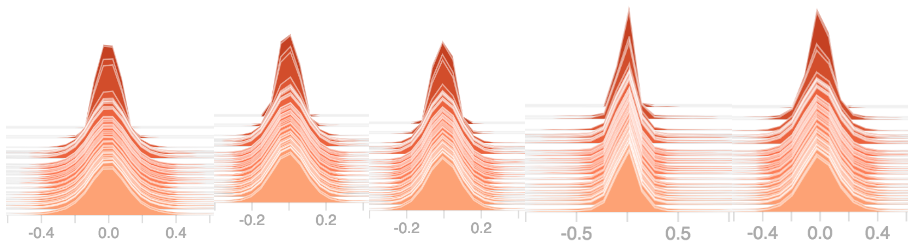
- The data-distribution of our trainable variables.
- Attention visualization.

4.3.1. Data distribution's visualization



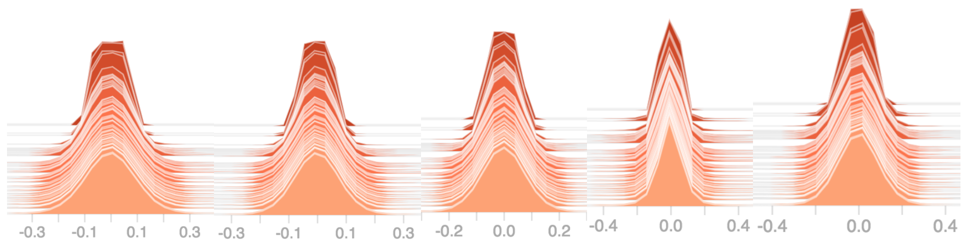
(a) Encoder1 (b) Encoder2 (c) Encoder3 (d) Decoder1 (e) Decoder2

Figure 4: Baseline model



(a) Encoder1 (b) Encoder2 (c) Encoder3 (d) Decoder1 (e) Decoder2

Figure 5: Layer Normalization on encoder only



(a) Encoder1 (b) Encoder2 (c) Encoder3 (d) Decoder1 (e) Decoder2

Figure 6: Layer Normalization only on decoder

Previously in *section*2, we have explained that accordingly to the original paper of layer normalization, the authors explained that their trick works because it remedies the Internal Covariate Shift (ICS).
ICS refers to the change of distribution of inputs to the hidden layers as the parameters of the model change. At this point, visualizing the data's distribution in *Figure*4, 5 and 6, shows us that this reasoning might not be correct and layer normalization has nothing to do with reducing the covariate shift.
This raises an important question for us:

- Is the effectiveness of Layer normalization indeed related to the ICS?

In this context, an empirical work have been done by [7] where they provided both empirical evidence and theoretical proof that Batch Normalization helps optimization by making optimization landscape smoother rather than reducing ICS.
One manifestation of this impact is improvement in the *Lipschitzness* of the loss function. That is, the loss changes at a smaller rate. Similar observations we have when we plotted the evolution of the *loss* function with and without layer normalization.

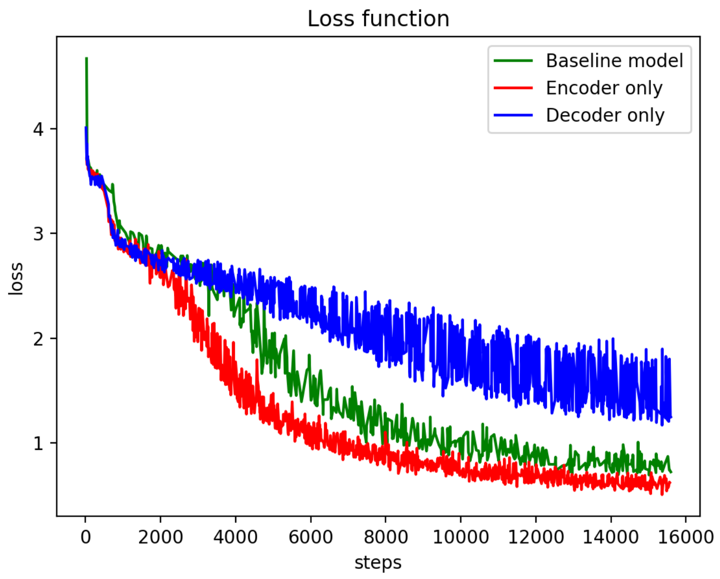


Figure 7: Evolution of *loss* function with and without layer normalization.

4.3.2. Attention visualization

The content-based attention mechanism creates an explicit alignment between the characters and the input audio signal.
To evaluate the performance of the model under the different scenarios of layer normalization, we visualize the attention mechanism by recording the attention distribution on the acoustic sequence at every character output time-step.

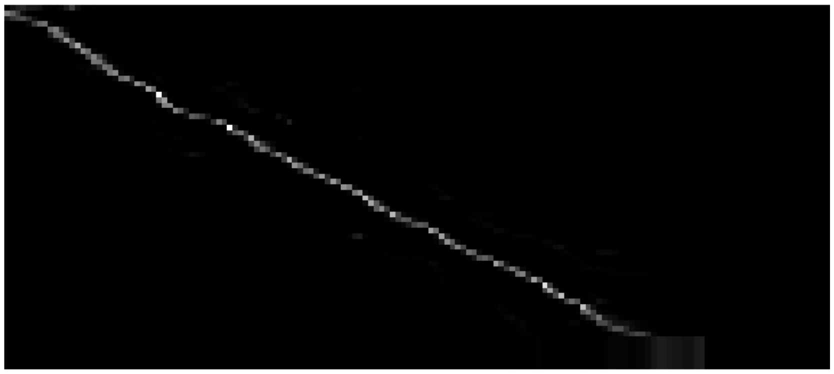


Figure 8: Alignments between character outputs and audio signal produced by the LAS model.

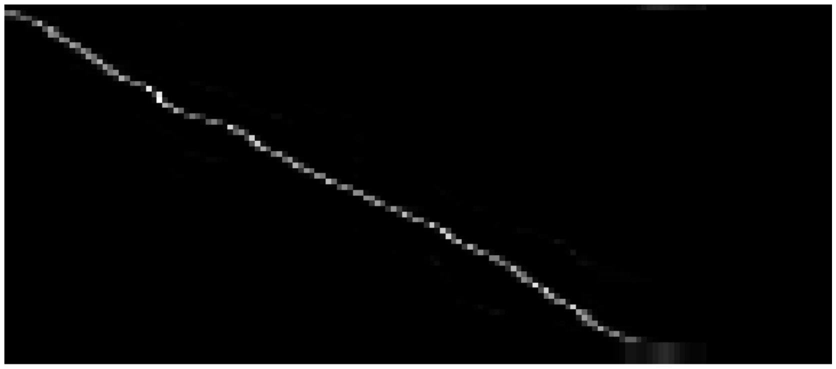


Figure 9: Alignments between character outputs and audio signal produced by the LAS model with layer-normalization on encoder only.

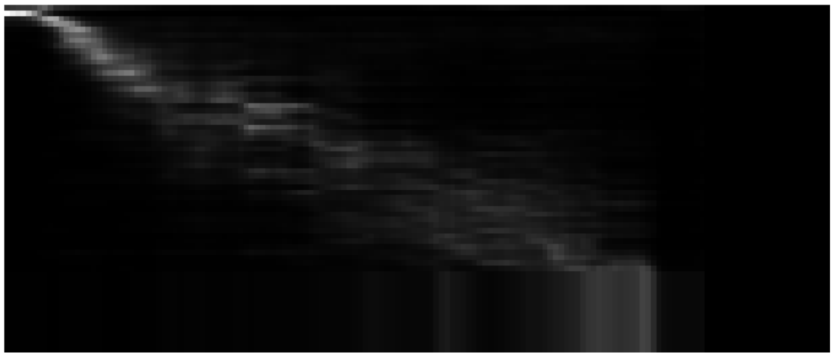


Figure 10: Alignments between character outputs and audio signal produced by the LAS model with layer-normalization on decoder only.

We can see clearly, that the attention-mechanisms has failed when the layer normalization is added to the decoder ($Figure 10$).
In the other hand, both baseline model and (baseline-model + layer normalization on encoder) have almost the same alignment between the input audio and the decoded characters except that (LAS + layer normalization on encoder) is paying more attention to some regions of the input audio features (the more intense pixels).

5. Conclusion

We have in this chapter explained the layer-normalization technique and demonstrated that it greatly optimizes our baseline-model "The Listen-attend-and-Spell".
In our experiments we have showed that adding layer normalization to the encoder only gave the best performance in terms of fast convergence and better generalization at test-time. We added also another explication of why layer normalization works within our model and explained that it is not only due to reducing the ICS but also to smoothing the landscape of the loss function.

6. Bibliography

[1] J. L. Ba, J. R. Kiros, G. E. Hinton, Layer normalization, arXiv preprint arXiv:1607.06450 (2016).

[2] W. Chan, N. Jaitly, Q. Le, O. Vinyals, Listen, attend and spell: A neural network for large vocabulary conversational speech recognition, in: Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on, IEEE, pp. 4960–4964.

[3] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167 (2015).

[4] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.

[5] V. Panayotov, G. Chen, D. Povey, S. Khudanpur, Librispeech: an asr corpus based on public domain audio books, in: Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on, IEEE, pp. 5206–5210.

[6] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1, NASA STI/Recon technical report n 93 (1993).

[7] S. Santurkar, D. Tsipras, A. Ilyas, A. Madry, How does batch normalization help optimization?, 2018.