RESOURCE CONSTRAINED BERT FINETUNING

How To Make The Most Out Of BERT Finetuning

Weight Initializations, Data Orders, and Early Stopping

Jonas Vetterle Apr 20, 2020 · 7 min read ★

Photo by Drew Patrick Miller on Unsplash

Finetuning pretrained language models like BERT on downstream tasks has become ubiquitous in NLP research and applied NLP. That's in part because one can save a lot of time and money by using pretrained models. They also often serve as strong baseline models which, when finetuned, significantly outperform training models from scratch.

While finetuning BERT is relatively straightforward in theory, it can be time-intensive and unrewarding in practice due to seemingly random outcomes of different training runs. In fact, even when finetuning a model with the same hyperparameters over and over again, there can be a great degree of variability in final model performance due to randomness in (1) weight intializations and (2) data orders (how data sets are shuffled). This is especially a problem when finetuning BERT on small data sets.

Recent research[1] explores these often overlooked sources of randomness. The authors offer **2 practical tips** you can use to finetune better models given a certain computational budget, thereby making the most out of BERT finetuning:

1. **Evaluate your model multiple times during an epoch; and**
2. **Identify bad initializations early and stop them.**

Apart from these practical tips, the paper provides two interesting insights:

1. Training BERT with the same hyperparameters as in the original setup, but with different random seeds, leads to a big performance increase, making it competitive with newer architectures like ALBERT on some tasks; and
2. There are some weight initializations which are globally better than others: they lead to better models on multiple tasks than other initializations.

. . .

Experimental Setup

The authors finetune BERT multiple times on 4 different GLUE binary classification datesets. Three of them are quite small (MRPC, RTE, and CoLA), and one is relatively big (SST).

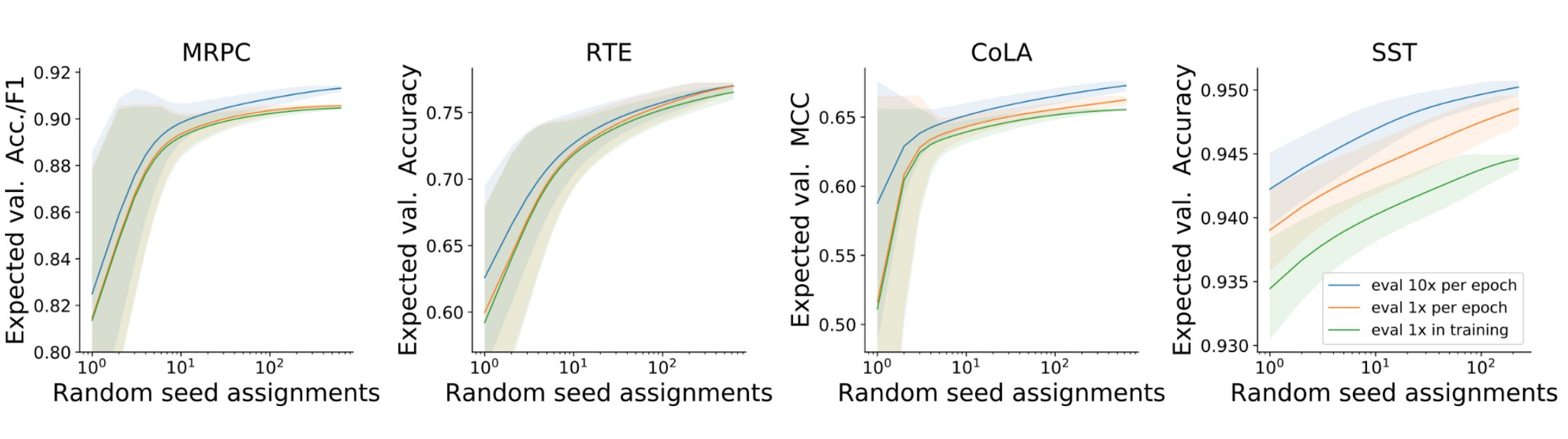
On each data set, they jointly finetune all BERT parameters and train a classification head in what is called an episode. In each episode, the hyperparameters are the same, but the random seeds which control weight initialization and data order are modified. The authors try 25 different random seeds for both randomizations on the small datasets, and 15 on SST. That is, in total they run 2,100 episodes (3 x $25^2 + 15^2$).

Tip #1: Evaluate often

The standard machine learning workflow amounts to training a certain number of models on training data, picking the preferred model on a validation set and evaluating its final performance on a test set.

Given this workflow, training more models naturally leads to higher expected performance of the best model and smaller variance. But training more models is also more time- and resource-intensive. So practitioners face a trade-off between expected performance and resource spent.

It is really important to know what this trade-off looks like. We can visualize it by plotting the Expected Validation Performance[2] after running a certain number of finetuning episodes:



x-axis: Number of finetuning episodes (models trained), y-axis: Expected performance of the best model. Evaluating models 10x per epoch leads to higher expected performance than evaluating it once per epoch or once in training. [Source]

As expected, when training a model with some hyperparameters only once, the variance in validation performance is huge. The variance decreases as more models (with the same hyperparameters, but different seeds for weight initialiazation and data order) are trained. Similarly, the expected value of the performance of the best model is increasing in the number of different initializations we try out.

The key take-away:

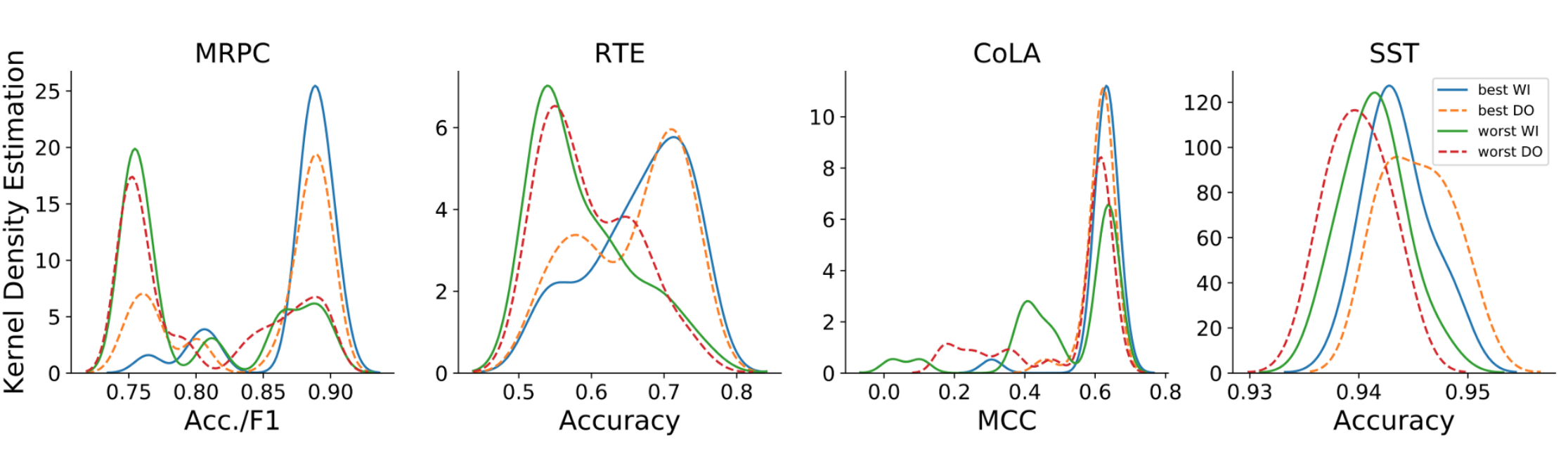
We see from the above graphs that evaluating models 10 times per epoch (blue lines) tends to lead to higher expected validation performance than evaluating once per epoch or less. The authors therefore conclude that

> "more frequently evaluating the model on validation data leads to higher expected validation values"

How big is the impact of random seeds?

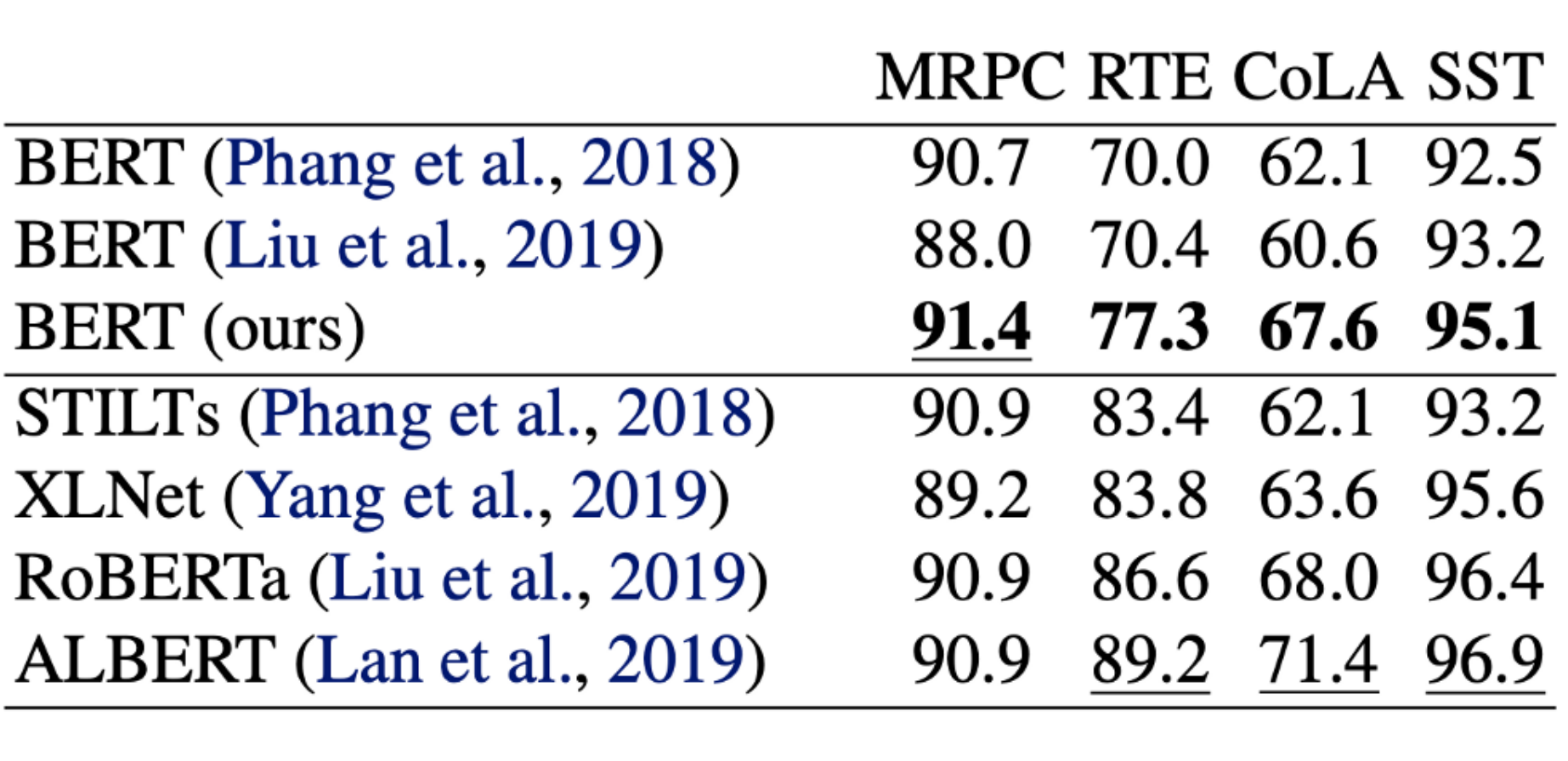
We've just seen than if we finetune BERT only once, with a certain hyperparameter setting, there is agreat amount of variance in validation performance. This is due to random weight initialization **(WI)** and data order **(DO)**. How much of an impact do these initializations have on performance?

To answer this question the authors plot the distribution of validation performance across all episodes of the best/worst WI/DO seeds for each dataset:



The density of validation performance tends to be located in lower performance regions for the worst seeds (green and dashed red line). This is particularly the case for MRPC and RTE where the distributions appear to be bimodal. [Source]

From the graphs above we see that the best seeds tend to have more density in high performance regions than the worst seeds. The authors conduct an ANOVA test which provides evidence that the distributions of the best and worst seeds indeed have different means.



	MRPC	RTE	CoLA	SST
BERT (Phang et al., 2018)	90.7	70.0	62.1	92.5
BERT (Liu et al., 2019)	88.0	70.4	60.6	93.2
BERT (ours)	**91.4**	**77.3**	**67.6**	**95.1**
STILTs (Phang et al., 2018)	90.9	83.4	62.1	93.2
XLNet (Yang et al., 2019)	89.2	83.8	63.6	95.6
RoBERTa (Liu et al., 2019)	90.9	86.6	68.0	96.4
ALBERT (Lan et al., 2019)	90.9	89.2	71.4	96.9

Training BERT multiple times with different random seeds leads to big improvements over previously reported scores [Source]

Different random seeds for weight initialization and data order can affect performance a lot. Compared to previously reported scores with the same hyperparameters and experimental setting, the authors' best finetuned BERT models perform much better. On some tasks, BERT even becomes competitive with more recent models like ALBERT.

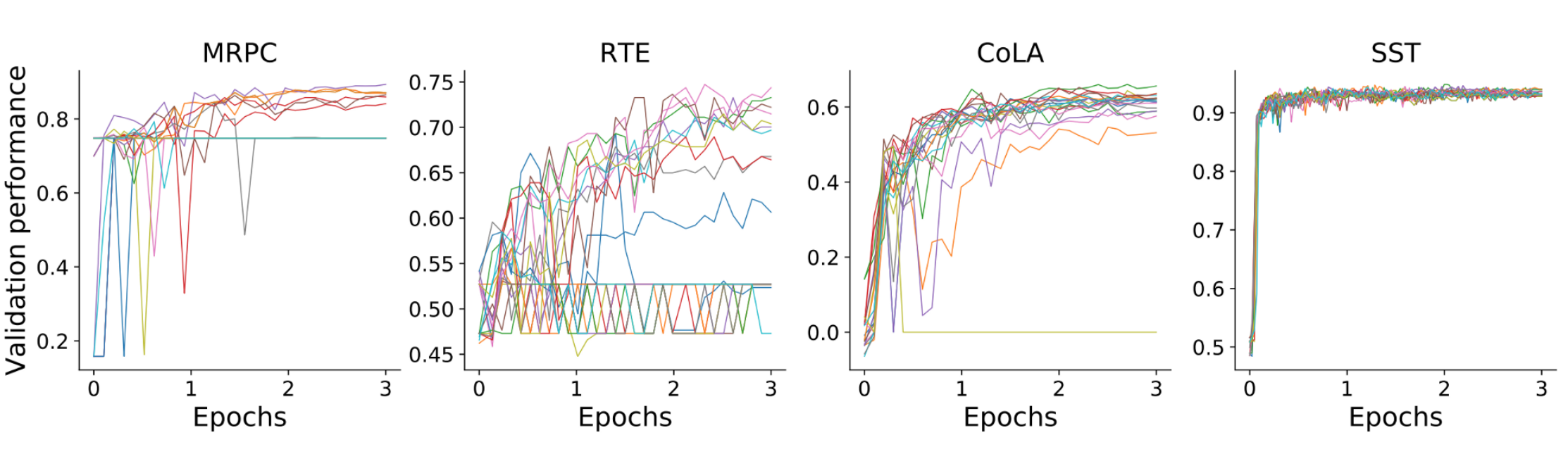
Interestingly, there appear to be **globally good initializations**. As all 4 tasks involve training a binary classifier, it is possible to check whether good seeds for weight initialization for one task are also good for other tasks. The authors find that this is the case: **there are some initializations that perform consistently well across all tasks!** This is really exciting and the authors leave this question open for future research.

The Key Takeaway:

The performance difference between best and worst random seeds is significant. *Can we identify bad random seeds early on in training and derive practical insights from this?* (the answer is yes! read on)

Tip #2: Start many, stop early, continue some

In practice, resources are often limited — both in terms of time and money. We therefore want to try and get the best model given the constraints imposed on us. If we could identify episodes which lead to bad final models early on during the training process, we could stop them and spend our resources on more promising episodes. Luckily the authors demonstrate that we can do exactly that.



Training curves for 20 random initializations per task. Poor initializations can be identified early on. [Source]

The graphs above demonstrate that it is possible to identify bad initializations which will lead to bad models at the end of training. This is especially true for the smaller data sets. For the larger SST dataset this appears to be less obvious from the graph, but in this case there is still a strong correlation between validation performance after 2 epochs and final validation performance.

The question then becomes how to decide when to stop training a model, and how many models to train. The authors use an algorithm that was inspired by an early stopping criterion for hyperparameter search[3] for this purpose. The algorithm takes the following 3 parameters:

- t: the number of models we start training
- f: when to evaluate models, as a fraction of the total number epochs
- p: the number of top performing models to continue training

Running this algorithm takes $(tf + p(1-f))s$ number of steps to complete where s is the number of total epochs (in this case $s=3$). The authors obtain the best results with f in the region of 20–30%. They also run experiments and show the best parameters for different computational budgets. Common trends are:

- t should be substantially larger than p; and
- p should be roughly half the number of models our computational budget would allow us to train fully (for s epochs).

The results are summarised in the graph below. It shows the relative error reduction when using the above algorithm for each of the 4 tasks. Error reduction is relative to not using the above algorithm — that is, just training a certain number of models fully (x-axis) and selecting the best one. As we can see, for any computational budget, the early stopping algorithm leads to a sizeable performance increase.

Relative error reduction when finetuning BERT with the above early stopping algorithm, compared to just training t number of models (x-axis) fully [Source]

The Key Takeaway:

Starting many models, stopping the bad ones early, and proceeding with only a few can lead to better performance overall when finetuning BERT on a constrained budget.

Conclusion

In a resource constrained setting (there is a fixed computational budget) use these 2 tips to make the most out of BERT finetuning:

1. **Evaluate your model multiple times during an epoch; and**
2. **Identify bad initializations early and stop them.**

Resource Constrained Pretraining

If you found this interesting, you might also want to check out the article below, which discusses ways to improve the pretraining of Transformer models like BERT.

This is how to train better transformer models
How to train faster, higher performant transformers
towardsdatascience.com

. . .

[1]: Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, Noah Smith, Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping (2020).

[2]: Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, Noah A. Smith, Show your work: Improved reporting of experimental results. In Proc. of EMNLP (2019).

[3]: Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, Ameet Talwalkar, Hyperband: A novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research (2018).

Machine Learning · Artificial Intelligence · Deep Learning · Tech · Data Science