Transformers with Learnable Activation Functions

Haishuo Fang[1] **Ji-Ung Lee**[1] **Nafise Sadat Moosavi**[1,2] **Iryna Gurevych**[1]

[1]Ubiquitous Knowledge Processing Lab (UKP Lab)
Department of Computer Science and Hessian Center for AI (hessian.AI)
Technical University of Darmstadt
www.ukp.tu-darmstadt.de

[2]Department of Computer Science, The University of Sheffield

Abstract

Activation functions can have a significant impact on reducing the topological complexity of input data and therefore, improving a model's performance. However, the choice of activation functions is seldom discussed or explored in Transformer-based language models. As a common practice, commonly used activation functions like Gaussian Error Linear Unit (GELU) are chosen beforehand and then remain fixed from pre-training to fine-tuning. In this paper, we investigate the impact of activation functions on Transformer-based models by utilizing rational activation functions (RAFs). In contrast to fixed activation functions (FAF), RAFs are capable of learning the optimal activation functions from data. Our experiments show that the RAF-based Transformer model (RAFT) achieves a better performance than its FAF-based counterpart (FAFT). For instance, we find that RAFT outperforms FAFT on the GLUE benchmark by 5.71 points when using only 100 training examples and by 2.05 points on SQuAD with all available data. Analyzing the shapes of the learned RAFs further unveils that they vary across different layers and different tasks; opening a promising way to better analyze and understand large, pre-trained language models.[1]

1 Introduction

Activation functions introduce non-linearity and increase neural networks' representational capacity, and therefore, play an essential role in designing deep learning models (Nwankpa et al., 2018; Sharma et al., 2020; Dubey et al., 2022). Naitzat et al. (2020) explain the importance of activation functions by proposing to consider data as a topology with its own shape. They empirically show that activation functions accelerate the data topology transformation through different layers of a neural network to simplify its complexity and make it linearly separable in the output space. Their experiments show that choosing the right activation function can have a significant impact on the overall performance.

While any activation function can be used with Transformers (Vaswani et al., 2017), their choice is made before pre-training and remains fixed afterwards. Hence, the inductive bias an activation function imposes on the model cannot be adjusted during pre-training or fine-tuning. As many Transformer-based models are pre-trained on a large amount of data, and changing the activation function for or during fine-tuning may negatively impact the performance[2]. Moreover, the simple case of finding the optimal combination of k different activation functions in n different feedforward layers results in k^n possible combinations and becomes intractable; e.g., 531,441 experiments for a 12-layer BERT model and three different activation functions. As a result, most Transformer-based pre-trained models adopt the GELU activation function that has been initially used for the BERT model (Devlin et al., 2019).

To overcome the limitation of using a potentially suboptimal activation function that remains fixed during training, we propose to use a learnable activation function, namely, the rational activation function (RAF, Molina et al. 2020). The RAF is a universal function approximator that can approximate any existing activation function. The advantage of using RAFs over fixed activation functions (FAF) such as ReLU or GELU, is that the model can learn the optimal activation function from the data during (pre)training without the need to consider the choice of activation function as an additional dimension during hyperparameter tuning.[3]

---

[1]Code, models, and datasplits are available on GitHub https://github.com/UKPLab/2022-RAFT.

[2]In our preliminary experiments, the performance of BERT becomes worse on downstream tasks when the activation functions are changed after pre-training.

[3]Liu et al. (2019a) consider different activation functions during Neural Architecture Search (Zoph and Le, 2017), but this becomes quickly infeasible for compute-intensive experi-

To evaluate the effectiveness of RAFs, we pre-train two encoder-only Transformers using RAF and GELU respectively, within an academic budget. In our experiments, we find that:

- The RAF-based Transformer (RAFT) learns different activation functions at different layers after pre-training with shapes that differ from frequently used activation functions.

- During fine-tuning, RAFT outperforms its fixed activation function counterpart (FAFT) on the general language understanding benchmark (GLUE) and the SQuAD machine reading comprehension dataset in various settings.

- After fine-tuning, the learned RAFs of the top layers are more task-specific and change the most, which are corresponding to layer behaviors of Transformers according to prior work (Mosbach et al., 2020; Merchant et al., 2020; Zhou and Srikumar, 2022). This provides new opportunities to analyze language models with respect to their learned activation functions at different layers for different tasks.

- RAFT boosts the performance when combined with a parameter-efficient fine-tuning approach, i.e., BitFit (Ben Zaken et al., 2022), which improves the model performance by 3.08 points in full-data scenario.

2 Related Work

Activation functions. There exists various predefined activation functions such as Sigmoid, Hyperbolic Tangent (Tanh), Rectified Linear Unit (ReLU, Fukushima 1969), and Gaussian Error Linear Unit (GELU, Hendrycks and Gimpel 2016). There are also approaches that leverage automatic search to obtain optimal combinations of several base activation functions in a predefined search space (Ramachandran et al., 2018; Manessi and Rozza, 2018; Sütfeld et al., 2020; Bingham and Miikkulainen, 2022; Bingham et al., 2020). For instance, Ramachandran et al. (2018) discovered the Swish activation function by using this method. Bingham et al. (2020) show that further extending the search space using evolutionary algorithms can also lead to an improvement. Finally, several search-based works investigate how to train a combination of a set of activation functions to better adapt to specific tasks and architectures (Manessi and Rozza,

ments such as pre-training large language models.

2018; Sütfeld et al., 2020; Bingham and Miikkulainen, 2022). One substantial drawback of these search-based methods is that they are computationally expensive. Especially for pre-trained language models where pre-training is costly, it is infeasible to perform a hyperparameter search for selecting the best activation function (even more so their combination). In contrast, the flexibility of rational activation functions (RAFs) allows them to be trained along with the model parameters in an end-to-end fashion (Molina et al., 2020). Therefore, they can learn the optimized activation function from data during training. RAFs have been successfully used in deep reinforcement learning for improving plasticity (Delfosse et al., 2021), cell detection models in biology (Prangemeier et al., 2020), and adapter architectures (Moosavi et al., 2022).

Model	Act. Funct.
BERT (Devlin et al., 2019)	GELU
GPT-1 (Radford et al., 2018)	GELU
RoBERTa (Liu et al., 2019b)	GELU
XLNet (Yang et al., 2019)	GELU
ALBERT (Lan et al., 2020)	GELU
GPT-2* (Radford et al., 2019)	GELU
Megatron-LM (Shoeybi et al., 2019)	GELU
ELECTRA[+] (Clark et al., 2020)	GELU
T5 (Raffel et al., 2020)	ReLU
T5v1.1 (Raffel et al., 2020)	GeGLU
DeBERTa[+] (He et al., 2021)	GELU
BART (Lewis et al., 2020)	GELU
GPT-3* (Brown et al., 2020)	GELU
Jurassic* (Lieber et al., 2021)	GELU
Gopher* (Rae et al., 2021)	GELU
Megatron-Turing NLG* (Smith et al., 2022)	GELU
Chinchilla* (Hoffmann et al., 2022)	GELU
CANINE[+] (Clark et al., 2022)	GELU
LaMBDA (Thoppilan et al., 2022)	GeGLU
OPT (Zhang et al., 2022)	ReLU

Table 1: Activation functions in different NLP Transformer models. Models marked by * do not explicitly state the activation function but refer to GPT-1 as the base architecture ([+] refers to BERT respectively). GeGLU is a variant that combines GELU and GLU.

Frequently used activation functions in NLP. Table 1 shows a list of 20 different language models that have been introduced after BERT. As we see, the vast majority of the works (80%) use the GELU activation function. Moreover, many works even do not explicitly state the used activation function (45%). There are only a few works that investigate the impact of activation functions on pre-trained Transformer models. So et al. (2021) leverage automatic search methods to identify more efficient Transformer architectures. They find that a combi-

nation of squared ReLU used in the feedforward network (FFN) layer and a convolution layer added in self-attention can lead to a substantial boost in performance. Shazeer (2020) replace the FFN in the Transformer with a gated linear unit (GLU, Dauphin et al. 2017) combined with different activation functions and find a higher performance during pre-training as well as on downstream tasks. In our work, we do not change the structure of FFNs and only replace activation functions in them.

Closest to our work is the work by Moosavi et al. (2022) who investigate the use of RAF in adapters (Houlsby et al., 2019); i.e., lightweight layers that are added on top of pre-trained Transformer layers. They propose adaptable adapters that consist of RAFs and learnable switches to select a subset of adapter layers during training. They show that using both RAFs and a fewer number of adapter layers results in considerable performance gains, especially in low-data settings. However, only using RAF instead of ReLU does not result in a considerable gain in their experiments. Furthermore, adapter layers are only added and updated during fine-tuning, as a result using RAF in adapter layers has a limited impact compared to already applying them for pre-training.

In this work, we show that using RAF in Transformer layers brings additional flexibility to the model to learn the optimized activation function for each of its layers during training, and that this additional flexibility benefits both pre-training and fine-tuning steps.

3 RAFT: RAF-based Transformers

We adopt the BERT architecture (Devlin et al., 2019) where all activation functions in feedforward layers $Activation(W_1 X)W_2$ are replaced with rational activation functions (illustrated in Appendix A). The equation of rational activation function $F(x)$ is as below:

$$F(x) = \frac{P(x)}{Q(x)} = \frac{\sum_{j=0}^{m} a_j x^j}{1 + |\sum_{k=0}^{n} b_k x^k|} \quad (1)$$

Where a and b are learnable parameters, and m and n are degrees of $F(x)$, which decide the complexity and fitting ability of rational functions. Following Molina et al. (2020), we use the *safe PAU* formulation that further stabilizes training.

Selecting m and n. Similar to Taylor series, the higher the degrees m and n are, the more precise

is the approximation of rational functions. However, indefinitely increasing the degrees also means adding more complexity and increasing training time. The challenge is to find suitable degrees that leads to rational functions with a strong fitting ability while keeping their complexity as low as possible. As this is still an open question, we set the search space of m and n to $\{4, 5\}$, and evaluate their ability to approximate the GELU function in the range of [-3,3]. Our results show that using $m = 5$ and $n = 4$ perfectly fits the GELU function with a low complexity and thus, are adopted in this work (cf. Figure 5, Appendix B). This matches the findings in previous work (Telgarsky, 2017; Molina et al., 2020; Delfosse et al., 2021) as well. So overall, each rational activation function adds nine parameters, resulting in a total of 108 additional parameters in a 12-layer Transformer model (less than 0.000098% of its original parameters). The weights of $F(x)$ can further be initialized to approximate any existing activation functions. In our experiments, we initialize it with weights that approximate GELU.

4 Pre-training

To evaluate the viability of RAFT, we pre-train two comparable Transformer models from scratch—one using the common fixed GELU activation function (FAFT), and another one using RAFs (RAFT).

Model architecture. For our experiments, we use a frequently considered model configuration and train 12 Transformer encoder layers with a hidden size of 768 and 12 attention heads (Devlin et al., 2019; Liu et al., 2019b; Rae et al., 2021; Zhang et al., 2022). The only difference between RAFT and FAFT is the use of RAFs instead of GELUs as activation functions.

Data. We use English Wikipedia as our pre-training data.[4] The dataset consists of 3.8×10^9 tokens from which we select 50k sentences containing 6.4×10^6 tokens as the validation data.

Pre-training objective. Following RoBERTa (Liu et al., 2019b), we use dynamic masked language modeling (MLM) as our learning task and randomly mask tokens in the input sentences at each step before feeding them into the model. We use the same masking probabilities and mask 15% of the tokens with an 80% chance of replacing them

---

[4] https://dumps.wikimedia.org

Model	Validation loss	Validation PPL
FAFT	1.645	5.18
RAFT	**1.611**	**5.00**

Table 2: Performance of the models on the validation set after pre-training.

with the [MASK] token, a 10% chance of replacing them with a randomly selected different token, and a 10% chance of not replacing them at all.

Training parameters. As our primary goal is to validate the effectiveness of RAFs in Transformers rather than releasing a RoBERTa-like model, we focus on training two comparable models within a limited training budget. Both models are optimized using AdamW (Loshchilov and Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and a weight decay of 0.01. The learning rate lr_θ is set to 7E-4 for both models while the learning rate lr_{RAF} for the RAF coefficients is set to 5E-3. Both learning rates are warmed up over the first 1% steps, then lr_θ decays linearly while lr_{RAF} remains constant.[5] The batch size is set to 4096. Tuning hyperparameters during pre-training is expensive, to conduct hyperparameters tuning of both models with limited resources, we follow up 24hour BERT (Izsak et al., 2021) to pre-train the model for 23k steps equipped with various methods to accelerate training, including mixed-precision, sparse output prediction, fused linear layer, and tied embeddings (Press and Wolf, 2017). Detailed parameters and results of hyperparameter tuning are provided in Appendix C. It takes ~16 hours for RAFT and ~12 hours for FAFT using four A100 GPUs.

Results. Table 2 shows the MLM validation losses and validation perplexity of the best performing hyperparameter configuration for RAFT and FAFT. We observe that RAFT achieves a bit lower perplexity than FAFT during pre-training. The learned RAFs vary across different layers after pre-training (cf. Figure 6, Appendix E). More analysis is conducted in Section 6.

5 Fine-tuning

We conduct experiments on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019) and SQuAD (Rajpurkar

et al., 2016) to see how well pre-trained RAFs can adapt to downstream tasks. Dataset descriptions are provided in Appendix D. We further investigate the flexibility of the pre-trained RAFs by considering different training data sizes especially in a low-data regime. We fine-tune RAFT in two different settings:

- RAFT^{full}: We fine-tune the whole model, i.e., all model parameters including the RAFs.

- RAFT^{fixed}: We fix the pre-trained RAFs and only tune the rest of the parameters.

5.1 Evaluation on the GLUE Benchmark

We evaluate pre-trained models on GLUE benchmark in different data settings: (a) the full-data scenario, and (b) two low-data scenarios when only 100 or 300 labelled examples are available.

Experimental Setup. We split 75% of the training dataset as the training set and use the remaining 25% as the development set in the full-data scenario. Following previous works, we use the provided development set as the test dataset. For our low-data scenarios, we randomly sample 100 or 300 examples with ten different random seeds and report the average and standard deviation across all runs. For the full-data scenario, we report the average and standard deviation of the results across six runs with different random seeds. We use the same evaluation metrics as proposed in the GLUE benchmark; more specifically, for MRPC, QQP, and STSB, we use the average of the two corresponding metrics as the final score.[6]

Results. Table 3 shows the performance of RAFT and FAFT on the GLUE benchmark. We observe that on average, RAFT achieves consistent improvements in all data settings. We further find that especially in the low-data scenarios, the flexible activation functions of RAFT substantially outperform their static GLUE counterparts of the FAFT model. For 100 examples, RAFT achieves better results in seven out of eight tasks, outperforming FAFT by 5.31 points (RAFT^{full}) and 5.71 points (RAFT^{fixed}) on average, respectively. While the performance gap becomes smaller as the number of examples increases, the tendency remains the same with an average performance gain of 0.98

---

[5]We find in our preliminary experiments that a constant rational learning rate with warm up leads to better results.

[6]Note that the full-data scenario is computationally more expensive to run, but also more stable as the training instances experience less variability.

Model	ColA	SST2	MRPC	QQP	STSB	MNLI-matched/mismatched	QNLI	RTE	Avg.
low-data 100 examples[1]									
FAFT	1.88±2.27	71.02±5.61	74.88±0.23	55.19±5.96	57.57±8.32	32.86±1.50/32.92±1.46	53.34±3.24	**53.14±1.67**	48.07
RAFT^{full}	4.38±3.2	**73.28±3.95**	**75.89±1.39**	**62.65±2.86**	70.30±3.44	38.31±1.87/39.06±2.35	**63.58±3.74**	53.0±1.91	53.38
RAFT^{fixed}	**7.25±4.77**	72.04±5.04	75.76±0.65	62.15±4.09	**71.39±3.56**	**39.3±1.60/40.4±1.73**	63.13±3.05	52.6±2.99	**53.78**
low-data 300 examples[1]									
FAFT	13.12±5.29	77.67±3.07	**79.37±1.56**	66.63±1.35	76.70±1.89	43.74±2.20/45.33,2.29	69.17±2.25	55.45±2.66	58.58
RAFT^{full}	12.36±5.07	78.22±2.10	77.84±1.09	**68.25±1.01**	79.77±2.34	**45.70±1.69/47.27±1.86**	71.92±1.10	54.70±2.26	59.56
RAFT^{fixed}	**17.34±3.23**	**78.95±2.33**	76.97±0.96	68.20±0.76	**80.32±0.1**	45.35±1.62/46.53±1.63	**72.07±1.56**	**55.78±2.72**	**60.17**
Full data[2]									
FAFT	43.18±1.52	89.2±0.63	86.42±1.37	88.08±0.08	**87.08±0.21**	80.92±0.21/81.78±0.22	**89.42±0.38**	62.22±1.35	78.70
RAFT^{full}	**45.84±1.47**	89.85±0.45	**87.21±0.54**	**88.27±0.10**	86.96±0.29	80.88±0.22/81.85±0.23	89.32±0.20	**64.44±2.49**	**79.40**
RAFT^{fixed}	45.66±1.55	**90.06±0.70**	86.36±1.03	88.21±0.06	86.64±0.24	**81.10±0.22/82.06±0.21**	89.36±0.34	63.90±2.85	79.28

[1] Results are averaged over ten random seeds: 5309, 202206, 20220602, 2259, 49, 2022, 1046, 622, 320, 53
[2] Results are averaged over six random seeds: 5309, 202206, 20220602, 2259, 49, 2022

Table 3: The performance of RAFT and FAFT on the GLUE benchmark across different data sizes. RAFT^{full} fine-tunes all model parameters including RAFs. RAFT^{fixed} instead fixes the RAFs pre-training.

points (RAFT^{full}) and 1.59 points (RAFT^{fixed}) for 300 examples. In the full data scenario, RAFT still outperforms FAFT by 0.7 (RAFT^{full}) and 0.58 (RAFT^{fixed}) points on average.

Our experiments indicate that fixing the RAFs is a better choice for the GLUE benchmark in the low-data scenarios. We conjecture that one reason for this may be that the number of instances to tune all parameters of the model are insufficient. On the contrary, we find that in the full-data scenario tuning RAFs can lead to better results. The increasing number of instances especially benefit RAFs as they can better adapt to different downstream tasks and learn better features. We provide further analysis in Section 6.

5.2 Evaluation on SQuAD

Similar to GLUE, we evaluate models on SQuAD v1.1 in different data settings: (a) the full-data scenario, and (b) four low-data scenarios with 100, 300, 500, and 1000 training examples.

Experimental Setup. We split the official training data into separate training (75%) and development sets (25%)[7] and use the official development set as the test data. We evaluate the results by computing the F1 score over the word overlap of the predicted answer and the gold answer. The hyperparameters search space is provided in Appendix C.

Results. Table 4 shows our results of RAFT and FAFT. Compared to GLUE, that consists of sentence-level text matching tasks, SQuAD is a more complex task in which the model needs to comprehend a longer text sequence to predict an answer span. The increased task difficulty is especially reflected in the low-data scenarios, as the

---

[7] Again, we use the development set to identify the best performing model across all epochs.

	100 examples[1]	300 examples[1]	500 examples[1]	1000 examples[1]	full data[2]
FAFT	**12.72±1.54**	**22.11±2.46**	26.46±1.42	34.58±1.68	72.33±0.23
RAFT^{full}	11.81±0.95	19.49±2.01	**26.68±1.91**	**36.69±1.56**	**74.45±0.47**
RAFT^{fixed}	12.19±1.08	19.00±2.68	26.27±1.39	35.98±1.81	74.38±0.25

[1] Results are averaged over ten random seeds: 5309, 202206, 20220602, 2259, 49, 2022, 1046, 622, 320, 53
[2] Results are averaged over six random seeds: 5309, 202206, 20220602, 2259, 49, 2022

Table 4: Results of RAFTs and FAFT on SQuAD.

	Validation Loss	Validation PPL.
Identity	Divergent	Divergent
RELU	1.626	5.08
GELU	**1.611**	**5.00**

Table 5: Different initializations of RAF.

performances of both models are below 25 points when only 100 or 300 annotated examples are available. As a result, when there are not enough annotated examples available to learn the task, the use of RAFs instead of GELU is not beneficial for the Transformer model. However, we again see that RAFT outperforms the FAFT model as enough training examples become available.

In addition, we observe that tuning RAFs during fine-tuning (RAFT^{full}) is more beneficial compared to fixing RAFs (RAFT^{fixed}) when the task is more complex. Considering our findings on the GLUE benchmark, we conjecture that the task difficulty may play an additional role besides the amount of available training data for the performance of RAFT^{full} vs. RAFT^{fixed}; however, this remains to be investigated in future work.

6 Analysis

Impact of RAF initialization. To investigate how initialization affects the performance of RAFT, we train RAFT models initialized with GELU, RELU, and the identity function. Other hyperparameters are the same as those in section 4. Table 5 shows the performance of different initialization

	SNLI	Trivia QA	
		verified-web	verified-wiki
FAFT	74.22±0.19	24.62±1.48	21.01±0.75
RAFT[full]	**74.80±0.29**	**25.40±1.84**	21.50±0.76
RAFT[fixed]	74.76±0.25	**25.40±1.25**	**21.78±0.87**

Table 6: Zero-shot performance of FAFT and RAFT. Models evaluated on SNLI are trained on MNLI. Results on TriviaQA are based on models trained on SQuAD.
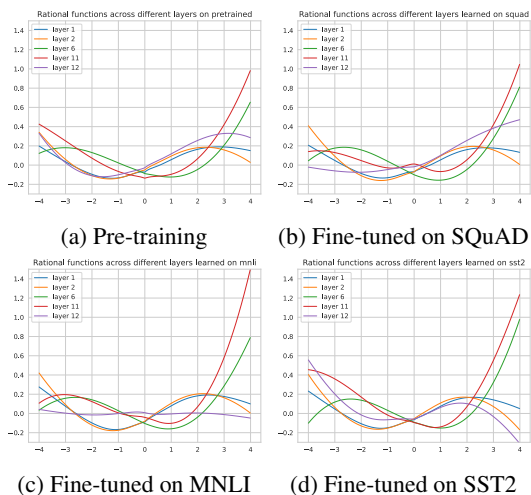

(a) Pre-training (b) Fine-tuned on SQuAD

(c) Fine-tuned on MNLI (d) Fine-tuned on SST2

Figure 1: Rational activation functions of RAFT[full] among different layers after pre-training and fine-tuning.

methods during pre-training. As we can see, choosing common activation functions such as ReLU or GELU leads to a similar performance while using the identity function for initialization leads to divergence.

Zero-shot generalization. To investigate if the higher performances of RAFT vs FAFT come from overfitting on the in-domain data, we conduct cross-domain zero-shot experiments. We use the models that have been fine-tuned on MNLI and SQuAD in the full-data scenario and evaluate them on the same tasks but for different data, namely, SNLI (Bowman et al., 2015) and TriviaQA (Joshi et al., 2017), respectively. MNLI and SNLI are both datasets that aim to evaluate natural language inference while SQuAD and TriviaQA contain examples for evaluating reading comprehension in different domains. Table 6 shows the results of our zero-shot evaluation. We observe that the increased flexibility and adaptivity of RAFT does not negatively impact its generalization capabilities. In fact, both variants of RAFT consistently achieve better performance than the corresponding FAFT model.

Visualizing learned RAFs. Next, we analyze how the shapes of RAFs change after pre-training and fine-tuning. First, we analyze the learned RAFs in different layers of RAFT after pre-training. As shown in Figure 1a, rational functions have different shapes across different layers, none of which are similar to GELU, or other commonly used activation functions in Transformers (cf. Table 1). This indicates that different layers may need different activation functions to achieve the optimal performance. Moreover, we see that some features like monotonicity that often are deemed to be good for predefined activation functions are not necessary, which is in line with the findings of the Swish activation function (Ramachandran et al., 2018).

Second, we analyze how the learned RAFs during pre-training change after fine-tuning in RAFT[full]. Figures 1b–1d show learned RAFs after fine-tuning RAFT[full] on SQuAD, MNLI and SST2 datasets. We observe that some of the learned RAFs trained on these three tasks differ from each other and the RAFs after pre-training. We further see that several RAFs between both tasks have similar shapes but different slopes across many layers.

To better understand the behavior of learned RAFs after fine-tuning in different layers on various tasks, we plot RAFs from the same layer together across all tasks. Figure 2 shows the learned RAFs in layer 1 (the bottom layer), layer 6, and layer 12 (the top layer) after pre-training and fine-tuning on different tasks. We observe that after fine-tuning, the RAFs in the top layer are more task-specific and change the most, compared to those in bottom layers. This is in line with prior work that analyzed the behavior of BERT layers during fine-tuning, which showed that higher layers exhibit more changes compared to lower layers (Mosbach et al., 2020; Merchant et al., 2020; Zhou and Srikumar, 2022). Our results confirm this finding from the perspective of learned activation functions. It also demonstrates that RAFs can self-adapt to different layers and tasks during fine-tuning. In addition, an interesting observation is that the output ranges of the RAFs of MNLI and QQP in the top layer are very close to zero. The output of the FFN layer $Layernorm(\text{FFN}(x) + x)$ consists of two parts: the feedforward branch $\text{FFN}(x)$ and the skip connection branch x. The very small output of activation functions may indicate that the FFN branch of the top layer does not contribute much to the final model performance on MNLI and QQP and
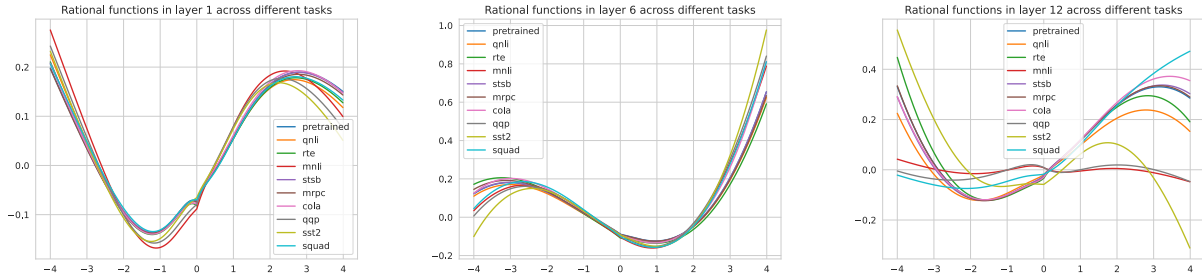
Figure 2: Learned rational activation functions of RAFT$^{\text{full}}$ in layers 1 (bottom), 6, and 12 (top) among different tasks.

Model	ColA	SST2	MRPC	QQP	STSB	MNLI-matched/mismatched	QNLI	RTE	Avg.
low data 100 examples[1]									
$BitFit_{\text{FAFT}}$	1.44±2.85	63.33±9.63	68.82±1.74	55.49±3.94	46.04±24.69	32.92±1.33/32.95±1.24	51.95±3.50	**52.20±2.82**	45.02
$BitFit_{\text{full}}$	4.39±3.41	**76.49±1.90**	74.11±1.04	**61.53±3.09**	50.41±20.20	**33.75±1.38**/33.81±1.30	**57.22±6.15**	50.83±2.74	**49.17**
$BitFit_{\text{fixed}}$	**6.25±3.68**	75.96±1.24	**74.71±0.34**	61.35±3.42	49.91±26.88	33.73±1.40/**34.04±1.71**	53.19±4.02	51.63±2.26	48.97
Full data[1]									
$BitFit_{\text{FAFT}}$	37.75±1.26	87.80±0.67	82.94±1.20	**81.35±0.13**	59.29±33.04	**71.94±0.38**/73.57±0.38	**85.38±1.07**	55.89±1.70	70.66
$BitFit_{\text{full}}$	38.46±1.37	88.19±0.16	**86.73±1.00**	81.03±0.12	85.28±0.33	70.23±0.41/72.53±0.33	80.51±10.75	**60.72±1.88**	**73.74**
$BitFit_{\text{fixed}}$	**39.96±1.95**	**88.46±0.28**	84.91±5.10	81.02±0.14	**85.55±0.44**	71.25±0.19/73.26±0.36	77.23±14.23	60.15±0.90	73.53

[1] Results are averaged over five random seeds: 5309, 202206, 20220602, 2259, 49

Table 7: Comparison between RAFT and FAFT combined with BitFit.

thus could be pruned. We leave this as future work.

RAFT$^{\text{fixed}}$ vs. RAFT$^{\text{full}}$. In our experiments on GLUE and SQuAD (Tables 3 and 4), we observe that fixing the RAFs after fine-tuning (RAFT$^{\text{fixed}}$) often achieves the best or second best performance compared to the full-tuning model (RAFT$^{\text{full}}$) and FAFT. Fine-tuning RAFs results in higher performances when (a) more data is available, i.e., the full-data scenario in GLUE, or (b) the input task is more complex such as in SQuAD. We hypothesize that training RAFs during fine-tuning will be more effective when evaluated on more complex tasks and datasets than the ones used this work.

Efficiency comparison between RAFT and FAFT. In RAFT, RAFs are polynomial ratios and their coefficients are learned during training, which adds extra computation overhead. We use RAFs library with CUDA extension to accelerate. As shown in Table 8, RAFT is slower than FAFT during training since RAFs need to be updated (36.8% slower at pre-training, 14.8% slower at fine-tuning). However, RAFT is faster when doing inference due to the CUDA implementation (13.8% faster at pre-training, 3.9% faster at fine-tuning).

Parameter-efficient fine-tuning with RAFTs. In contrast to fine-tuning all parameters in a pre-trained language model, parameter-efficient tuning techniques that freeze the majority of

steps/second	Pre-training		Fine-tuning	
	Train	Inference	Train	Inference
RAFT	0.38	**3.3**	12.54	**71.05**
FAFT	**0.52**	2.9	**14.4**	68.38

Table 8: Number of steps per second for training and inference for RAFT and FAFT.

pre-trained parameters and only fine-tune a small set can be promising alternatives (Ding et al., 2022). One such method is BitFit (Ben Zaken et al., 2022) which only updates the bias terms in the Transformer model. To investigate the effectiveness of RAFT in a parameter-efficient fine-tuning paradigm, we fine-tune the FAFT and RAFT models with BitFit on the GLUE benchmark. We use the same settings as in our previous experiments and test RAFT and FAFT in three configurations in the low-data 100 and full-data scenario: (a) $BitFit_{\text{FAFT}}$ uses BitFit with FAFT, (b) $BitFit_{\text{full}}$ uses BitFit with RAFT$^{\text{full}}$, and (c) $BitFit_{\text{fixed}}$ uses BitFit with RAFT$^{\text{fixed}}$. As shown in Table 7, RAFT-based BitFit achieves higher performance than the FAFT on average in both data settings: $BitFit_{\text{fixed}}$ achieves 3.95 points improvements and $BitFit_{\text{full}}$ gets 4.15 points improvements in the low-data scenario while $BitFit_{\text{fixed}}$ performs better with a 2.87 points boost and $BitFit_{\text{full}}$ performs better with a 3.08
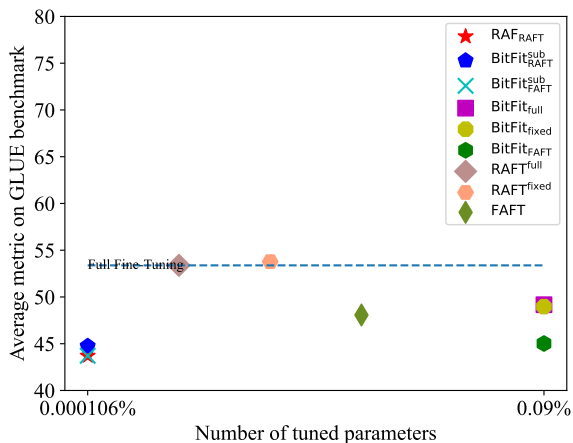
(a) Comparison performance in low-data 100 scenario
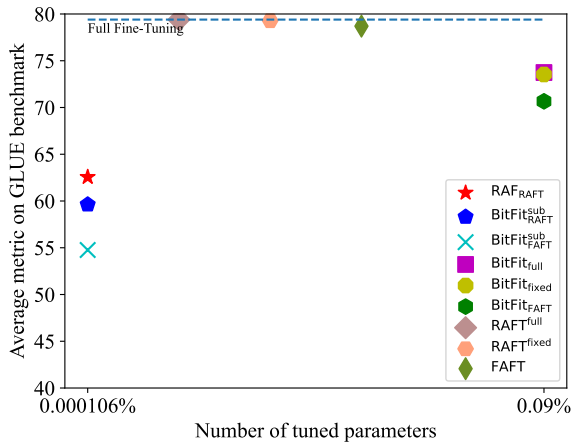
(b) Comparison performance in full-data scenario

Figure 3: The number of parameters vs. the performance for fine-tuning of RAFT and FAFT.

points boost in the full-data scenario. It is worth noting that in some tasks, the reported results have a very large standard deviation (e.g., 33.04 for $BitFit_{\text{FAFT}}$ on STSB) due to several random seed runs not converging. In our experiments, BitFit is not as stable as fine-tuning the whole model.

How much can we achieve by only fine-tuning RAFs? To see to what extent the model can learn from different tasks by only updating RAFs, we conduct experiments to only tune RAFs on the GLUE benchmark in low- and full-data settings. We call this setup where only 117[8] parameters of the RAFs are updated during fine-tuning, RAF_{RAFT}.

For comparison, we tune our models with the BitFit setting using the same amount of parameters,

i.e., 117.[9] $BitFit_{\text{FAFT}}^{\text{sub}}$ represents tuning the subset of BitFit of FAFT, and $BitFit_{\text{RAFT}}^{\text{sub}}$ represents tuning the subset of BitFit of RAFT. The result is presented in Appendix F (Table 13). To compare it from a broader view, we plot Figure 3 based on Table 3, Table 7 and Table 13. We observe that if only a few annotated examples are available (100 examples), $BitFit_{fixed}$ and $BitFit_{full}$ can achieve better performance than full fine-tuning of FAFT. Only fine-tuning 117 parameters ($BitFit_{\text{FAFT}}^{\text{sub}}$, $BitFit_{\text{RAFT}}^{\text{sub}}$ and RAF_{RAFT}) —i.e., a negligible number of parameters compared to 110M parameters in FAFT—results in a comparable performance as fine-tuning all the parameters with only a drop of 4.21–6.68 percentage points. In the full-data scenario, the performance of BitFit ($BitFit_{full}$, $BitFit_{fixed}$ and $BitFit_{FAFT}$) lags behind full fine-tuning of both models. Only tuning RAFs or a subset of BitFit cannot achieve comparable results as well. However, RAF_{RAFT} outperforms $BitFit_{\text{FAFT}}^{\text{sub}}$ by 7.8% and performs better than $BitFit_{\text{RAFT}}^{\text{sub}}$ by 2.94% in this setting.

7 Conclusion and Future Work

In this work, we propose to utilize rational activation functions (RAF) in Transformers to directly learn optimal activation functions from data during pre-training and fine-tuning. To evaluate the effectiveness of rational activation functions, we pre-trained a Transformer-based language model, namely, RAFT. RAFT achieves a lower validation perplexity than FAFT during pre-training. Our experimental results show that RAFT performs better than FAFT in general language understanding tasks and reading comprehension tasks across different data size scenarios. We further visualize and analyze rational activation functions across different layers and tasks after pre-training and fine-tuning and find that they can substantially vary across different layers and tasks. This provides us a new way to analyze and better understand Transformer-based language models. For instance, we can investigate whether layers with similar rational activation functions encode similar linguistic properties. We further find that some layers exhibit a close to zero throughput of the rational activation function which indicates that the corresponding feedforward layer does not contribute too much to a model's prediction. We consider these as our future work.

---

[8]Including RAF in the pooling layer for classification

[9]Note that we also update the classification head in all models and experiments.

Limitations

Limited training resources. This work evaluates the effectiveness of rational activation Transformers using limited GPU resources. To provide a fair comparison, we train and release RAF- and GELU-based models for a reduced GPU budget; hence, they are not comparable to publicly available large pre-trained models such as RoBERTa-base etc. Still, a fully pre-trained RAFT could be released once more GPU resources are available. We furthermore note that we use GELU activation functions and the original FFN architecture as our baseline as it is dominantly used in existing models.

Societal impact. The main focus of this work is the evaluation of trainable activation functions. While our visualization of the learned activation functions show that they exhibit substantial differences depending on the downstream task, further analysis is necessary to better understand and interpret the shapes. Moreover, it is unclear if the additional flexibility of the models may increase their susceptibility towards capturing biases in the data. At the same time, we conjecture that especially susceptible models could also be used as good indicators to detect such biases.

Acknowledgements

References

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2:*

Short Papers), pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Garrett Bingham, William Macke, and Risto Miikkulainen. 2020. Evolutionary optimization of deep learning activation functions. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 289–296, New York, NY, USA. Association for Computing Machinery.

Garrett Bingham and Risto Miikkulainen. 2022. Discovering parametric activation functions. *Neural Networks*, 148:48–65.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 632–642. The Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Object Classification and Recognizing Textual Entailment, First*

PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers, volume 3944 of *Lecture Notes in Computer Science*, pages 177–190. Springer.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 933–941. PMLR.

Quentin Delfosse, Patrick Schramowski, Alejandro Molina, and Kristian Kersting. 2021. Recurrent rational networks. *CoRR*, abs/2102.09407.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *CoRR*, abs/2203.06904.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing, IWP@IJCNLP 2005, Jeju Island, Korea, October 2005, 2005*. Asian Federation of Natural Language Processing.

Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. 2022. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108.

Kunihiko Fukushima. 1969. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Trans. Syst. Sci. Cybern.*, 5(4):322–333.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks,

Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *CoRR*, abs/2203.15556.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Peter Izsak, Moshe Berchansky, and Omer Levy. 2021. How to train BERT with an academic budget. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10644–10652, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. 2021. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019a. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis,

Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Franco Manessi and Alessandro Rozza. 2018. Learning combinations of activation functions. In *24th International Conference on Pattern Recognition, ICPR 2018, Beijing, China, August 20-24, 2018*, pages 61–66. IEEE Computer Society.

Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. 2020. What happens to BERT embeddings during fine-tuning? In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 33–44, Online. Association for Computational Linguistics.

Alejandro Molina, Patrick Schramowski, and Kristian Kersting. 2020. Padé activation units: End-to-end learning of flexible activation functions in deep networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Nafise Moosavi, Quentin Delfosse, Kristian Kersting, and Iryna Gurevych. 2022. Adaptable adapters. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3742–3753, Seattle, United States. Association for Computational Linguistics.

Marius Mosbach, Anna Khokhlova, Michael A. Hedderich, and Dietrich Klakow. 2020. On the interplay between fine-tuning and sentence-level probing for linguistic knowledge in pre-trained transformers. In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 68–82, Online. Association for Computational Linguistics.

Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. 2020. Topology of deep neural networks. *J. Mach. Learn. Res.*, 21:184:1–184:40.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2018. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378.

Tim Prangemeier, Christoph Reich, and Heinz Koeppl. 2020. Attention-based transformers for instance segmentation of cells in microstructures. In *IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2020, Virtual Event, South Korea, December 16-19, 2020*, pages 700–707. IEEE.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *CoRR*, abs/2112.11446.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2018. Searching for activation functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net.

Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. 2020. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 04:310–316.

Noam Shazeer. 2020. GLU variants improve transformer. *CoRR*, abs/2002.05202.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zheng, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. 2022. Using deepspeed and megatron to train megatron-turing NLG 530b, A large-scale generative language model. *CoRR*, abs/2201.11990.

David R. So, Wojciech Manke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. 2021. Searching for efficient transformers for language modeling. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 6010–6022.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642. ACL.

Leon René Sütfeld, Flemming Brieger, Holger Finger, Sonja Füllhase, and Gordon Pipa. 2020. Adaptive blending units: Trainable activation functions for deep neural networks. In *Intelligent Computing - Proceedings of the 2020 Computing Conference, Volume 3*, volume 1230 of *Advances in Intelligent Systems and Computing*, pages 37–50. Springer.

Matus Telgarsky. 2017. Neural networks and rational functions. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3387–3393. PMLR.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *CoRR*, abs/2201.08239.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. 2022. Deepnet: Scaling transformers to 1, 000 layers. *CoRR*, abs/2203.00555.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *CoRR*, abs/2205.01068.

Yichu Zhou and Vivek Srikumar. 2022. A closer look at how fine-tuning changes BERT. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1046–1061, Dublin, Ireland. Association for Computational Linguistics.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

A Model Architecture

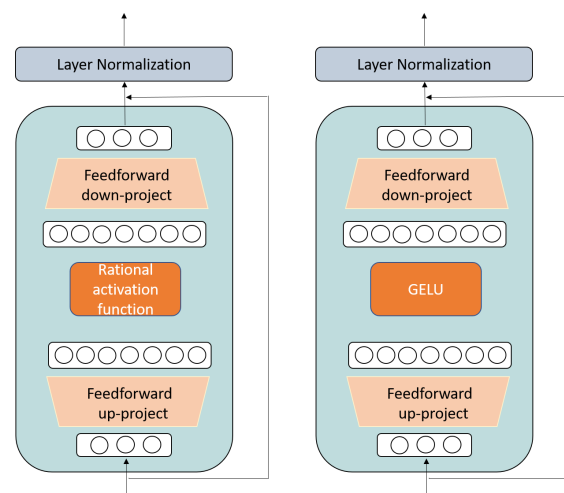Figure 4 shows the difference part of RAFT and FAFT.

Figure 4: Rational activation function in the feedforward layer (left) and the vanilla GELU counterpart (right).

B Fitting abilities of different degrees of Rational Functions

Figure 5 show the approximate functions of GELU using rational functions with different degrees. As we can see, when $m = 5$ and $n = 4$ or $n = 5$, rational function fit GELU very well in the same shape. Finally, it is important to note that rational functions are an universal approximator in a limited range, e.g., [-5,5]. Especially for out-of-bound inputs (i.e., values that are not guaranteed by rational functions), the output of rational functions may result in values very different from the approximated function (e.g., GELU). While pre-training a model from scratch with RAFs does not lead to any problem, directly replacing activation functions in pre-trained models with RAFs only for fine-tuning may lead to divergence due to out-of-bound inputs.

C Hyperparameters Tuning

C.1 Pre-training

In our preliminary experiments that some hyperparameter configurations can lead to instability during training due to diverging model updates (e.g., for lr_θ =7E-4 and batch size of 2048). To stabilize the training without having to rely on a larger warmup phase (e.g., 6% of the training steps), we instead adopt the DeepNorm (Wang et al., 2022) to initialize both models. DeepNorm stabilizes training by bounding the updates and further scaling the residual branches in Transformers. Using DeepNorm makes both models, FAFT and RAFT, achieve lower validation loss and leads to a more stable training.

We tune the learning rate lr_θ for model parameters and lr_{RAF} for RAFs, batch size, warmup steps, and learning rate scheduler as hyperparameters for both models separately. The hyperparameter search space for pre-training stage is as follows:

- Learning rate lr_θ for model parameters: 1E-4, 4E-4, 7E-4, 1E-3

- Learning rate lr_{RAF} for RAFs: 1E-3, 5E-3, 1E-2

- Batch size: 2048, 4096

- Warmup ratio: 0%, 1%, 6%

Some results of hyperparameters tuning are provided in Table 9.

Table 10 shows final hyperparameters we used for pre-training RAFT and FAFT.

	lr_θ	lr_{RAF}	Batch Size	Validation Loss
RAFT	1E-4	0.005	2048	2.217
RAFT	4E-4	0.005	2048	1.808
RAFT	7E-4	0.005	4096	1.732
RAFT	**7E-4**	**0.005**	**4096**	**1.611**
RAFT	1E-3	0.005	4096	1.638

Table 9: Part of Hyperparameters Tuning Results of RAFT

Hyperparameters	FAFT	RAFT
Peak lr_θ	7E-4	7E-4
Peak lr_{RAF}	n/a	5E-3
Learning rate decay	linear	constant
Gradient clipping	0	0
Batch size	4096	4096
Sequence length	128	128
Adam_beta1	0.9	0.9
Adam_beta2	0.98	0.98
Attention dropout	0.1	0.1
Warmup ratio	1%	1%
Training steps	23k	23k

Table 10: Hyperparameters for pre-training RAFT and FAFT

C.2 Fine-tuning

The hyperparameters search space for GLUE during fine-tuning stage is as follows:

- lr_θ: 2E-5, 5E-5

- lr_{RAF}: 1E-4, 5E-4, 1E-3, 5E-3

- Batch size: 32

- Weight decay: 0.1

- Number of epochs: 3, 10, 20

We further tune the learning rates and number of training epochs for RAFT and FAFT separately on a single random seed. For our low-data experiments we fix the number of training epochs to 20 and use early stopping with a patience of 10 epochs. For our full-data experiments, we train the large datasets (QQP, MNLI, and QNLI) for 3 epochs and the others for 10 epochs.

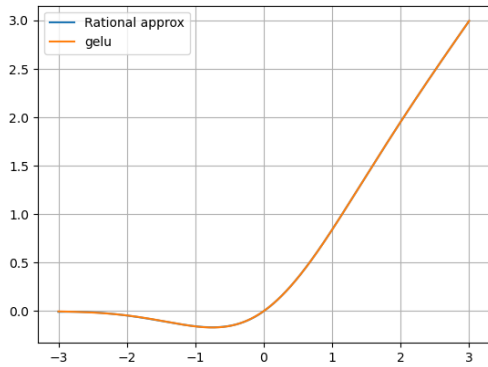The hyperparameters search space for SQuAD during fine-tuning is as below:
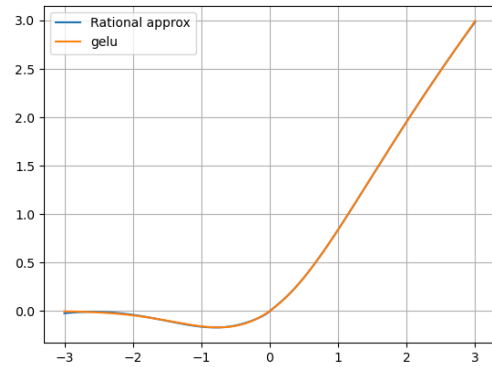
- lr_θ: 2E-5, 5E-5, 1E-4
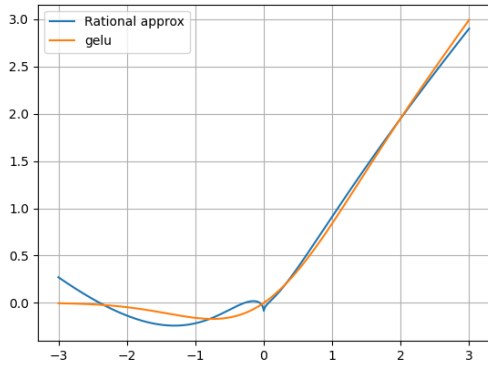
(a) Approximate function with degrees $m = 4$ and $n = 4$
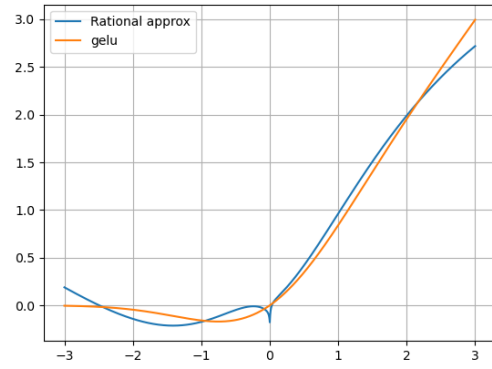
(b) Approximate function with degrees $m = 4$ and $n = 5$

(c) Approximate function with degrees $m = 5$ and $n = 4$ Rational Function is overlapping with GELU

(d) Approximate function with degrees $m = 5$ and $n = 5$ Rational Function is overlapping with GELU

Figure 5: Approximate Functions of GELU using rational functions

- lr_{RAF}: 1E-4, 5E-4, 1E-3, 5E-3

- Batch size: 32

- Weight decay: 0.1

- Number of epochs: 10, 20

For our experiments, we fine-tune both models with their best performing lr_θ =1E-4 for 10 epochs in the full-data scenario and 20 epochs in the low-data scenario.

The hyperparameters search space for BitFit is as below:

- Learning rate lr_θ for model parameters: 5E-5, 1E-3, 5E-3, 1E-2

- Learning rate lr_{RAF} for RAFs: 1E-3, 5E-3, 1E-2

- Batch size: 32

- Training epochs: 3, 10, 20 epochs

We use 3 training epochs for large dataset(QQP, MNLI, QNLI), 10 epochs for other datasets and 20 epochs for low-resource scenarios. Both models can converge in the above settings.

D Data Statistics

GLUE is a collection of nine different language understanding tasks: CoLA (Warstadt et al., 2019), SST2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), QQP [10], STSB (Cer et al., 2017), MNLI (Williams et al., 2018), RTE (Dagan et al., 2005), and WNLI (Levesque et al., 2012). We exclude WNLI due to the adversarial nature of its development set and the still unbeaten majority vote upper bound.[11]

Table 11 show data statistics of GLUE benchmark.

---

[10] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

[11] Cf. (12) in https://gluebenchmark.com/faq

2395

Task	CoLA	SST2	MRPC	QQP	STSB	MNLI-matched/mismatched	QNLI	RTE
|Train|	8,551	67,349	3,668	363,846	5,749	392,702	104,743	2,490
|Dev|	1,043	872	408	40,430	1,500	9,815/9,832	5,463	277
Metric	Matthews corr.	acc.	acc./F1	acc./F1	Person/Spearman corr.	acc.	acc.	acc.

Table 11: Dataset statistics of the GLUE benchmark

SQuAD   is a reading comprehension task where each example consists of a question, a context, and the respective span from the context that answers the question. Table 12 show data statistics of SQuAD.

E   Learned RAFs during pre-training and after fine-tuning

Figure 6 and Figure 7 show learned RAFs in 12 layers after pre-training and fine-tuning on different tasks, respectively.

F   Results of only tuning RAFs

Table 13 shows comparison results between only tuning RAFs and BitFit with the same parameters with RAFT and FAFT.

	|Train|	|Dev|	|Test|
SQuAD v1.1	66,236	21,530	10,789

Table 12: Statistics of SQuAD: the official training dataset is split into training and development sets, and the official development dataset is used as the test data.
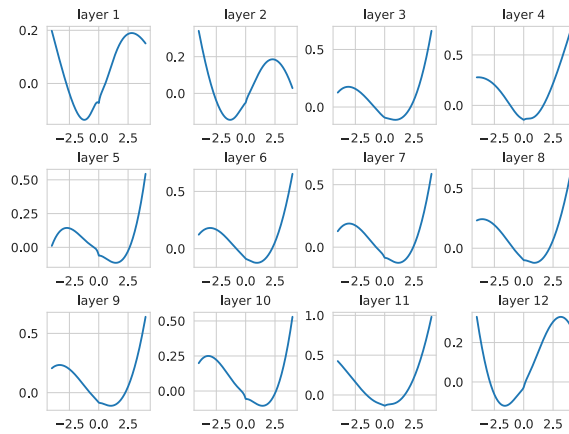

Figure 6: Learned RAFs of different layers after pre-training

Model	ColA	SST2	MRPC	QQP	STSB	MNLI-matched/mismatched	QNLI	RTE	Avg.
low data 100 examples[1]									
$BitFit^{sub}_{FAFT}$	1.49±1.87	62.82±7.56	**74.80±0.00**	52.57±3.83	14.71±7.21	32.73±1.41/32.76±1.30	49.77±0.40	**50.83±1.86**	41.39
$BitFit^{sub}_{RAFT}$	2.45±3.58	72.34±3.41	74.67±0.68	**55.61±2.35**	**23.99±10.41**	**35.32±0.67**/35.66±1.05	51.08±0.71	51.70±1.85	**44.75**
RAF_{RAFT}	**4.33±3.02**	**72.91±2.82**	74.47±0.88	51.92±5.03	17.27±10.60	35.24±0.61/**35.69±0.92**	51.12±0.48	50.47±1.63	43.71
Full data[1]									
$BitFit^{sub}_{FAFT}$	6.61±7.08	79.52±0.52	71.32±0.22	70.48±0.66	37.33±5.70	53.33±1.13/55.30±0.75	64.04±2.03	54.88±1.42	54.76
$BitFit^{sub}_{RAFT}$	8.78±5.54	**82.02±0.57**	71.76±0.77	70.88±1.17	71.40±0.52	51.57±0.54/53.27±1.20	69.87±1.20	**57.04±1.19**	59.62
RAF_{RAFT}	**9.71±12.04**	81.70±0.12	**74.81±3.09**	**73.57±0.48**	**80.79±0.60**	**57.34±0.19/60.69±0.51**	67.89±8.64	56.53±1.83	**62.56**

[1] Results are averaged over five random seeds: 5309, 202206, 20220602, 2259, 49

Table 13: Comparison between fine-tuning RAFs and a subset of 117 BitFit parameters with RAFT and FAFT.
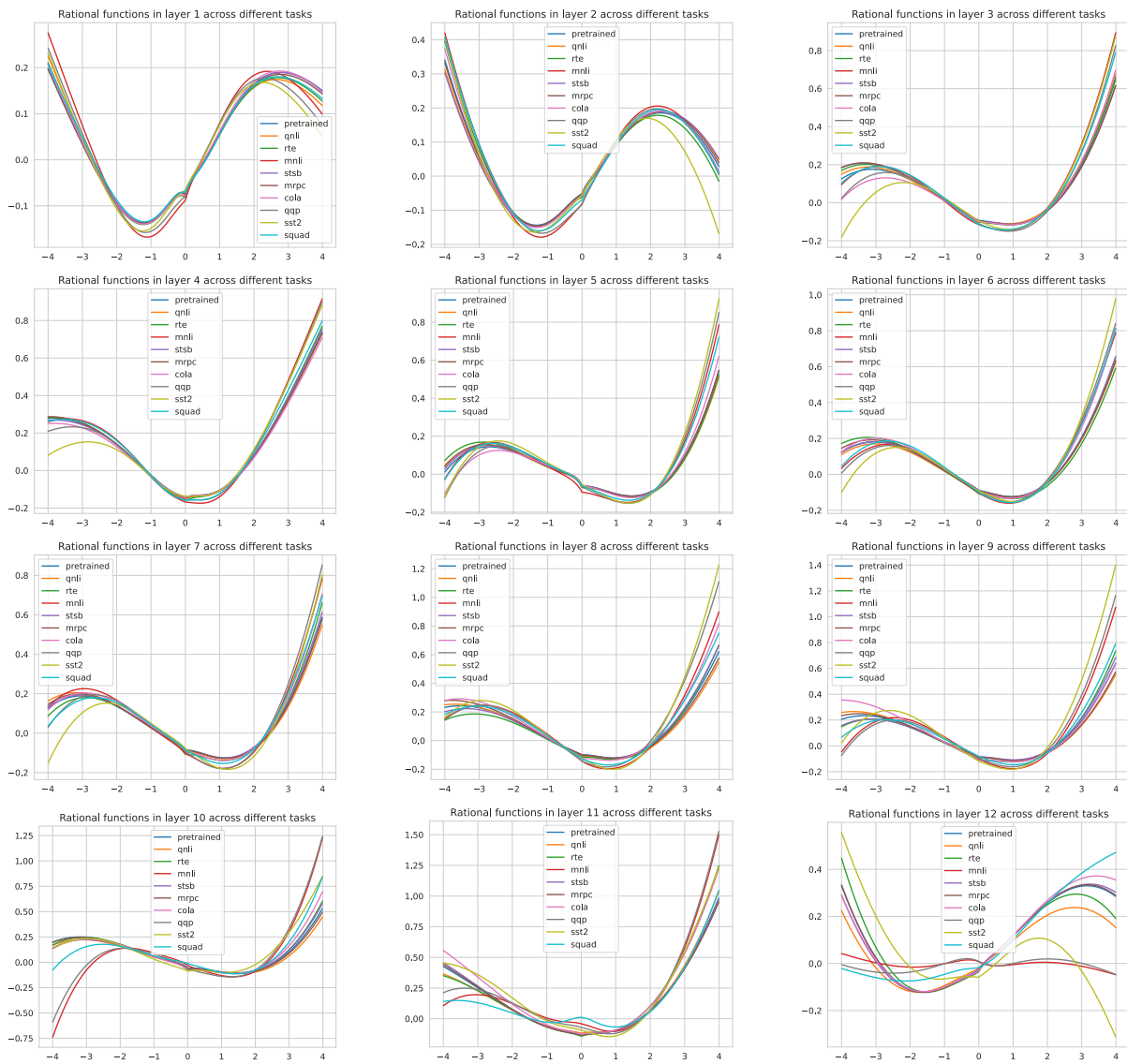
Figure 7: Learned RAFs in 12 layers across different tasks after fine-tuning