Member-only story

A Quick Guide on Normalization for Your NLP Model

Accelerate your model convergence and stabilize the training process with normalization

Thao Vu · Following

Published in Towards Data Science · 7 min read · Sep 14

Photo by Mattia Bericchia on Unsplash

Introduction

Efficiently training deep learning models is challenging. The problem becomes more difficult with the recent growth of NLP models' size and architecture complexity. To handle billions of parameters, more optimizations are proposed for faster convergence and stable training. One of the most remarkable techniques is normalization.

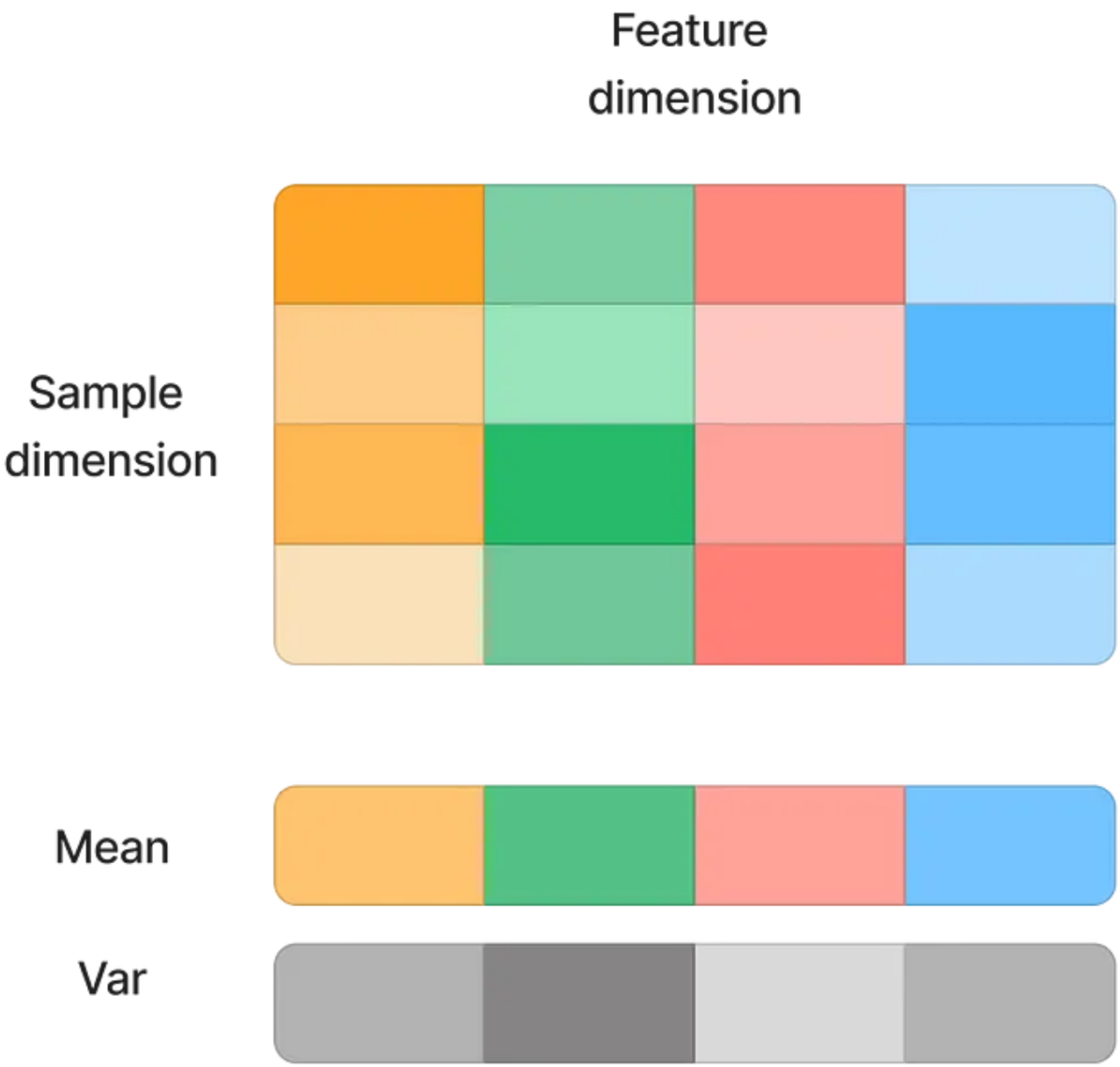
In this article, we will learn about some normalization techniques, how they work, and how they can be used for NLP deep models.

Why not BatchNorm?

BatchNorm [2] is an early normalization technique proposed to solve internal covariate shifts.

To explain in simple terms, an internal covariate shift occurs when there is a change in the layer's input data distribution. When the neural networks are forced to fit different data distributions, the gradient update changes dramatically between batches. Therefore, the models take longer to adjust, learn the correct weights and converge. The problem gets worse as the model size grows.

Initial solutions include using a small learning rate (so the impact of data distribution shifting is minor) and careful weight initialization. BatchNorm solved the problem effectively by normalizing the input on the feature dimension.



Batch Norm (Image by the author)

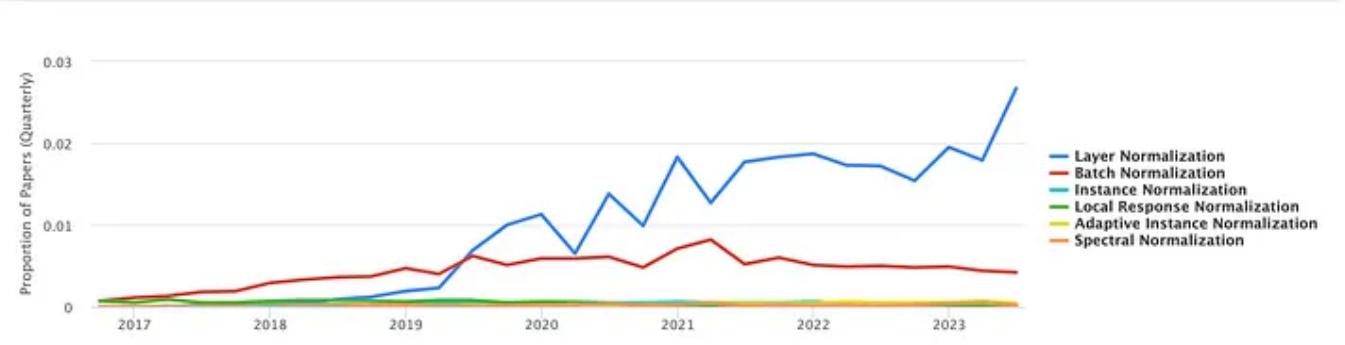
The technique helps speed up the convergence significantly and allows a higher learning rate as the model becomes less sensitive to outliers. However, it still has some drawbacks:

- **Small batch size:** BatchNorm relies on batch data to compute the feature's mean and standard deviation. When the batch size is small, the

mean and variance can no longer represent the population. Therefore, online learning is impossible with BatchNorm.

- **Sequence input:** In BatchNorm, each input sample's normalization depends on other samples from the same batch. This does not work so well with sequence data. For example, we have 2 training samples with different lengths *(a1, a2,.., a10)* and *(b1,b2,...,b20)*. Is it okay if token *b11* is normalized along with a padding token *a11*? At the inference step, if we have a sequence of length 30 *(c1, c2,.., c30)*, how do we get the mean and variance to normalize token *c21*? This is the crucial reason BatchNorm is not suitable for NLP tasks.
- **Parallelization**: It's difficult to parallelize batch-normalized models. Since there is dependence between elements (mean and variance), we need to synchronize them between devices. NLP models, such as Transformers, suffer from this setting due to their large-scale setup.

This is where LayerNorm [1] came in. Proposed in 2016, LayerNorm has dethroned BatchNorm and steadily become the most popular normalization technique among the research community.

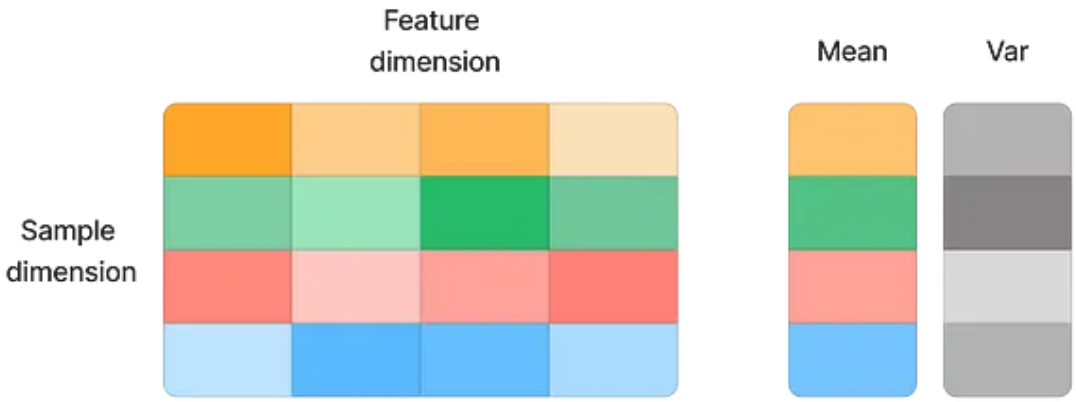


Normalization Technique Usage (Source)

So what is Layer Norm, and why it's so effective?

Why LayerNorm works so well?

In contrast to BatchNorm, LayerNorm normalizes the inputs on each data sample dimension. So, in a training batch of n samples ***(x1, x2,.., xn)***, the normalization is done on each ***xi*** independently.



Layer Norm (Image by the author)

LayerNorm has been widely used in many state-of-the-art language models, such as BERT [5] and BLOOM [6]. Why does it work so well?

Firstly, we need to mention a few advantages of LayerNorm compared to BatchNorm:

- **Sequence data:** The technique introduces no dependency between training examples. Therefore, we can safely normalize sequence inputs without worrying about the different characteristics between training samples.

- **Flexible batch size:** Since the normalization is done on each sample, batch size is no longer a problem.
- **Training and Testing:** Unlike BatchNorm, LayerNorm does not need to keep the moving mean or variance of the population. Therefore, it performs the exact computation at training and inference.
- **Parallelization*:*** There is no dependency between training samples, so we can train the model on different devices without the need for synchronization.

Secondly, we will discuss one of the critical reasons normalization techniques help us so much with training stabilization: their invariance under weights and input transformation. Invariance means the normalization technique result does not get affected by input transformation.

Common known transformations are re-scaling and re-centering. Re-centering invariance makes the model insensitive to random noise in both weights and data. Meanwhile, re-scaling invariance keeps the output resilient to arbitrary scaling of inputs and weights.

It is much easier to understand the invariance characteristic by building a layer normalization function and trying it on different transformations to see how the mean, variance and result change.

```
def custom_ln(x: torch.Tensor, w : torch.Tensor, dim: Tuple[int], eps: float=1e-
  sum_input = x@w # multiply the input and the weight matrix
  mean = torch.mean(sum_input, dim=dim, keepdim=True) # get the mean on sample d
```

```
    std_var = torch.sqrt(torch.var(sum_input, dim=dim, keepdim=True) + eps) # get
    return (sum_input-mean)/(std_var )
```

I got the following result after re-scaling and re-centering the matrix weight (w) and dataset (x).

	Invariant	x	w	x*w	mean	var	layer_norm
Baseline		[5. 7.] [8. 8.] [5. 1.]	[5. 5. 6. 0.] [4. 9. 3. 9.]	[53. 88. 51. 63.] [72. 112. 72. 72.] [29. 34. 33. 9.]	[63.75] [82.] [26.25]	[17.] [20.] [11.7]	[-0.63 1.43 -0.75 -0.04] [-0.5 1.5 -0.5 -0.5] [0.24 0.66 0.58 -1.47]
Weight re-scale with delta= 0.7	Yes	[5. 7.] [8. 8.] [5. 1.]	[3.5 3.5 4.2 0.] [2.8 6.3 2.1 6.3]	[37.1 61.6 35.7 44.1] [50.4 78.4 50.4 50.4] [20.3 23.8 23.1 6.3]	[44.62] [57.4] [18.38]	[11.9] [14.] [8.19]	[-0.63 1.43 -0.75 -0.04] [-0.5 1.5 -0.5 -0.5] [0.24 0.66 0.58 -1.47]
Weight re-center with gamma= 0.5	Yes	[5. 7.] [8. 8.] [5. 1.]	[5.5 5.5 6.5 0.5] [4.5 9.5 3.5 9.5]	[59. 94. 57. 69.] [80. 120. 80. 80.] [32. 37. 36. 12.]	[69.75] [90.] [29.25]	[17.] [20.] [11.7]	[-0.63 1.43 -0.75 -0.04] [-0.5 1.5 -0.5 -0.5] [0.24 0.66 0.58 -1.47]
Dataset re-scale with delta= 0.7	Yes	[3.5 4.9] [5.6 5.6] [3.5 0.7]	[5. 5. 6. 0.] [4. 9. 3. 9.]	[37.1 61.6 35.7 44.1] [50.4 78.4 50.4 50.4] [20.3 23.8 23.1 6.3]	[44.62] [57.4] [18.38]	[11.9] [14.] [8.19]	[-0.63 1.43 -0.75 -0.04] [-0.5 1.5 -0.5 -0.5] [0.24 0.66 0.58 -1.47]
Dataset re-center with gamma= 0.5	No	[5.5 7.5] [8.5 8.5] [5.5 1.5]	[5. 5. 6. 0.] [4. 9. 3. 9.]	[57.5 95. 55.5 67.5] [76.5 119. 76.5 76.5] [33.5 41. 37.5 13.5]	[68.88] [87.12] [31.38]	[18.19] [21.25] [12.3]	[-0.63 1.44 -0.74 -0.08] [-0.5 1.5 -0.5 -0.5] [0.17 0.78 0.5 -1.45]

Layer Normalization results with different transformations (Image by the author)

As we can see, the normalized result remains the same with weight re-scaling and re-center. For dataset transformation, the technique is invariant to re-scaling but not re-centering.

It's pretty amazing how LayerNorm can keep the output resilient to such transformation. How can it do that? Let's delve into the mathematical details to grasp better what is happening under the hood.

Things get a bit intimidating here, so feel free to skip this section. However, I bet this will give you a strong intuition of how normalization works.

Mathematical proof of Layer Norm invariance

We have a neural network with weight W, input x, bias b and activation function f. The neural network output is $y = f(Wx + b)$.

Then LayerNorm can be expressed as:

$$y = f((Wx - \mu)/\sigma + b)$$

where μ and σ are the mean and variance vectors of Wx along the sample dimension.

The mathematical proof of LayerNorm variance can be shown as follows.

Weight matrix re-scaling invariance

Weight matrix re-scaling with factor δ

$$W' = \delta * W$$

$$\mu' = mean(W'x) = mean(\delta * Wx) = \delta * mean(Wx) = \delta * \mu$$

$$\sigma' = var(W'x) = var(\delta * Wx) = \delta * var(Wx) = \delta * \sigma$$

$$y' = f(\frac{W'x - \mu'}{\sigma'} + b)$$

$$y' = f(\frac{\delta * Wx - \delta * \mu}{\delta * \sigma} + b)$$

$$y' = f(\frac{Wx - \mu}{\sigma} + b)$$

$$y' = y$$

Weight matrix re-scaling invariance (Image by the author)

Weight matrix re-centering invariance

Weight matrix re-center with factor γ

$$W' = 1\gamma^T + W$$
$$\mu' = mean(W'x) = mean((1\gamma^T + W)x) = mean(1\gamma^T x) + mean(Wx) = 1\gamma^T x + \mu$$
$$\sigma' = var(W'x) = var((1\gamma^T + W)x) = var(Wx) = \sigma$$
$$y' = f(\frac{W'x - \mu'}{\sigma'} + b)$$
$$y' = f(\frac{(1\gamma^T + W)x - (1\gamma^T x + \mu)}{\sigma} + b)$$
$$y' = f(\frac{Wx - \mu}{\sigma} + b)$$
$$y' = y$$

Weight matrix re-centering invariance (Image by the author)

Dataset re-scaling invariance

Dataset re-scaling with factor δ

$$x' = \delta * x$$
$$\mu' = mean(Wx') = mean(W(\delta * x)) = \delta * mean(Wx) = \delta * \mu$$
$$\sigma' = var(Wx') = var(W(\delta * x)) = \delta * var(Wx) = \delta * \sigma$$
$$y' = f(\frac{Wx' - \mu'}{\sigma'} + b)$$
$$y' = f(\frac{W(\delta * x) - \delta * \mu}{\delta * \sigma} + b)$$
$$y' = f(\frac{Wx - \mu}{\sigma} + b)$$
$$y' = y$$

Dataset rep-scaling invariance (Image by the author)

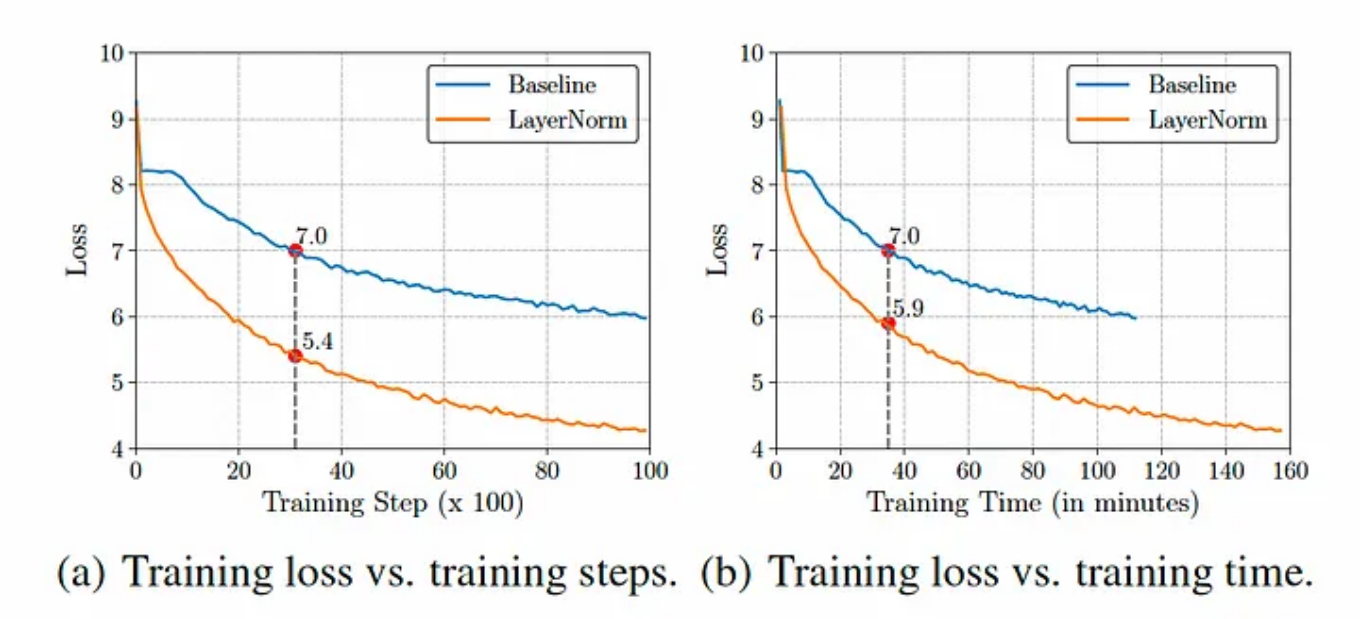
So that's the proof of LayerNorm invariance to transformations. However, not all invariance is necessary. RMSNorm [3] is a younger sibling of

LayerNorm with only re-scaling invariance characteristics. However, it has become a favoured choice for recent LLM architectures like Llama [4].

What makes RMSNorm more superior?

What is RMSNorm?

RMSNorm was published in 2019, with RMS stands for “root mean square”. Despite LayerNorm accelerating convergence, the authors pointed out that it consumes more time for each training step.



Training procedure of a GRU-based RNNSearch [3]

The author also argued that the mean normalization of LayerNorm has an insignificant impact on the final performance and, hence, can be removed for computation efficiency.

Given that hypothesis, RMSNorm is proposed to focus on the re-scaling invariance and using root mean square regularization as follows.

$$RMSNorm:$$

$$a = Wx$$

$$RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$$

$$\overline{a_i} = \frac{a_i}{RMS(a)}$$

RMSNorm (Image by the author)

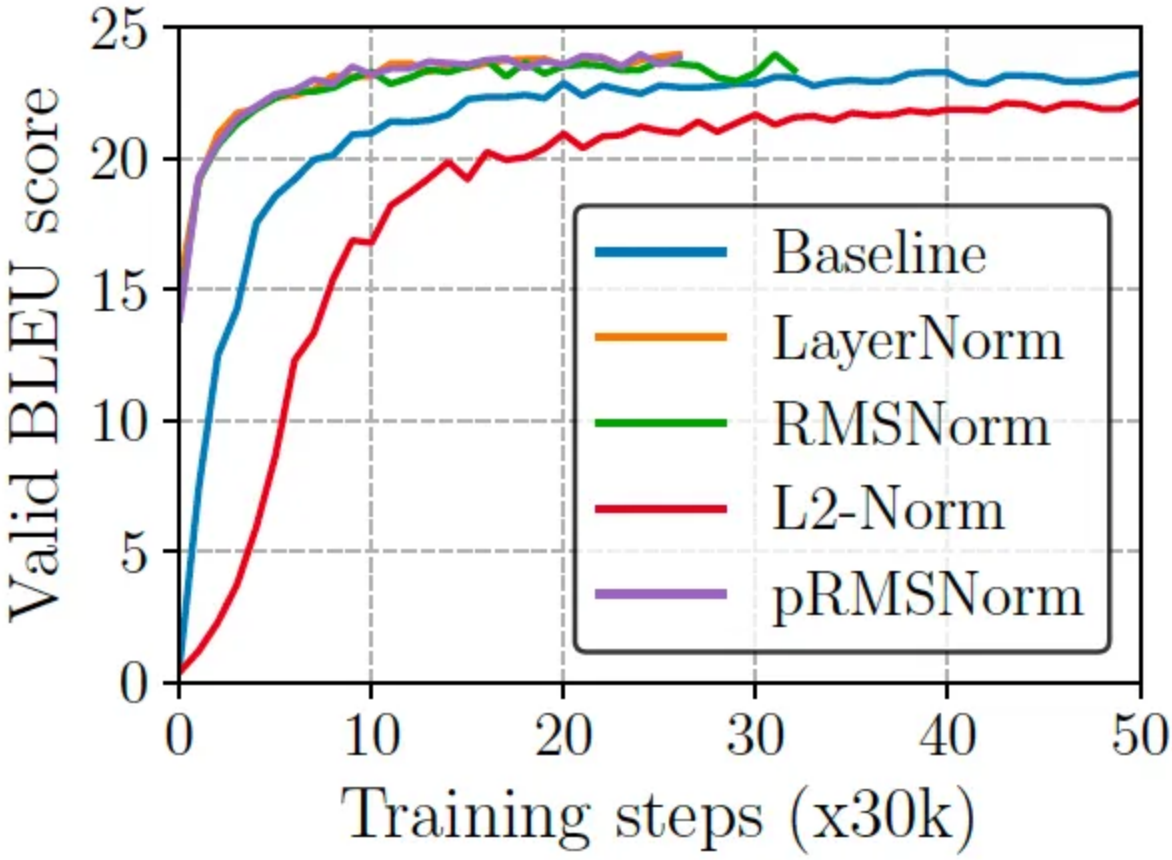
As we skip the mean computation, the normalization becomes simpler. With such elegant optimization, the author observed speedups of 7%-64% across different implementations without performance degradation!

One of the experiments is the WMT14 English-German translation task with a GRU-based RNNSearch using BLEU score metric. LayerNorm and RMSNorm both achieved similar scores on the test. But RMSNorm was much faster with 25% less training time.

Model	Test14	Test17	Time
Baseline	21.7	23.4	399±3.40s
LayerNorm	22.6	23.6	665±32.5s
L2-Norm	20.7	22.0	482±19.7s
RMSNorm	22.4	**23.7**	501±11.8s (24.7%)
*p*RMSNorm	**22.6**	23.1	493±10.7s (25.9%)

BLEU score on test14 and test17 [3]

A closer look at the validation score during training also suggests RMSNorm performance is comparable to LayerNorm during all training stages. This supports the initial argument that re-centering invariance is insignificant and RMSNorm is more efficient.



SacreBLEU score on newstest2013 for the RNNSearch [3]

Another intriguing metric is hidden vectors' mean and standard deviation at different token positions. While the baseline's mean and variance vary greatly, using LayerNorm and RMSNorm helps remarkably stabilize the distribution output. This is a strong proof of how vital normalization is to NLP models.

Model		1	2	3	4	ALL
Baseline	M	-2.60	-1.19	-1.43	-1.53	-1.60
	S	7.35	2.33	2.61	2.73	3.04
LayerNorm	M	-0.43	-0.48	-0.50	-0.50	-0.51
	S	1.19	1.51	1.51	1.51	1.51
RMSNorm	M	-0.40	-0.60	-0.69	-0.74	-0.73
	S	1.27	1.51	1.50	1.49	1.50

Mean (M) and standard deviation (S) statistics estimated for tokens at specific positions [3]

I hope you find this article helpful in understanding each normalization technique's pros and cons for the NLP tasks.

See you in the next post!

Reference

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. ICML, 2015.

[3] Zhang, Biao, and Rico Sennrich. "Root mean square layer normalization." *Advances in Neural Information Processing Systems* 32 (2019).

[4] Touvron, Hugo, et al. "Llama: Open and efficient foundation language models." *arXiv preprint arXiv:2302.13971* (2023).

[5] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

[6] Scao, Teven Le, et al. “Bloom: A 176b-parameter open-access multilingual language model.” *arXiv preprint arXiv:2211.05100* (2022).

NLP Deep Learning Large Language Models Machine Learning Tips And Tricks

Written by Thao Vu

Following

113 Followers · Writer for Towards Data Science

MLE @Tiktok. Follow me at https://www.linkedin.com/in/vu-phuong-thao-5ab465130/

More from Thao Vu and Towards Data Science

Thao Vu in Towards Data Science

Correct Sampling Bias for Recommender Systems

What is sampling bias in recommendation, and how to correct them

✨ · 8 min read · 3 days ago

113

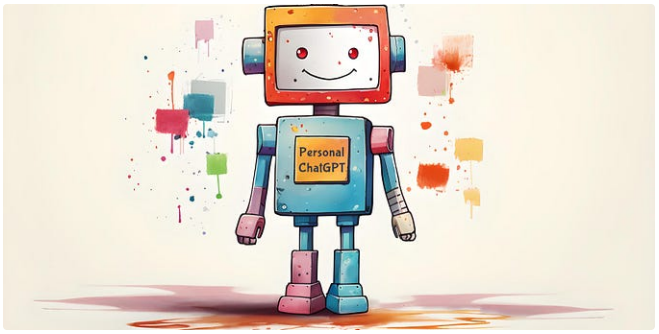
Antonis Makropoulos in Towards Data Science

How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and...

10 min read · Sep 17

549 11



Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT

Thao Vu in Towards Data Science

BERT vs GPT: Comparing the NLP Giants

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you...

✦ · 15 min read · Sep 8

595 7

How different are their structure, and how do the differences impact the model's ability?

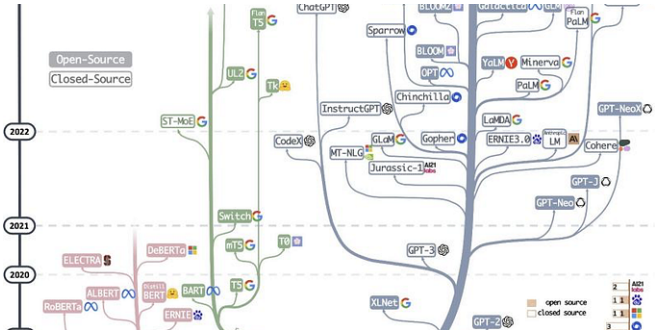
✦ · 7 min read · Aug 20

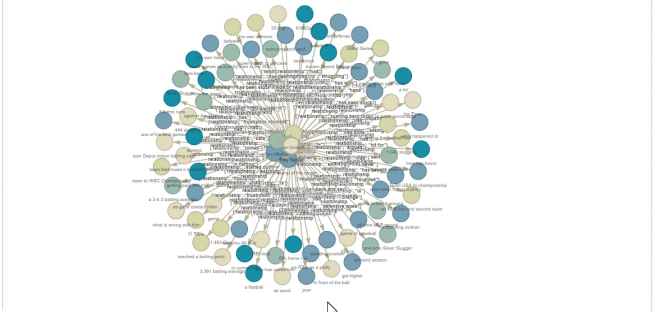
284 1

See all from Thao Vu

See all from Towards Data Science

Recommended from Medium





Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

372

Wenqi Glantz in Better Programming

7 Query Strategies for Navigating Knowledge Graphs With...

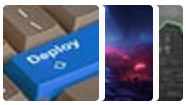
Exploring NebulaGraph RAG Pipeline with the Philadelphia Phillies

· 17 min read · 4 days ago

501 4

Lists

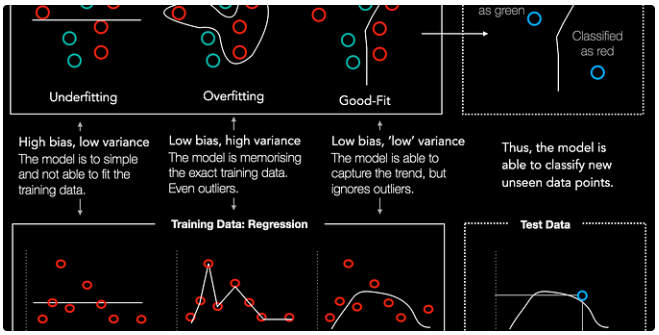
Natural Language Processing
669 stories · 283 saves



Predictive Modeling w/ Python
20 stories · 452 saves

Practical Guides to Machine Learning
10 stories · 519 saves

The New Chatbots: ChatGPT, Bard, and Beyond
13 stories · 133 saves



Han HELOIR, Ph.D. in Artificial Corner

MongoDB and Langchain Magic: Your Beginner's Guide to Setting...

Introduction:

✨ · 7 min read · Sep 12

1.4K 12

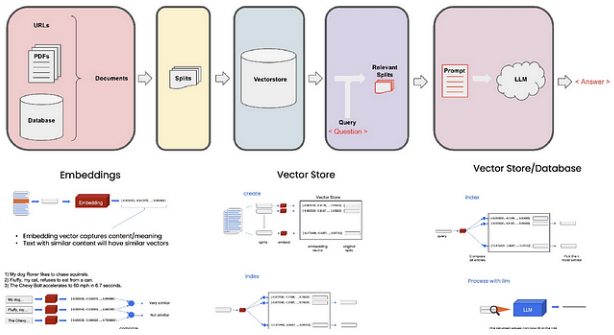
Frederik vl in Advanced Deep Learning

Understanding Bias and Variance in Machine Learning

The terms bias and variance describe how well the model fits the actual unknown data...

3 min read · Sep 15

44



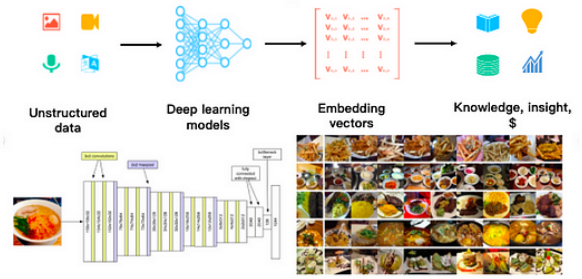
TeeTracker

Chat with your PDF (Streamlit Demo)

Conversation with specific files

4 min read · Sep 15

56



Jayita Bhattacharyya in GoPenAI

Primer on Vector Databases and Retrieval-Augmented Generation...

Vector Databases Generation (RAG) Langchain Pinecone HuggingFace Large...

9 min read · Aug 16

228 1

See more recommendations