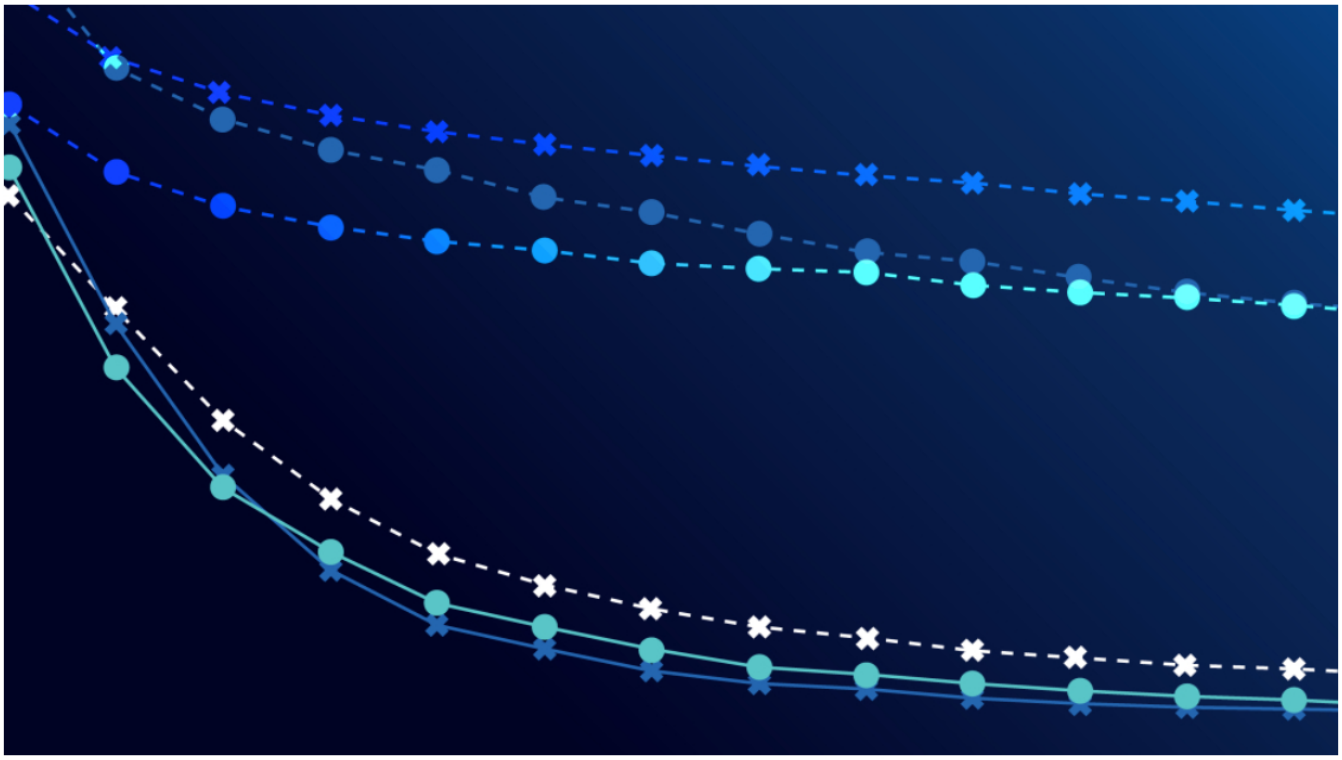
Tutorials | Natural Language Processing

Tutorial #17: Transformers III Training

Aug. 06, 2021 P. Xu, S. Prince



Tricks for training transformers

Why are these tricks required?

Learning rate warm-up

Residual connections, layer normalization and Adam

Gradient shrinkage effect

Unbalanced dependencies and amplified output perturbations

Aggravating effect of Adam

In part I of this tutorial we introduced the self-attention mechanism and the transformer architecture. In part II, we discussed position encoding and how to extend the transformer to longer sequence lengths. We also discussed connections between the transformer and other machine learning models.

In this final part, we discuss challenges with transformer training dynamics and introduce some of the tricks that practitioners use to get transformers to converge. This discussion will be suitable for researchers who already understand the transformer architecture, and who are interested in training transformers and similar models from scratch.

Tricks for training transformers

Despite their broad applications, transformers are surprisingly difficult to train from scratch. One of the contributions of the original transformer paper was to use four tricks that collectively allow stable training:

1. Residual connections: Each transformer layer takes the $I \times D$ data matrix X where I is the number of inputs and D the dimensionality of those inputs and returns an object of the same size. It performs the following operations:

$$X \leftarrow X + \mathrm{MhSa}[X]$$
$$X \leftarrow \mathrm{Layernorm}[X]$$

$$\begin{aligned} x_i &\leftarrow x_i + \mathrm{mlp}[x_i] \qquad \forall i \in \{1 \ldots I\} \\ X &\leftarrow \mathrm{Layernorm}[X], \end{aligned} \tag{1}$$

which include two residual connections around the multi-head self-attention $\mathrm{Mhsa}[\cdot]$ and multi-layer perceptron $\mathrm{mlp}[\cdot]$ components (figure 1).

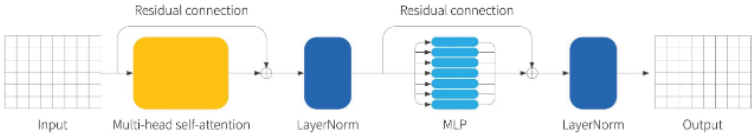


Figure 1. The transformer layer. The input consists of a $I \times D$ matrix containing the D dimensional embeddings for each of the I input tokens. The output is a matrix of the same size. The transformer layer consists of a series of operations. First, there is a multi-head attention block which allows the embeddings to interact with one another. This is in a residual network, so the original inputs are added back to the output. Second, a layer norm operation is applied. Third, there is a second residual network where the same two-layer fully-connected neural network is applied to each representation separately. Finally, layer-norm is applied again.

2. Layer normalization: After each residual connection, a layer normalization procedure is applied:

$$\mathrm{Layernorm}[X] = \gamma \cdot \frac{X - \mu}{\sigma} + \beta, \tag{2}$$

where μ and σ are the mean and standard deviation of the elements of X (but are separate for each member of the batch), and γ and β are learned parameters.

3. Learning rate warm-up: The learning rate is increased linearly from 0 to R over first T_R time steps so that:

$$\mathrm{lr}[t] = R \cdot \frac{t}{T_R}. \tag{3}$$

4. Adaptive optimizers: Transformers need to be trained with adaptive optimizers like *Adam*, which recursively estimates the momentum and the learning rate separately for each parameter at each time-step. In practice, relatively large batch sizes of $> 1,000$ are usually employed.

Removing any of these tricks makes training unstable and often leads to complete training failures. However, they have been employed without a full understanding of why they are required.

As transformers are applied more widely, it is increasingly important that we have a better understanding of transformer training. To this end, a number of recent papers have been devoted to demystifying this topic and exploring better training methods. In the rest of this blog post, we will connect these separate efforts to form a comprehensive overview of this topic.

Why are these tricks required?

In this section we will review some tricks and see that there are complex dependencies between them: Some tricks cause problems, which are in turn resolved by others. We will see that there are complex dependencies between them, so that some of the tricks cause problems, which are in turn resolved by others. In the subsequent section we will discuss improvements to the training process that follow from this understanding.

Learning rate warm-up

Learning rate warm-up (in which the learning rate is gradually increased during the early stages of training) is particularly puzzling. This is not required for most deep learning architectures. However, training fails for transformers if we just start with a

typical learning rate. If we start with a very small learning rate, then the training is stable, but then it takes an impractically long time.

Xiong *et* al., 2020 explored this phenomenon by conducting experiments on a machine translation task with different optimizers and learning rate schedules. Their results (figure 2) show that learning rate warm-up is essential for both Adam and SGD, and that the training process is sensitive to the warm-up steps.

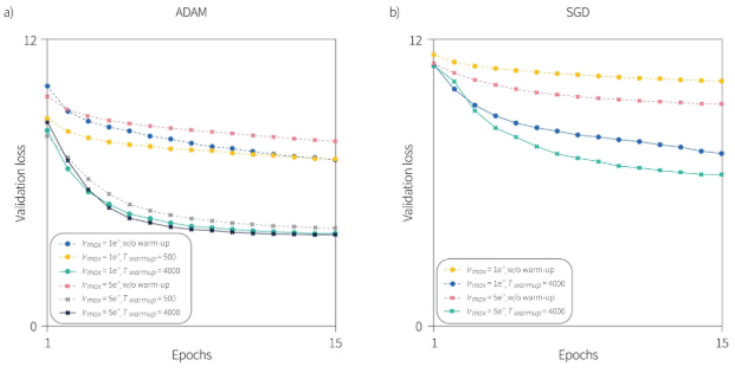


Figure 2. Importance of learning rate warm-up for transformers. a) Validation loss for Adam optimizer for IWSLT14 De-En task. Performance is poor (curves are higher) without warm-up or if learning rate increases too quickly. Performance is much better if learning rate is gradually increased. b) Validation loss for the SGD optimizer for IWSLT14 De-En task. A similar pattern is observed, but the overall results are much worse than for Adam. Adapted from Xiong et al., 2020.

Although learning rate warm-up works, it has some obvious disadvantages. It introduces an extra hyper-parameter (the number of warm-up steps) and it initializes the learning rate to zero which slows the training down. Hence, it's important that we understand why it is necessary.

To help answer this question, Huang *et* al., 2020 visualized the gradient of the loss L with respect to the input embeddings $\mathbf{X}$, and the size of the Adam updates during the first 100 steps of training (figure 3). They found that without warm-up, the gradients vanish very quickly, and the Adam updates also rapidly become much smaller. Diminishing gradients at lower layers in the transformer model without warm-up have also been observed by Liu *et* al., 2020.

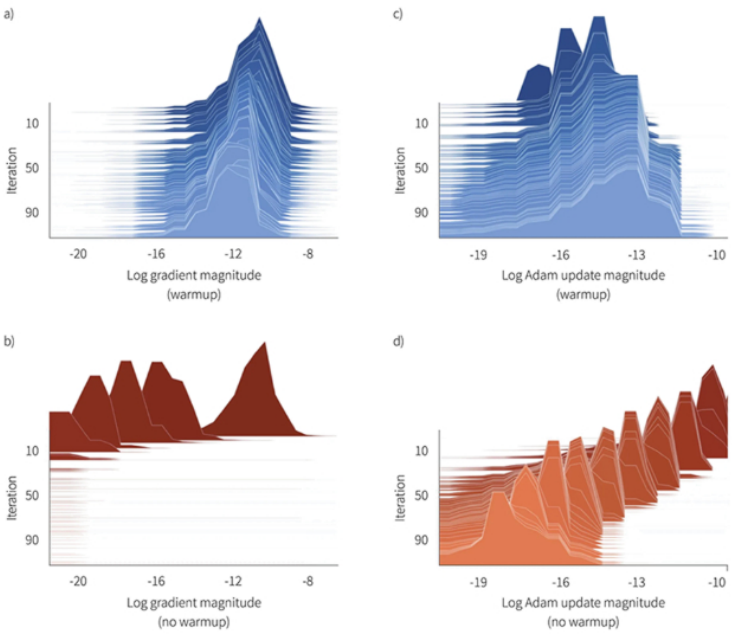


Figure 3. Removing learning rate warm-up makes gradients vanish. a) Histogram of magnitude of gradient of loss function with respect to inputs during first 100 steps of training. When we use learning rate warm-up, these gradients remain constant. b) Without warm-up the gradients vanish after just a few steps. c-d) We observe a similar pattern of behaviour in the histograms of Adam update sizes with and without warm-up. Adapted from Huang et al., 2020.

Residual connections, layer normalization and Adam

To understand why learning rate warm-up is required, and why the gradients vanish without it, we will first need to understand the reasons for, and the consequences of using residual connections, layer normalization, and Adam.

Residual networks were developed in computer vision; they make networks easier to optimize and allow deeper networks to be trained. In computer vision, the additive residual connections are usually placed around convolutional layers and combined with batch normalization. In the transformer, they are placed around the self-attention and feed-forward networks and combined with layer normalization (figure 1). From this perspective, the transformer architecture could be considered a "deep residual self-attention network".

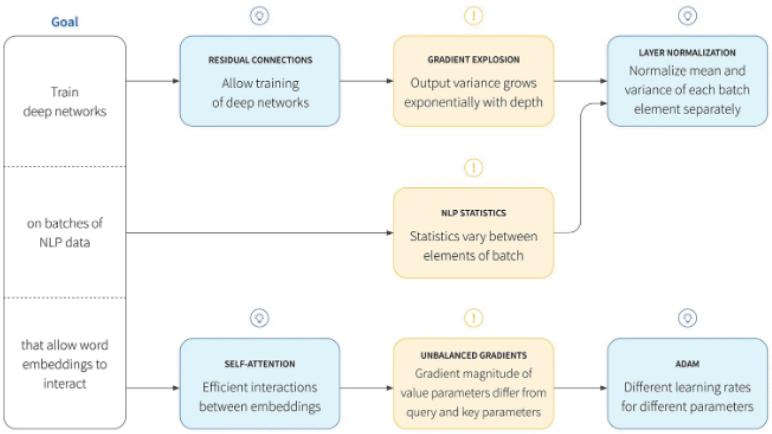


Figure 4. Residual connections, layer normalization, and Adam. We start from the premise that we would like to train deep networks on NLP data that allow the embeddings to interact. To train deep networks, we need residual connections, and these cause gradient explosion, which must be resolved by normalization. Transformers use layer normalization (which normalizes each member of the batch independently) because the batch statistics of language data exhibit large fluctuations. To allow the embeddings to interact efficiently, we use self-attention, but the gradients of the parameters in the self-attention module are unbalanced (i.e., some are much larger than others). This necessitates the use of Adam, which uses a separate learning rate for each parameter.

Zhang *et* al., 2019 show that the output variance of residual networks grows exponentially with depth. Hence, normalization is used to prevent *gradient explosion* for deep residual networks. Layer normalization is used in the transformer because the statistics of language data exhibit large fluctuations across the batch dimension, and this leads to instability in batch normalization.

Transformers also differ from convolutional networks in that stochastic gradient descent does not work well for training (figure 2) and adaptive optimizers like Adam are required. Liu *et* al., 2020 observed that differentiating through the self-attention mechanism creates *unbalanced gradients*. In particular, the gradients for the query Φ_q and key Φ_k parameters were much smaller than those for the value parameters Φ_v, and so the former parameters change much more slowly. This is a direct consequence of the mathematical expression for self-attention. The Adam optimizer fixes this problem by essentially having different learning rates for each parameter.

To conclude, we've seen that residual connections are needed to allow us to train deep networks. These cause gradient explosion, which is resolved by using layer normalization. The self-attention computation causes unbalanced gradients, which necessitates the use of Adam (figure 4). In the next section, we'll see that layer normalization and Adam themselves cause more problems, which ultimately result in the need for learning rate warm-up.

Gradient shrinkage effect

Xiong *et* al., 2020 found that the magnitude of the gradients through layer normalization is inversely proportional to magnitude of the input. Specifically, the gradient has the following property:

$$\left\| \frac{\partial \text{Layernorm}[X]}{\partial X} \right\| = O\left(\frac{\sqrt{D}}{\|X\|} \right), \tag{4}$$

where X is the input to layer normalization and D is the embedding dimension. If the input norm $\|X\|$ is larger than $\sqrt{D}$ then back-propagating through layer normalization reduces the gradient magnitude in lower layers. As this effect compounds through multiple layers, it causes the gradient to vanish at lower layers for deep models. We will term this the *gradient shrinkage effect*.

Unbalanced dependencies and amplified output perturbations

Layer normalization also causes *unbalanced dependencies* between the two branches of the residual connection around the self-attention module. In other words, the output of $\text{LayerNorm}[X + \text{Sa}[X]]$ depends much more on the self-attention computation $\text{Sa}[X]$ than the skip connection X. This means that the outputs depend much more on later layers than earlier layers. Liu *et* al., 2019 show that this happens empirically in practice.

Moreover, they show that this leads to *amplified output perturbations*; small changes to the network parameters cause large output fluctuations. More precisely, they proved that for a transformer network $T_N[X, \theta]$ with parameters θ, the output variance scales with the number of layers N when we randomly perturb the parameters to $\theta^* = \theta + \delta$:

$$\text{Var}[T_N[X;\theta] - T_N[X;\theta^*]] = O(N), \tag{5}$$

They also show that this happens empirically with both random parameter changes and Adam updates (figure 5). The result is that the output changes more and more when we update the parameters, which destabilizes transformer training.

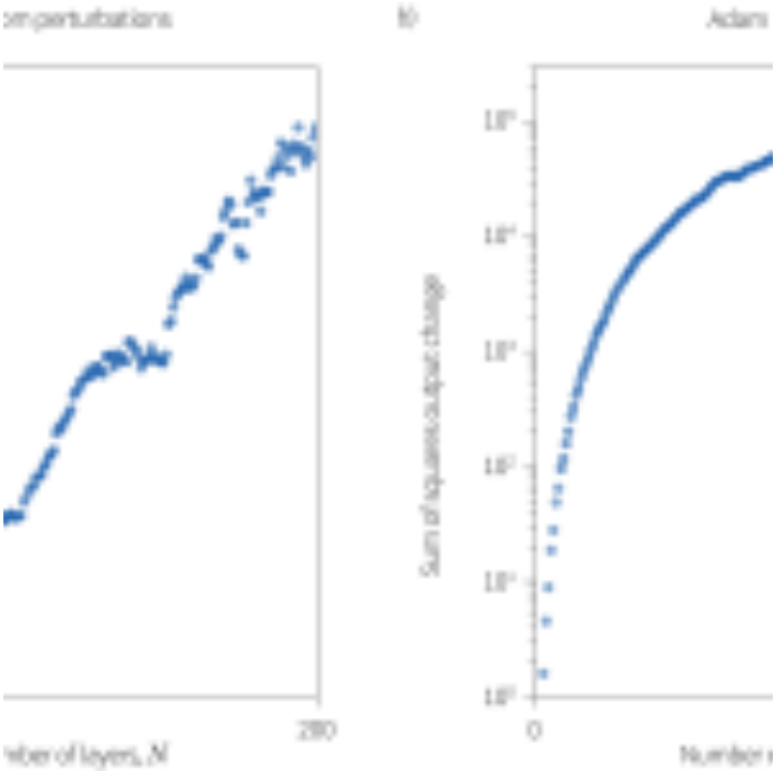


Figure 5. Amplified output perturbations. Consider a transformer network $T_N[X, \theta]$ with N layers and parameters θ. a) A random perturbation is added to the parameters θ to create new parameters θ^*. The sum of squares of the change in output $|T_N[X;\theta] - T_N[X;\theta^*]|_2^2$ increases linearly with the number of layers. b) The same effect is observed when we apply Adam updates (note log scale). Adapted from Liu et al., 2019.

Aggravating effect of Adam

Furthermore, using adaptive optimizers like Adam aggravates both the gradient shrinkage effect and the amplified output perturbations. Liu *et* al., 2019 show that the variance of the Adam updates is unbounded at the start of training, and these updates are also known to exhibit high variance in the early stages of training.

This can lead to problematic large updates early on which can make the input norm $\|X\|$ to each layer increase as we move through the network and thus increased gradient shrinkage as predicted by equation 2.3. Moreover, the output fluctuation which is already amplified by the network structure will be even greater for these large parameter updates.

Why learning rate warm-up helps

To summarize, residual connections are required in the transformer architecture for the ease of optimization, which further requires layer normalization to avoid gradient explosion and adaptive optimizers like Adam to address unbalanced gradients in the self-attention blocks. On the flip side, the use of layer normalization causes the gradients to shrink in the early layers and also amplifies the output perturbations. Moreover, the instability of Adam in the early stages of training exacerbates both of these effects (figure 6).

Figure 6. Consequences of using layer normalization and Adam. Layer normalization in transformer networks causes unbalanced dependencies between the two branches of the residual blocks. This in turn leads to amplified output perturbations; the output becomes increasingly sensitive to changes in the parameters as the depth of the network grows. Layer normalization also means that the gradient decreases proportionally to the output norm of each transformer block. This gradient shrinkage aggregates as we back-propagate through the network and means the gradients become very small for the early layers. In the early stages of training the Adam updates are unstable and can be large; this in turn causes the output of the later layers to fluctuate more causing large output norms which in turn causes more gradient shrinkage. The solution is to use learning rate warm-up which effectively damps the Adam updates during the unstable early phase.

This is where learning rate warm-up comes in: it effectively stabilizes the Adam updates during the early stages of training by making the parameter changes much smaller. Consequently, Adam no longer aggravates gradient shrinkage and amplification of output perturbations and training becomes relatively stable.

Better methods for training transformers

In the previous section, we argued that the transformer architecture, and the statistics of language data require us to use layer normalization and train with adaptive optimizers like Adam. These choices in turn cause other problems that are resolved by using learning rate warm-up. In this section, we consider alternative methods for training deep transformers that don't require learning rate warm-up.

We'll consider three approaches that respectively remove the normalization from the network, attempt to re-balance the dependency on the two paths of the residual normalization.

Recall that the normalization mechanism is introduced to prevent gradients exploding in

deep residual networks. It follows that if we can stabilize the gradient updates $\Delta\theta$, then we can remove layer normalization. Zhang *et* al., 2019 demonstrated that the gradient updates $\Delta\boldsymbol{\theta}$ can be bounded when using the SGD optimizer to train residual MLP or convolution blocks by appropriately initializing the weights. Based on this work, Huang *et* al., 2020 derived an analogous initialization scheme for residual self-attention blocks.

Although the theoretical derivations are for SGD updates, these results hold well for adaptive optimizers like Adam in practice. Furthermore, it follows from the Taylor expansion:

$$\Delta T[X,\theta] \approx \frac{\partial T[X,\theta]}{\partial \boldsymbol{\theta}} \Delta \boldsymbol{\theta}, \tag{6}$$

that the output fluctuation Δf is also bounded by bounding the gradient updates $\Delta\theta$. As a result, both the gradient vanishing and the amplified output perturbations are resolved with stable gradient updates.

The proposed initialization scheme is known as *T-Fixup* and is easy to implement. Consider a multi-head self-attention block where the h^{th} head computes

$$Sa_h[X] = Softmax\left[\frac{(X\Phi_{qh})(X\Phi_{kh})^T}{\sqrt{d_q}}\right] X\Phi_{vh}. \tag{7}$$

where X is the input data matrix containing word embeddings in its rows and Φ_{qh}, Φ_{kh} and Φ_{vh} are the weight parameters for the queries, keys, and values respectively. The outputs of these self-attention mechanisms are concatenated and another linear transform Φ_c is applied to combine them:

$$MhSa[X] = [Sa_1[X]\, Sa_2[X] \ldots Sa_H[X]]\Phi_c. \tag{8}$$

The T-Fixup scheme for encoder decoder attention is then as follows:

1. Apply Xavier initialization for all parameters excluding input embeddings. Use Gaussian initialization $Norm_\theta[0, D^{-\frac{1}{2}}]$ for input embeddings where D is the embedding dimension.

2. Scale Φ_{vh} and Φ_c in each encoder attention block and weight matrices in each encoder MLP block by $0.67N_e^{-\frac{1}{4}}$ where N_e is the number of transformer blocks (i.e, self-attention + MLP) in the encoder. Scale the input embeddings to the encoder by $(9N_e)^{-\frac{1}{4}}$

3. Scale parameters Φ_{vh} and Φ_c in each decoder attention block, weight matrices in each decoder MLP block and input embeddings in the decoder by $(9N_d)^{-\frac{1}{4}}$ where N_d is the number of transformer blocks in the decoder.

In practice, *T-Fixup* is able to train significantly deeper transformer models with improved performance on the task of machine translation. For the detailed derivation of this method, we refer the readers to the original paper.

Balancing the residual dependencies

An alternative approach is to balance the residual dependencies, which in turn will limit the output perturbations $\Delta T[X]$. Equation 6 shows that controlling the magnitude of the output fluctuation $\Delta T[X]$ also bounds the magnitude of the gradient updates $\Delta\theta$, which in turn mitigates the problem of gradient vanishing. Here we'll consider three possible approaches.

Pre-LN transformers: One simple solution is to change the location of layer normalization inside the transformer layer so that it occurs inside the residual blocks and before the self-attention or MLP (figure 7). This is known as the *pre-LN*

transformer. This simple change can help control the gradient magnitude and balance the residual dependencies.

Pre-LN transformer models can be trained without learning rate warm-up. However, they also lead to inferior empirical performance. It has been speculated that this is because now the models are restricted not to depend too much on the contents of their residual layers Liu *et* al., 2020.

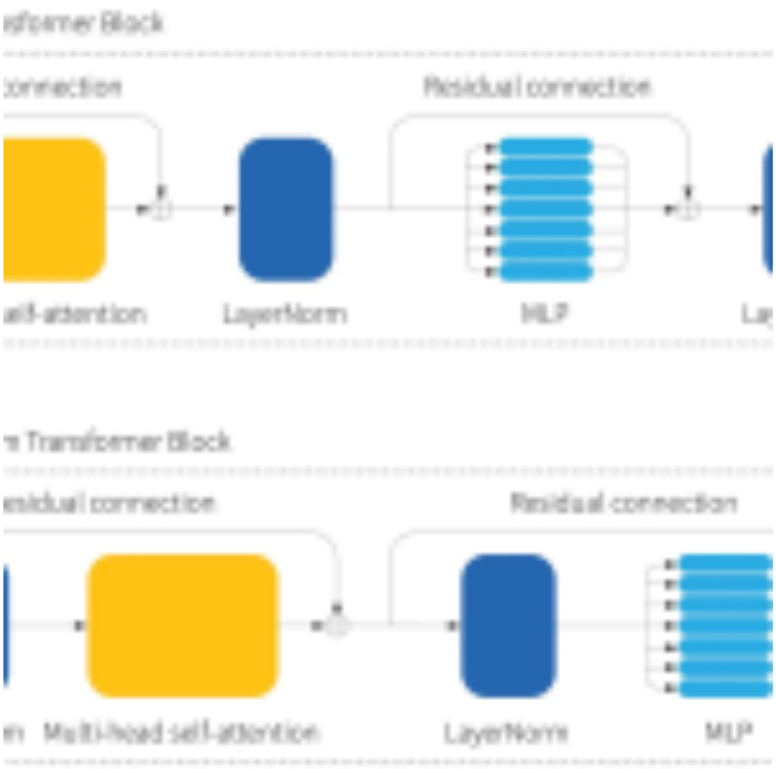


Figure 7. Moving the layer normalization to balance residual dependencies. (a) Original transformer layer (b) The pre-LN transformer layer moves the layer norm operation inside the residual block and before the self-attention / MLP blocks.

Admin: To bridge this performance gap, *adaptive model initialization* or *Admin* aims to bound the output fluctuation $\Delta \mathrm{T}[\mathrm{X}]$ by controlling the residual dependencies while retaining the original architecture.

Admin adds a new parameter $1 \times \mathrm{D}$ parameter vector ψ to each residual block. The self-attention block is then constructed as $\mathrm{LayerNorm}[\mathrm{X} \odot \Psi + \mathrm{MhSa}[\mathrm{X}]]$ where $\odot$is the element-wise product and Ψ is an $\mathrm{I} \times \mathrm{D}$ matrix where each row is a copy of ψ. The residual connection around the parallel MLP layer is treated in the same way (figure 8a).

The new parameters at the n^{th} layer are initialized to be the output standard deviation at that layer before this intervention. This can be estimated by setting all elements of ψ to one and forward propagating on a few training instances.

ReZero: In a similar vein, *ReZero* removes the layer normalization and introduces a single trainable parameter α per residual layer so that the self-attention block residual layer becomes, $\mathrm{X} + \alpha \mathrm{MhSa}[\mathrm{X}]$, where α is initialized to zero (figure 8b). The result of this is that the entire network is initialized just to compute the identity function, and the contributions of the self-attention and MLP layers are gradually and adaptively introduced.

Empirically, both Admin and ReZero work well for training deeper transformer models with better generalization performance, which demonstrates the effectiveness of balancing the residual dependencies.

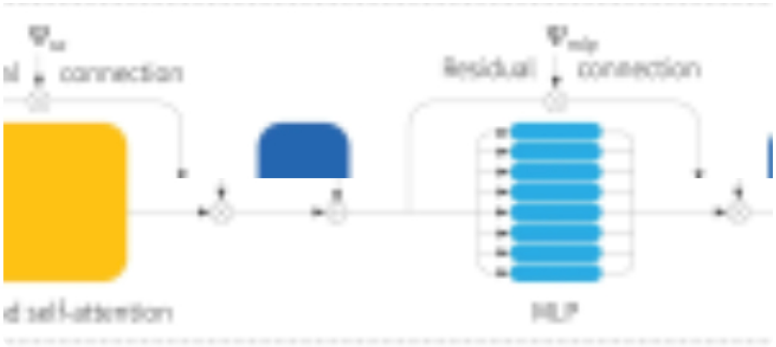


Figure 8. Balancing dependencies by adding extra parameters in the residual layers. a) The Admin method pointwise multiplies the input $\mathbf{X}$ being passed through the residual connection around the self-attention layer by a matrix Ψ_{sa} where each row of this matrix contains the same vector ψ_{sa}. With suitable initialization, this balances the contributions of the two paths. The MLP residual block is treated similarly, but with distinct parameters Φ_{mlp}. b) The ReZero method removes the LayerNorm altogether and multiplies the output of the self-attention layer and the output of the MLP layer to by a shared learned parameter α. The parameter α is initialized to zero, so this method initializes the entire network to the compute the identity and allows the outputs of the two branches of the residual blocks to be adaptively balanced.

Reducing the optimizer variance

We noted before that the high variance of learning rates in the Adam optimizer at the early stages of training exacerbates the problems of amplified output perturbations and gradient vanishing. Liu *et* al., (2019) argue that this is due to the lack of samples in the early stages of learning. They base their argument on an experiment in which they do not change the model parameters or momentum term of Adam for the first 2000 learning steps, but only adapt the learning rate. After this, warm-up is no longer required.

Based on these observations, they propose *Rectified Adam* or *RAdam* which gradually changes the momentum term over time in a way that helps avoid high variance. One way to think of this is that we have effectively incorporated learning rate warm-up into the Adam algorithm, but in a principled way.

How to train deeper transformers on small datasets

In the previous sections, we have seen that great progress has been made towards understanding transformer training. Several solutions have been proposed that allow the training of significantly deeper transformer models with improved empirical performance.

However, they have only been applied to tasks with sufficient training data such as machine translation and language modelling. This is possibly due to the commonly-held belief that training deep transformers from scratch requires large datasets. For small datasets, it is typical just to add shallow and simple additional layers (*e.g.*, a classifier head) to pre-trained models and then fine-tune.

So, what prevents practitioners from training deep transformers on small datasets? It turns out that the final missing piece of the puzzle is the batch size. For small datasets, it's necessary to leverage large pre-trained models and then fine-tune. However, the size of these models limits the batch size and when the batch size is small, the variance of the updates is even larger, which makes training even harder. Even if we could use a larger batch size, it usually results in poorer generalization, especially on small datasets.

In short, small datasets require pre-trained models and small batch sizes to perform well, but these two requirements make training additional transformer layers challenging. To resolve the high variance of training updates in small batch sizes, the three ideas from the previous section can all be applied. However, these approaches all assume that the inputs to the transformers are randomly initialized embeddings, but this is not true if we are adding yet-to-be-trained transformers on top of pre-trained models (figure 9).

Figure 9. Architecture on which DT-Fixup can be applied. The pre- and post-transformer modules can be any architectures that can be stably trained with Adam, including MLP, RNN, CNN, or a pre-trained transformer model which can be stably fine-tuned with a small learning rate.

DT-Fixup is a data-dependent initialization strategy developed by RBC Borealis. It adapts the T-Fixup method for this type of mixed setting. DT-Fixup allows significantly deeper transformers to be trained with small datasets for challenging tasks such as Text-to-SQL semantic parsing and logical reading comprehension. This demonstrates that training deep transformers with small datasets is feasible with the correct optimization procedure.

Conclusion

In the first two parts of this blog, we introduced the transformer, and discussed extensions and relations to other models. In this final part, we have discussed the

Founded by the
Royal Bank of Canada.

Research
AI Research
Open Source

Applications
Lumina
ATOM

Community
Who we are
RESPECT AI

Careers
Join Us
ML Research Internships