An Overview of the LoRA Family

LoRA, DoRA, AdaLoRA, Delta-LoRA, and more variants adaptation.

Dorian Drost
Mar 10, 2024 22 min read

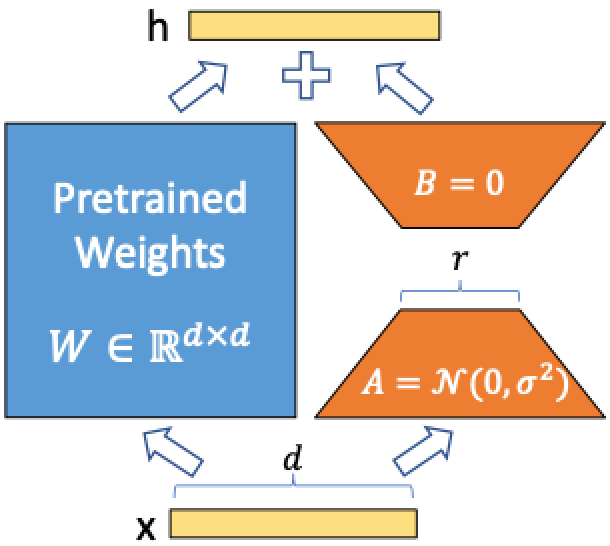
LoRA comes in different shapes and varieties. Photo by Lucas Georg

Low-**R**ank **A**daptation (**LoRA**) can be considered breakthrough towards the ability to train large l specific tasks efficiently. It is widely used today applications and has inspired research on how t main ideas to achieve better performance or train models even faster.

In this article, I want to give an overview of some variants of LoRA, that promise to improve LoRAs capabilities in different ways. I will first explain the basic concept of LoRA itself, before presenting **LoRA+**, **VeRA**, **LoRA-FA**, **LoRA-drop**, **AdaLoRA**, **D**... and **Delta-LoRA**. I will introduce the basic concepts and m... show, how these approaches deviate from the o... spare technical details, unless they are importa... concepts, and will also not discuss evaluations ... readers interested, I linked the original papers a...

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

Lora



The main idea of LoRA is to add two smaller tunable matrices A and B next to ... without changing the parameters of W. Image fron

Low-Rank Adaption (**LoRA**) [1] is a technique, th... today to train large language models (LLMs). La... come with the capability to predict tokens of na... a natural language input. This is an astonishing ... solving many problems this is not enough. Most want to train an LLM on a given downstream task, such as classifying sentences or generating answers to given questions. The most straightforward way of doing that is fine-tuning, where you train some of the layers of the LLM with data of the desired task. That means training very big models with millions to billions of parameters though.

LoRA gives an alternative way of training that is easier to conduct due to a drastically reduced n parameters. Next to the parameter weights of a trained LLM layer, LoRA introduces two matrice called *adapters* and that are much smaller. If th parameters W is of size *d x d*, the matrices A an and *r x d*, where *r* is much smaller (typically bel parameter *r* is called the *rank*. That is, if you use of *r=16*, these matrices are of shape *16 x d*. The more parameters you train. That can lead to bet the one hand but needs more computation time

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

Now that we have these new matrices A and B, them? The input fed to W is also given to B*A, ar* is added to the output of the original matrix W. some parameters on top and add their output t prediction, which allows you to influence the m don't train W anymore, which is why we sometir *frozen*. Importantly, the addition of A and B is nc very end (which would just add a layer on top) b to layers deep inside the neural network.

That is the main idea of LoRA, and its biggest ac have to train fewer parameters than in fine-tuni comparable performance. One more technical d mention at this place: At the beginning, the mat with random values of mean zero, but with some variance around that mean. The matrix B is initialized as a matrix of complete zeros. This ensures, that the LoRA matrices don't change the output of the original W in a random fashion from the very beginning. The update of A and B on W's output should rather be an addition to the original output, once the parameters of A and B are being tuned in the desired direction. However, we will later s approaches deviate from this idea for different r

LoRA as just explained is used very often with t However, by now there are many variants of LoR the original method in different ways and aim at performance, or both. Some of these I want to p the following.

LATEST

EDITOR'S PICKS

DEEP DIVES

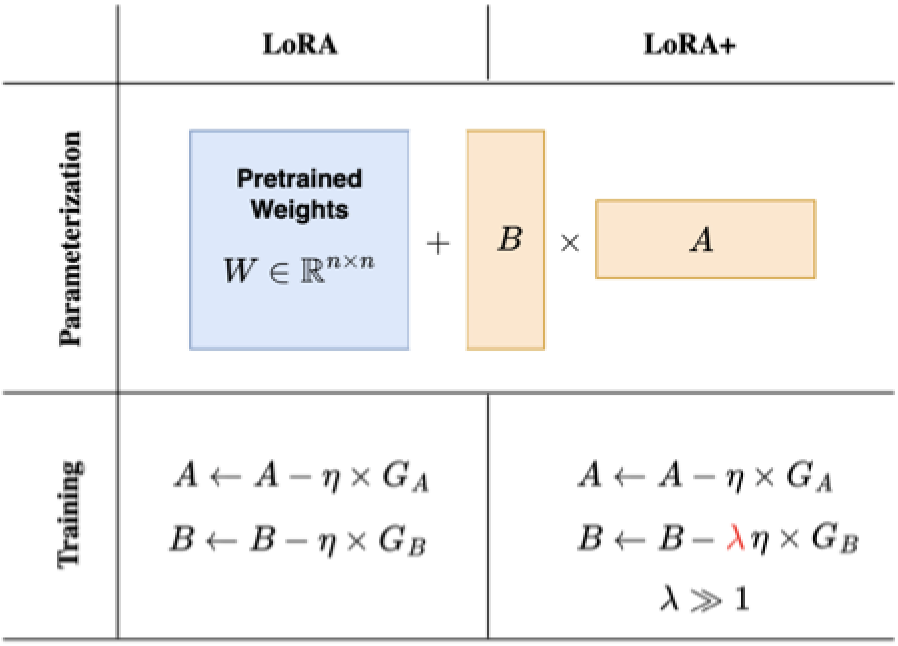
CONTRIBUTE

NEWSLETTER

Sign in

TDS is Now Independent!

LoRA+



LoRA+ introduces different learning rates for the two matrices A and B, here inc from [2].

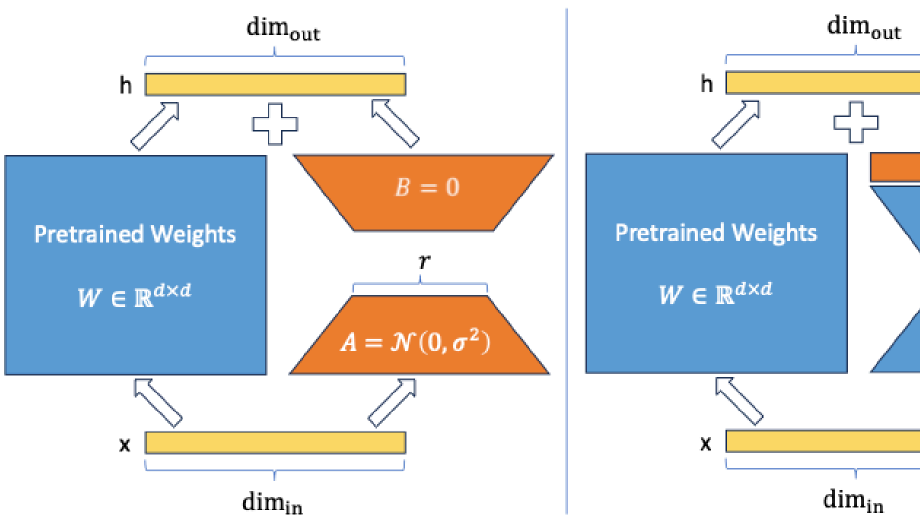
LoRA+ [2] **** introduces a more efficient way adapters by introducing different learning rates

B. Most of the time, when training a neural network, there is just one learning rate that is applied to all weight matrices the same way. However, for the adapter matrices used in LoRA, the authors of LoRA+ can show, that it is suboptimal to have that single learning rate. The training becomes more efficient by setting the learning rate of matrix B much higher than that of matrix A.

There is a theoretical argument to justify that a[
bases on numerical caveats of a neural network
model becomes very wide in terms of the numb
However, the math required to prove that is qui
you are really into it, feel free to take a look at t
[2]). Intuitively, you may think that matrix B, whi
zero, could use bigger update steps than the ra
matrix A. In addition, there is empirical evidence
improvement by that approach. By setting the le
matrix B 16 times higher than that of matrix A, t
been able to gain a small improvement in mode
2%), while speeding up the training time by fact
such as RoBERTa or Llama-7b.

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

VeRA



VeRA doesn't train A and B, but initializes them to a random projection and trains additional vectors d and b instead. Image from [3].

With **VeRA** (**Ve**ctor-based **R**andom Matrix **A**daptation) [3], the authors introduce an approach to drastically reduce the parameter size of the LoRA adapters. Instead of training the matrices A and B, which is the core idea of LoRA in the first place, they initialize these matrices with shared random weights (i.e and B in all the layers have the same weights) a vectors d and b. Only these vectors d and b are following.

You may wonder how this can work at all. A and random weights. How should they contribute an model's performance, if they are not trained at a based on an interesting field of research on so- projections. There is quite some research that in large neural network only a small fraction of the steer the behavior and lead to the desired perfo the model was trained for. Due to the random in parts of the model (or sub-networks) are contrib towards the desired model behavior from the ve the training, all parameters are trained though, a which are the important subnetworks. That mak costly, as most of the parameters that are upda value to the model's prediction.

Based on this idea, there are approaches to only relevant sub-networks. A similar behavior can b training the sub-networks themselves, but by ad vectors after the matrix. Due to the multiplicati with the vector, this can lead to the same outpu sparse parameters in the matrix would. That is e authors of VeRA propose by introducing the vect

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

Sign in

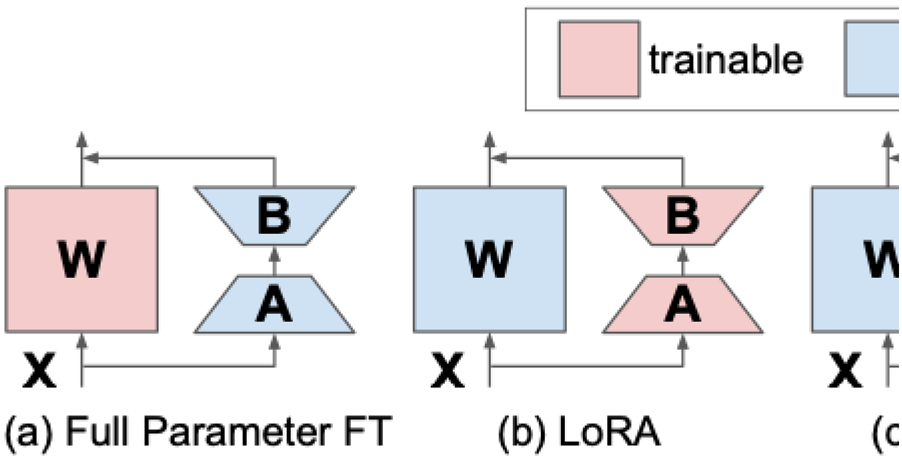
TDS is Now Independent!

are trained, while the matrices A and B are frozen. Also, in contrast to the original LoRa approach, matrix B is not set to zero anymore but is initialized randomly just as matrix A.

This approach naturally leads to a number of parameters that is much smaller than the full matrices A and B. For example, if you introduce LoRA-layers of rank 16 to GPT-3, you would have 75.5M parameters. With VeRA, you only have 2.8M (a re But how is the performance with such a small r parameters? The authors of VeRA performed an some common benchmarks such as GLUE or E2 based on RoBERTa and GPT2 Medium. Their resu the VeRA model yields performance that is only than models that are fully finetuned or that use technique.

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

LoRA-FA



LoRA-FA freezes matrix A and only trains matrix B. Imag

Another approach, **LoRA-FA** [4], which stands fo **F**rozen-**A,** is going in a similar direction as VeRA matrix A is frozen after initialization and hence projection. Instead of adding new vectors, matri though, after being initialized with zeros (just as

LoRA). This halves the number of parameters while having comparable performance to normal LoRA.

LoRa-drop

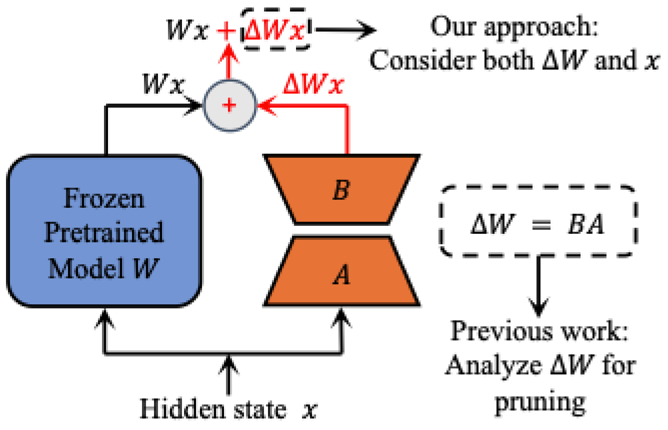


Figure 1: The diagram of LoRA

LoRA-drop uses the output of B*A to decide, which LoRA-layers are worth to

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

Sign in

TDS is Now Independent!

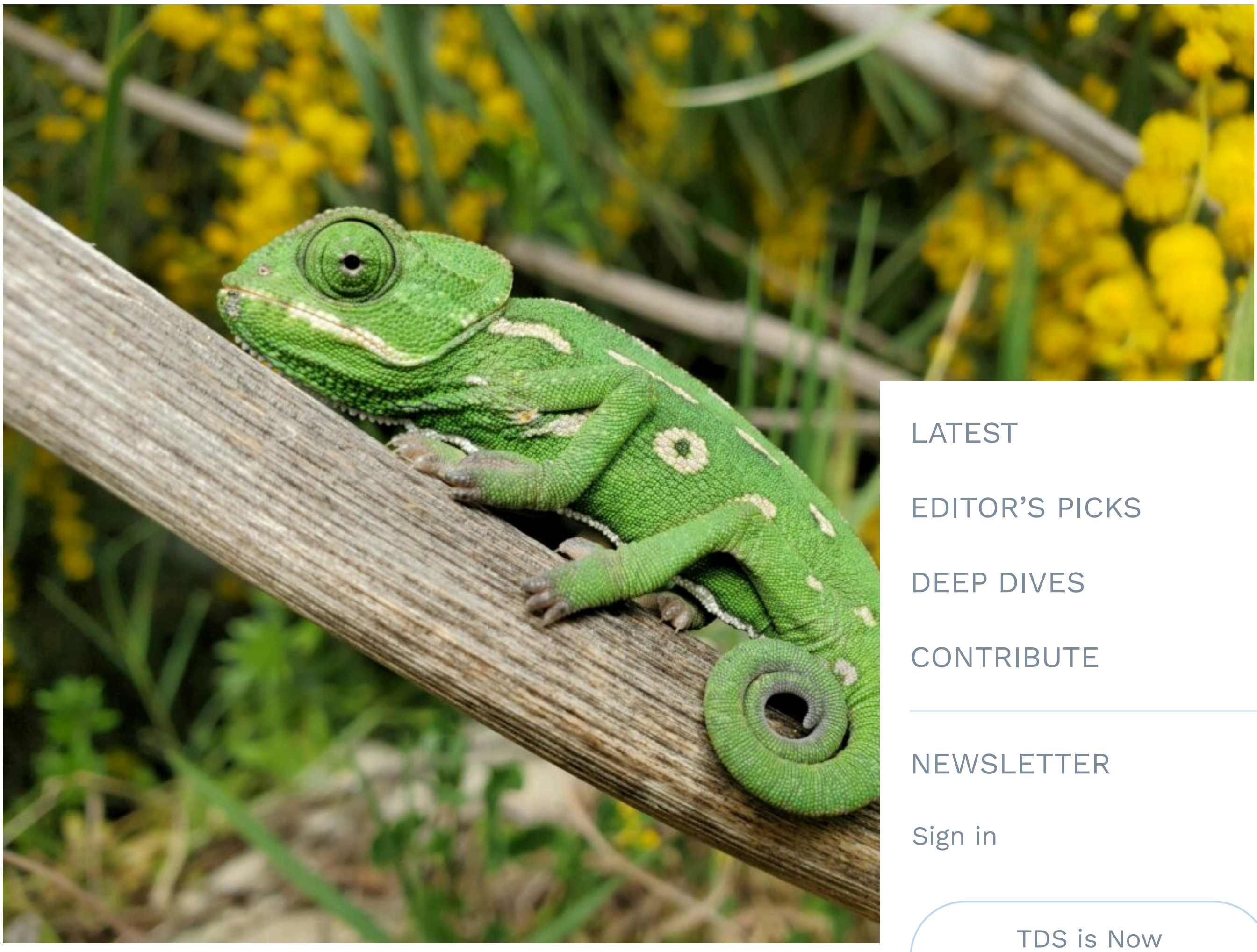
In the beginning, I explained, that you can add L layer in the neural network. **LoRA-drop** [5] intro to decide which layers are worth to be enhance which this is not worth the effort. Even if trainir much cheaper than finetuning the whole model adapters you add, the more expensive is the tra

LoRA-drop consists of two steps. In the first ste subset of the data and train the LoRA adapters Then you calculate the importance of each LoR/ where A and B are the LoRA matrices, and x is t simply the output of the LoRA adapters that is a of the frozen layer each. If this output is big, it behavior of the frozen layer more drastically. If i indicates that the LoRA adapter has only little ir frozen layer and could as well be omitted.

Given that importance, you now select the LoRA layers that are most important. The are different ways of doing that. You can sum up the importance values until you reach a threshold, which is controlled by a hyperparameter, or you just take the top n LoRA layers with the highest importance for a fixed n. In any case, in the next step, you conduct the full training on the whole dataset (remember that you used a subset of data for th
but only on those layers that you just selected.
fixed to a shared set of parameters that won't b
during training.

The algorithm of LoRA-drop hence allows to tra
just a subset of the LoRA layers. The authors pr
evidence that indicates only marginal changes ir
compared to training all LoRA layers, but at redu
time due to the smaller number of parameters t
trained.

AdaLoRA



AdaLoRA allows to adapt the rank of the LoRA matrices dynamically. Photo b
Unsplash

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

Sign in

TDS is Now Independent!

There are alternative ways how to decide which are more important than others. In this section, **AdaLoRA** [6], which stands for **Ada**ptive LoRa. W adaptive here? It is the rank (i.e. the size) of the main problem is the same as in the previous se worth adding LoRA matrices A and B to each lay layers, the LoRA training may be more importan more change in the model's behavior) than for c that importance, the authors of AdaLoRA propo singular values of the LoRA matrices as indicato importance.

What is meant by that? First, we have to unders multiplication can also be seen as applying a fu

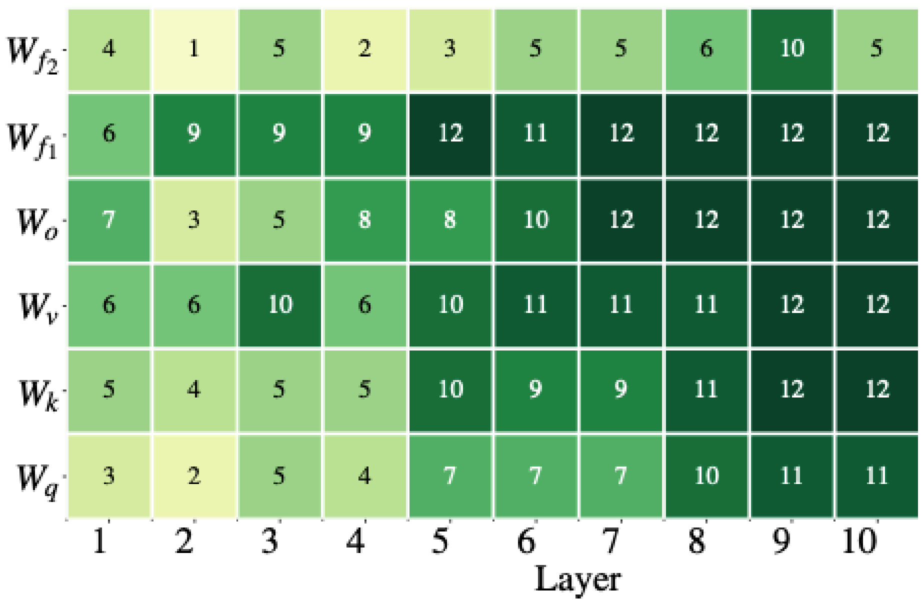
When dealing with neural networks, this is quite obvious: Most of the time you use neural networks as functions, i.e. you give an input (say, a matrix of pixel values) and obtain a result (say, a classification of an image). Under the hood, this function application is powered by a sequence of matrix multiplications. Now, let's say you want to reduce the number of parameters in such a matrix. That will change the function's be
want it to change as little as possible. One way
compute the eigenvalues of the matrix, which te
variance is captured by the rows of the matrix e
decide to set some rows to zero, that capture o
of the variance, and hence don't add much infor
function. This is the main idea of AdaLoRA since
aforementioned singular values are exactly the s
eigenvalues. That is, based on the singular value
which rows of which LoRA matrices are more im
can be omitted. This effectively shrinks the rank
which have many rows that don't contribute mu
an important difference to LoRA-drop from the
LoRA-drop, the adapter of a layer is selected to
fully, or not trained at all. AdaLoRA can also dec
adaptors for some layers but with lower rank. Tl
end, different adaptors can have different ranks
original LoRA approach, all adaptors have the sa

There are some more details to the AdaLoRA ap
omitted for brevity. I want to mention two of the
the AdaLoRA approach does not calculate the s
explicitly all the time (as that would be very cos
decomposes the weight matrices with a singula
decomposition. This decomposition is another w
the same information as in a single matrix, but
singular values directly, without costly computa

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

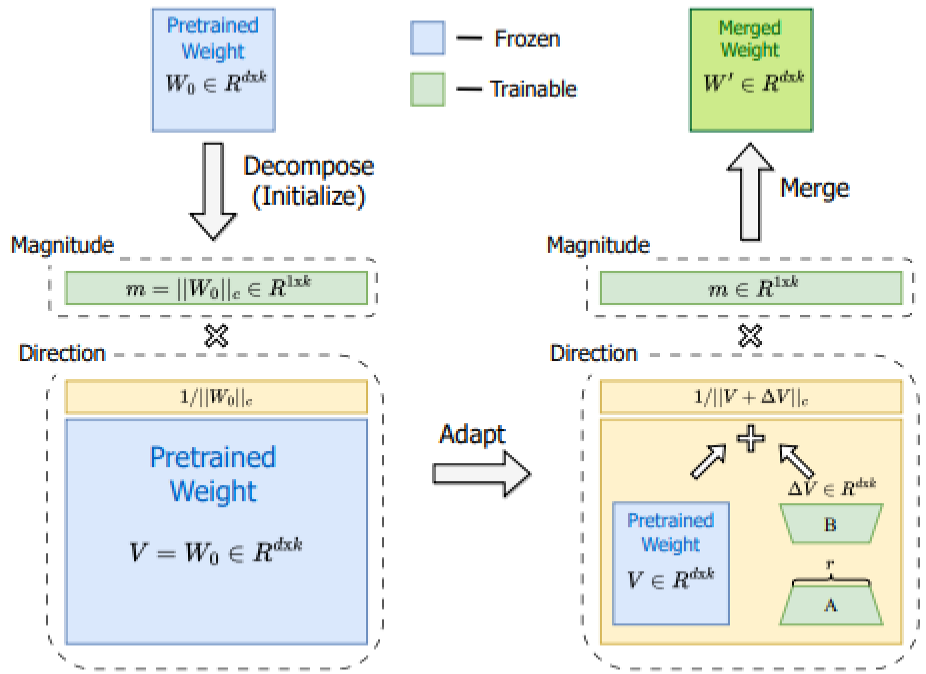
Second, AdaLoRA does not decide on the singular values alone but also takes into account the sensitivity of the loss to certain parameters. If setting a parameter to zero has a large influence on the loss, this parameter is said to have high sensitivity. When deciding where to shrink the rank, the mean sensitivity of a row's elements is taken into consideration in addition to the singular value.

Empirical evidence for the value of the approacl comparing AdaLoRA with standard LoRA of the s That is, both approaches have the same numbe total, but these are distributed differently. In Lo have the same rank, while in AdaLoRA, some ha some have a lower rank, which leads to the sam parameters in the end. In many scenarios, AdaL scores than the standard LoRA approach, indica distribution of trainable parameters on parts of of particular importance for the given task. The an example, of how AdaLoRA distributed the rar model. As we see, it gives higher ranks to the la end of the model, indicating that adapting these



On different layers of the network, the LoRA matrices are given different rank higher, in general. Image from [6].

DoRA

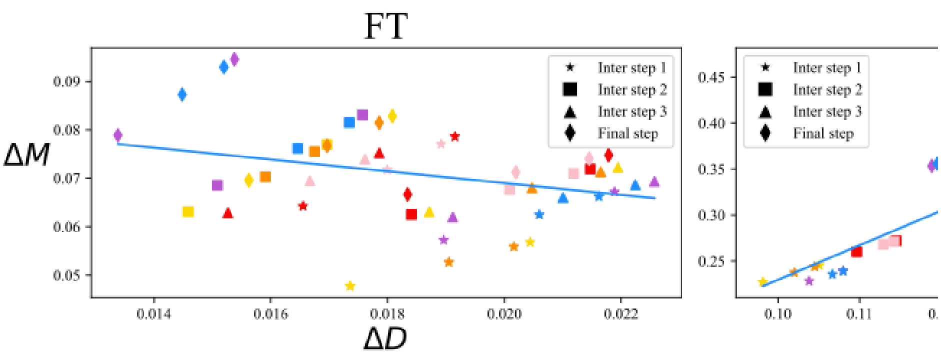


In DoRA, the weight matrix W is decomposed into magnitude m and direction \
Image from [7].

Another approach to modify LoRa to get better |
Weight-**D**ecomposed L**o**w-**R**ank **A**daption, or **Do**
with the idea, that each matrix can be decompo
product of a magnitude and a direction. For a ve
you can easily visualize that: A vector is nothing
starting at the position of zero and ending at a
vector space. With the vector's entries, you spe
by saying x=1 and y=1, if your space has two dim
Alternatively, you could describe the very same
way by specifying a magnitude and an angle (i.e.
as m=√2 and a=45°. That means that you start a
move in the direction of 45° with an arrow lengt
lead you to the same point (x=1,y=1).

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

This decomposition into magnitude and direction can also be done with matrices of higher order. The authors of DoRA apply this to the weight matrices that describe the updates within the training steps for a model trained with normal fine-tuning and a model trained with LoRA adapters. A comparison of these two techniques we see in the following plot:



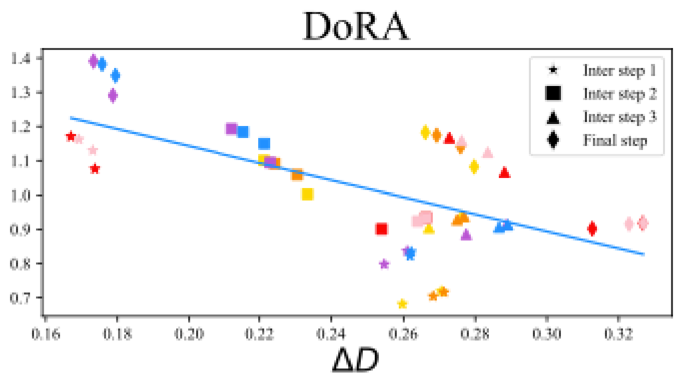
Finetuning and LoRA differ in the relationship between the changes in magnitu

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

We see two plots, one for a fine-tuned model (l model trained with Lora adapters (right). On the change in direction, on the y-axis we see the ch and each scatter point in the plots belongs to o model. There is an important difference betwee training. In the left plot, there is a small negativ between the update in direction and the update while in the right plot, there is a positive relatio much stronger. You may wonder which is better, any meaning at all. Remember, that the main id achieve the same performance as finetuning, bu parameters. That means, ideally we want LoRA's many properties with fine-tuning as possible, as not increase the costs. If the correlation betwee magnitude is slightly negative in fine-tuning, thi property for LoRA as well, if it is achievable. In c relationship between direction and magnitude i compared to full fine-tuning, this may be one of the reasons why LoRA sometimes performs less well than fine-tuning.

The authors of DoRA introduce a method to train magnitude and direction independently by separating the pretrained matrix W into a magnitude vector m of size *1 x d* and a direction matrix V. The direction matrix V is then enhanced by B*A, as known from the standard LoRA approach, and m is trained as it because it has just one dimension. While LoRA t both magnitude and direction together (as indic positive correlation between these two), DoRA c adjust the one without the other, or compensat with negative changes in the other. We can see between direction and magnitude is more like tl

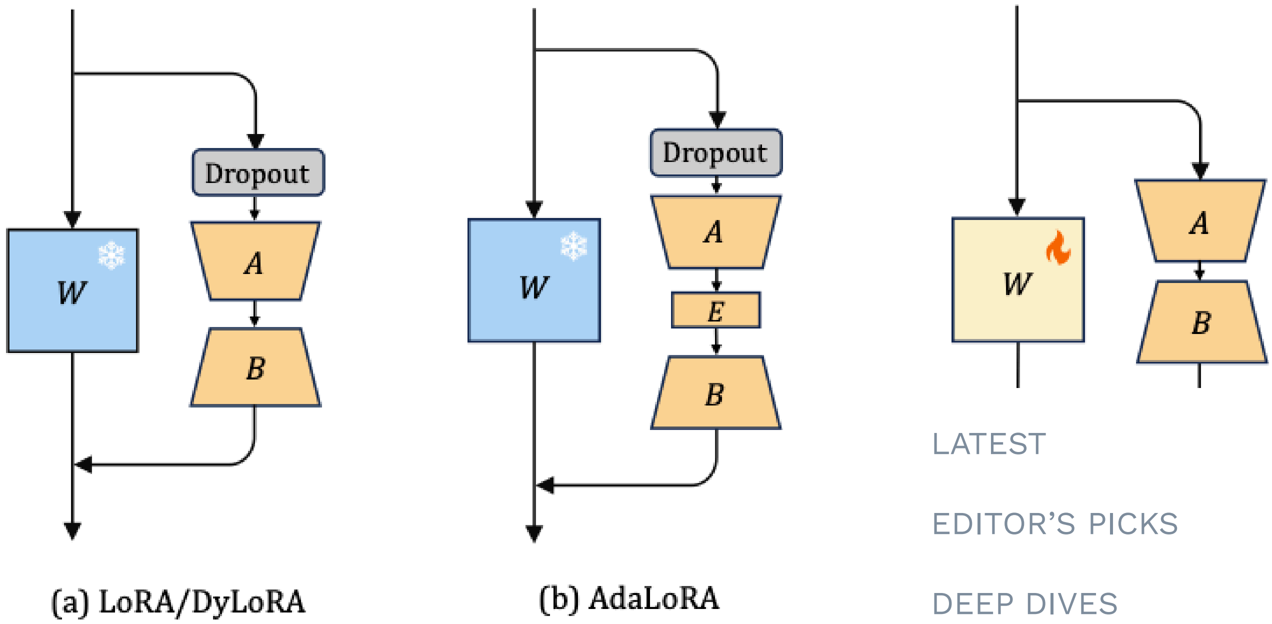
LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!



For DoRA, the relationship between magnitude and direction is more like tha

On several benchmarks, DoRA outperforms LoR Decomposing the weight updates into magnitud allow DoRA to perform a training that is closer t in fine-tuning, while still using the smaller para introduced by LoRA.

Delta-LoRA



Delta-LoRA doesn't freeze the matrix W but updates it with the gradient obta

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

Delta-LoRA [8] **** introduces yet another idea This time, the pre-trained matrix W comes into Remember that the main idea in LoRA is to not trained matrix W, as that is too costly (and that fine-tuning). That is why LoRA introduced new s and B. However, those smaller matrices have les learn the downstream task, which is why the pe LoRA-trained model is often lower than the perf tuned model. Tuning W during training would be we afford that?

The authors of Delta-LoRA propose to update th gradients of AB, *which is the difference between* consecutive time steps. This gradient is scaled w hyperparameter λ, which controls, how big the i training on the pre-trained weights should be, a W (while α and r (the rank) are hyperparameters LoRA setup):

$$\boldsymbol{W}^{(t+1)} = \boldsymbol{W}^{(t)} + \lambda \cdot \frac{\alpha}{r} \cdot \triangle \boldsymbol{AB}, \text{where } \triangle \boldsymbol{AB} = \boldsymbol{A}^{(t}$$

W is updated with the difference of AB in two consecutive steps. Image from [8].

That introduces more parameters to be trained at almost no computational overhead. We do not have to calculate the gradient for the whole matrix W, as we would within finetuning, but update it with a gradient we already got in the LoRA training anyway. The authors compared this method on a number of benchmarks using models like RoBERTA and GPT-2 and found a bo over the standard LoRA approach.

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

Summary

Congrats. You've made it to the end. Photo by david Griffith

We just saw a number of approaches, that vary LoRA to reduce computation time or improve pe both). In the end, I will give a short summary of approaches:

- **LoRA** introduces low-rank matrices A and B that are trained, while the pre-trained weight matrix W is frozen.
- **LoRA+** suggests having a much higher learning rate for B than for A.
- **VeRA** does not train A and B, but initializes them randomly and trains new vectors d and b on top.
- **LoRA-FA** only trains matrix B.
- **LoRA-drop** uses the output of B*A to deterr
are worth to be trained at all.
- **AdaLoRA** adapts the ranks of A and B in diff
dynamically, allowing for a higher rank in the
more contribution to the model's performar
- **DoRA** splits the LoRA adapter into two com
magnitude and direction and allows to train
independently.
- **Delta-LoRA** changes the weights of W by th

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

The field of research on LoRA and related meth
vivid, with new contributions every other day. In
wanted to explain the core ideas of some appro
that was only a selection of such, that is far aw
complete review.

I hope that I have been able to share some kno
possibly inspire you to new ideas. LoRA and rela
field of research with great potential, as we saw
breakthroughs in improving performance or com
training large language models can be expected

References and Further Reading

These are the papers on the concepts explainec

- **[1] LoRA**: Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., … & Chen, W. (2021). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- **[2] LoRA**+: Hayou, S., Ghosh, N., & Yu, B. (2024). LoRA+: Efficient Low Rank Adaptation of Large Models. *arXiv preprint arXiv:2402.12354*.
- **[3] VeRA**: Kopiczko, D. J., Blankevoort, T., & A Vera: Vector-based random matrix adaptatio *arXiv:2310.11454*.
- **[4]: LoRA-FA**: Zhang, L., Zhang, L., Shi, S., C (2023). Lora-fa: Memory-efficient low-rank a language models fine-tuning. *arXiv preprint*
- **[5] LoRA-drop**: Zhou, H., Lu, X., Xu, W., Zhu, (2024). LoRA-drop: Efficient LoRA Paramete Output Evaluation. *arXiv preprint arXiv:2402.*
- **[6] AdaLoRA**: Zhang, Q., Chen, M., Bukharin, Chen, W., & Zhao, T. (2023). Adaptive budget parameter-efficient fine-tuning. *arXiv prepri*
- **[7] DoRA**: Liu, S. Y., Wang, C. Y., Yin, H., Molc F., Cheng, K. T., & Chen, M. H. (2024). DoRA: Decomposed Low-Rank Adaptation. *arXiv pr arXiv:2402.09353*.
- **[8]: Delta-LoRA**: Zi, B., Qi, X., Wang, L., Wang Zhang, L. (2023). Delta-lora: Fine-tuning hig with the delta of low-rank matrices. *arXiv pr arXiv:2309.02411*.

For some core ideas on random projection, as m section on VeRA, this is one of the major contrib

- Frankle, J., & Carbin, M. (2018). The lottery ti Finding sparse, trainable neural networks. *a arXiv:1803.03635*.

LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
TDS is Now Independent!

For a more fine-grained explanation of LoRA and DoRA, I can recommend this article:

- https://magazine.sebastianraschka.com/p/lora-and-dora-from-scratch

Like this article? Follow me to be notified of my future posts.

WRITTEN BY

Dorian Drost

See all from Dorian Drost

Topics: Adapter Dora Editors Pick Ll

Share this article:

Related Articles