Transformers with Learnable Activation Functions

Haishuo Fang[1], Ji-Ung Lee[1], Nafise Sadat Moosavi[2], Iryna Gurevych[1]
[1]UKP Lab, TU Darmstadt, Germany
[2]Department of Computer Science, The University of Sheffield, UK
[1]$<$ lastname $>$@ukp.informatik.tu-darmstadt.de

Abstract

Activation functions can have a significant impact on reducing the topological complexity of input data and therefore improve the performance of the model. Selecting a suitable activation function is an essential step in designing neural models. However, the choice of activation function is seldom discussed or explored in Transformer-based pre-trained language models. As a common practice, their activation functions are chosen beforehand and then remain fixed across the whole life cycle of the models—from pre-training to fine-tuning. As a result, the inductive biases they imposed on models cannot be adjusted during this long life cycle. Moreover, subsequently developed models (e.g., RoBERTa, BART, T5, and GPT-3) often follow up prior work (e.g., BERT) to use the same activation function such as Gaussian Error Linear Unit (GELU) without justification. In this paper, we investigate the effectiveness of using Rational Activation Function (RAF) that is a learnable activation function in the Transformer architecture. In contrast to conventional, predefined activation functions, RAFs can adaptively learn the optimal activation function during training according to the input data. Our experiments show the RAF-based Transformer model (RAFT) achieves a lower validation perplexity compared to a vanilla BERT with the commonly used GELU function. We further evaluate RAFT on downstream tasks in low- and full-data settings. Our results show that RAFT outperforms the counterpart model across the majority of tasks and settings. For instance, RAFT outperforms vanilla BERT on the GLUE benchmark by 5.71 points on average in low-data scenario (where 100 training examples are available) and by 2.05 points on SQuAD in full-data setting. Analysis of the shapes of learned RAFs further unveils that they substantially vary between different layers of the pre-trained model and mostly look very different from conventional activation functions. RAFT opens a new research direction for analyzing and interpreting pre-trained models according to the learned activation functions.[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, Sharma, and Athaiya 2017; Dubey, Singh, and Chaudhuri 2022). Naitzat, Zhitnikov, and Lim (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 it is possible to use any of the existing activation functions in Transformers (Vaswani et al. 2017), the choice of activation functions is however determined and fixed before pre-training. Therefore, their inductive biases imposed on the model cannot be adjusted during pre-training or fine-tuning. It is because (a) commonly used Transformers are pre-trained on a large amount of data, and changing the activation function during fine-tuning will negatively impact the performance[2], and (b) the simple case of selecting the best out of k different activation functions in n different feedforward layers of a Transformer model will result in k^n possible combinations for the hyperparameter optimization step, e.g., 531,441 necessary experiments for a 12-layer BERT model and three different activation functions. As a result, most recent Transformer-based pre-trained models use the GELU activation function that has been initially used for the BERT model (Devlin et al. 2019).

To overcome this limitation of using a predefined and not necessarily optimized activation function in Transformers, we propose to use a learnable activation function, namely rational activation function (Molina, Schramowski, and Kersting 2020), within the Transformer architecture. The rational activation function (RAF) is a universal function approximator that can approximate any existing activation function. The advantage of using RAF over fixed activation functions (e.g., ReLU or GELU) is that the model can learn the optimal activation function from data during (pre)training without the need for hyperparameter optimization. To evaluate the effectiveness of RAFs in Transformers, we pre-train two encoder-only Transformers with RAF and fixed GELU acti-

[1]Code and models will be available on Github under an open source license: `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 only for fine-tuning.

vation functions within an academic budget. In our experiments, we find that:

- RAF-based Transformer (RAFT) learns different activation functions at different layers after pre-training, and they are not necessarily similar to commonly used activation functions.
- During fine-tuning, RAFT outperforms vanilla BERT on the general language understanding benchmark (GLUE) and the SQuAD machine reading comprehension dataset.
- 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).
- RAFT outperforms the vanilla BERT in the majority of evaluated tasks and data settings.
- RAFT boosts the performance when combined with parameter-efficient fine-tuning approaches such as BitFit (Ben Zaken, Goldberg, and Ravfogel 2022).

2 Related Work

Activation functions. There exist various predefined activation functions such as Sigmoid, Hyperbolic Tangent (Tanh), ReLU (Fukushima 1969), Softplus (Dugas et al. 2000), and the 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, Zoph, and Le 2017; Manessi and Rozza 2018; Sütfeld et al. 2020; Bingham and Miikkulainen 2022; Bingham, Macke, and Miikkulainen 2020). For instance, Ramachandran, Zoph, and Le (2017) discovered the Swish activation function by using this method. Bingham, Macke, and Miikkulainen (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 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, Schramowski, and Kersting 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, Reich, and Koeppl 2020), and adapter architectures targeting pre-trained transformer models (Moosavi et al. 2022).

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 GELU activation functions; moreover, many works even do not explicitly state the used activation function anymore (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 combination 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.

Model	Act. Funct.
BERT (Devlin et al. 2019)	GELU
GPT-1 (Radford et al. 2018)	GELU
RoBERTa (Liu et al. 2019)	GELU
XLNet (Yang et al. 2019)	GELU
ALBERT (Lan et al. 2019)	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. 2020)	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: Used activation functions across 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 of GELU that combines GELU and GLU.

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—a commonly used activation function in AdapterHub (Pfeiffer et al. 2020)—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 using them in Trans-

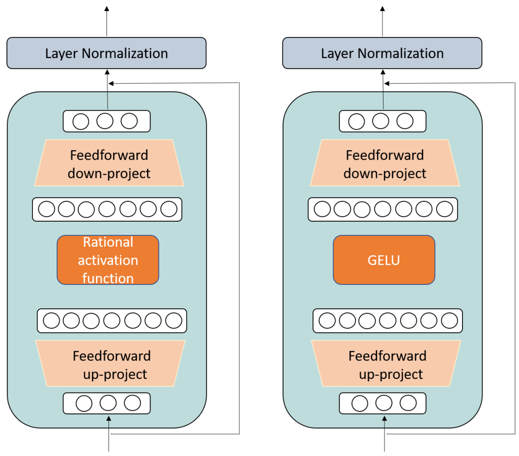


Figure 1: Rational activation function in the feed-forward layer (left) and the vanilla GELU counterpart (right).

former layers.

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 feed-forward layers $Activation(W_1X)W_2$ are replaced with rational activation functions as shown in Figure 1. The equation of rational activation function $F(x)$ is as below:

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

Where a and b are learnable parameters, and m and n are the order of $F(x)$.

Similar to previous work (Telgarsky 2017; Molina, Schramowski, and Kersting 2020; Delfosse et al. 2021), we use order $m = 5$ and $n = 4$ in our model. In this case, each rational activation function only has nine parameters. This only introduces 108 new parameters in a 12-layer Transformer model; i.e., less than 0.000098% of its original parameters. We can initialize $F(x)$ to approximate any of the existing activation functions. In our experiments, we initialize it with weights that approximate GELU.

To evaluate the viability of RAFT, we pre-train two comparable Transformer models from scratch—one using the common fixed GELU activation functions (vanilla BERT), and another one using RAFs (RAFT). We then further fine-tune and evaluate both models on two NLP benchmarks.

4 Pre-training

Training Objective. Following RoBERTa (Liu et al. 2019), we use dynamic masked language modeling (MLM) as our training objective; that means tokens in the input sentences are randomly masked at each step before feeding them into the model (in contrast to the static masked language modeling used in BERT). We keep the masking probabilities of the tokens the same as in RoBERTa and mask 15% of the tokens with an 80% chance of replacing them 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.

Model	Validation loss	Validation PPL
Vanilla BERT	1.645	5.18
RAFT	**1.611**	**5.00**

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

Data. English Wikipedia is used as our pre-training data.[3] 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. Each sentence is split into character- or word-level tokens with Byte-Pair Encoding (Sennrich, Haddow, and Birch 2015). Following Izsak, Berchansky, and Levy (2021), we set the sequence length to 128 tokens throughout the whole pre-training process for a better computational efficiency.

Model Configuration. In all experiments, we use the configuration of the BERT-base model which consists of 12 Transformer encoder layers with a hidden size of 768 and 12 attention heads. The only difference between RAFT and vanilla BERT is the use of RAFs instead of GELUs as activation functions.

Optimization Setup For fair comparison, both models are optimized by AdamW (Loshchilov and Hutter 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and weight decay of 0.01. We use a batch size of 4096 instances. The learning rate lr_θ for model parameters in both models is 7e-4 while the learning rate lr_{RAF} for coefficients of rational functions in RAFT is 5e-3. Both learning rates are warmed up over the first 1% steps, then lr_θ decays linearly while lr_{RAF} remains constant.[4]

Pre-training Setup and Hyperparameter Tuning. Pre-training a high-performanced Transformer-based language model from scratch is prohibitively expensive. To conduct this experiment with an academic budget (Izsak, Berchansky, and Levy 2021), we train the models for 23k steps combined with different acceleration methods, including mixed-precision training, sparse output prediction and tied embeddings (Press and Wolf 2017). The whole pre-training process is conducted on four A100 40GiB GPUs for both models and takes $\sim$ 16 hours for both models. We further find 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

[3] `https://dumps.wikimedia.org`

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

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, vanilla BERT and RAFT, achieve lower validation loss and leads to a more stable training.

We then 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 full hyperparameter search space is reported in the technical appendix. All tuning experiments are conducted using a single, fixed random seed.

Results. Table 2 shows the MLM validation losses and validation perplexity of the best performing hyperparameter configuration for each RAFT and vanilla BERT. We observe that RAFT achieves a lower perplexity than vanilla BERT during pre-training.

5 Fine-tuning

We conduct experiments on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al. 2018) and SQuAD (Rajpurkar et al. 2016) to see how well pre-trained RAFs can adapt to specific downstream tasks. 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$^{\mathrm{full}}$: We fine-tune the whole model, i.e., all model parameters including the RAFs.
- RAFT$^{\mathrm{fixed}}$: We fix the pre-trained RAFs and only tune the other model parameters.

5.1 Evaluation on the GLUE Benchmark

Data. GLUE is a collection of nine different language understanding tasks: CoLA (Warstadt, Singh, and Bowman 2019), SST2 (Socher et al. 2013), MRPC (Dolan and Brockett 2005), QQP (Iyer, Dandekar, and Csernai 2017), STSB (Cer et al. 2017), MNLI (Williams, Nangia, and Bowman 2018), RTE (Dagan, Glickman, and Magnini 2005), and WNLI (Levesque, Davis, and Morgenstern 2012). We exclude WNLI due to the adversarial nature of its development set and the still unbeaten majority vote upper bound.[5] Note that 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.

Experimental Setup. For a comprehensive evaluation of RAFT, we consider two different scenarios: full-data and low-data fine-tuning. In both scenarios, we split 75% of the training dataset as the training set and use the remaining 25% as the development set. Following previous works, we further use the provided development set as the test dataset. For our low-data scenario we consider two different dataset sizes, namely, when only 100 and 300 examples are available. We further 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.[6]

Hyperparameter Tuning. We further tune the learning rates and number of training epochs for RAFT and vanilla BERT 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 hyperparameter search space (including fixed parameters) 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

Results. Table 3 shows the performance of RAFT and vanilla BERT on the GLUE benchmark. We observe that RAFT achieves consistent improvements across different tasks in all data settings. We further find that especially in the low-data scenario, the flexible activation functions of RAFT substantially outperform their static GLUE counterparts of the vanilla BERT model. For 100 examples, RAFT achieves better results in seven out of eight tasks, outperforming vanilla BERT by 5.31 points (RAFT$^{\mathrm{full}}$) and 5.71 points (RAFT$^{\mathrm{fixed}}$) 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 points (RAFT$^{\mathrm{full}}$) and 1.59 points (RAFT$^{\mathrm{fixed}}$) for 300 examples. In the full data scenario, RAFT still outperforms vanilla BERT by 0.7 (RAFT$^{\mathrm{full}}$) and 0.58 (RAFT$^{\mathrm{fixed}}$) 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 the 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

Data. 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 (it is also possible that the provided context contains no answer at all). We evaluate models on SQuAD v1.1 in different data settings: (a) the full-data scenario and (b) the low-data scenario with 100, 300, 500, and 1000 examples. Similar as for

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

[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]									
Vanilla-BERT	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$^{\text{full}}$	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$^{\text{fixed}}$	**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]									
Vanilla-BERT	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$^{\text{full}}$	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$^{\text{fixed}}$	**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]									
Vanilla-BERT	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$^{\text{full}}$	**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$^{\text{fixed}}$	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 vanilla BERT on the GLUE benchmark across different data sizes. RAFT$^{\text{full}}$ fine-tunes all model parameters including RAFs. RAFT$^{\text{fixed}}$ instead fixes the RAFs pre-training.

GLUE, 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.

Experimental Setup. We again tune hyperparameters for both models separately. More specifically, we evaluate $lr_\theta \in$ {2e-5, 5e-5, 1e-4} and $lr_{\mathrm{RAF}} \in$ {1e-4, 5e-4, 1e-3, 5e-3}. 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. We evaluate the results using by computing the F1 score over the word overlap of the predicted answer and the gold answer.

Results. Table 4 shows our results of RAFT and vanilla BERT. 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 (instead of a label). The increased task difficulty is especially reflected in the low-data scenarios, as the 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 vanilla BERT model as more examples become available.

In addition, we observe that tuning RAFs during fine-tuning (RAFT$^{\text{full}}$) is more beneficial compared to fixing RAFs (RAFT$^{\text{fixed}}$) when the task is more complex. Considering our findings on the GLUE benchmark, we conjecture that depending on the task difficulty, there exists different ranges with respect to the available training data where using Transformers with (flexible and fixed) RAFs have an advantage.

6 Analysis

Finally, we provide analysis on RAFT regarding its zero-shot capabilities and the shapes of the learned activation functions after pre-training and fine-tuning. Besides, we evaluate RAFT under parameter-efficient tuning paradigm.

[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]
Vanilla BERT	**12.72±1.54**	**22.11±2.46**	26.46±1.42	34.58±1.68	72.33±0.23
RAFT$^{\text{full}}$	11.81±0.95	19.49±2.01	**26.68±1.91**	**36.69±1.56**	**74.45±0.47**
RAFT$^{\text{fixed}}$	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 vanilla BERT on SQuAD.

	SNLI	Trivia QA	
		verified-web	verified-wiki
Vanilla BERT	74.22±0.19	24.62±1.48	21.01±0.75
RAFT$^{\text{full}}$	**74.80±0.29**	**25.40±1.84**	21.50±0.76
RAFT$^{\text{fixed}}$	74.76±0.25	**25.40±1.25**	**21.78±0.87**

Table 5: Zero-shot performance of vanilla BERT and RAFT. Models evaluated on SNLI are trained on MNLI. Results on TriviaQA are based on models trained on SQuAD.

Zero-shot generalization. To investigate if the higher performances of RAFT vs vanilla BERT come from overfitting on the in-domain data (therefore may reduce generalization), we conduct cross-domain zero-shot experiments. To do so, 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 5 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 outperform the corresponding vanilla BERT model. We therefore conclude that the improved performances of RAFT on GLUE and SQuAD cannot be attributed to overfitting on the in-domain data.

Visualizing learned RAFs. Next, we analyze how the shapes of RAFs change after pre-training and fine-tuning at different layers of the model, and for different tasks. First, we analyze the learned RAFs in different layers of RAFT

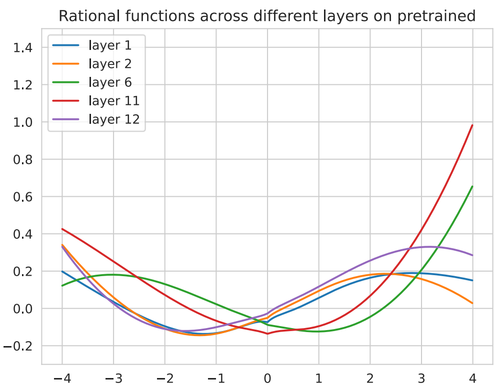


Figure 2: Rational activation functions across different layers after pre-training

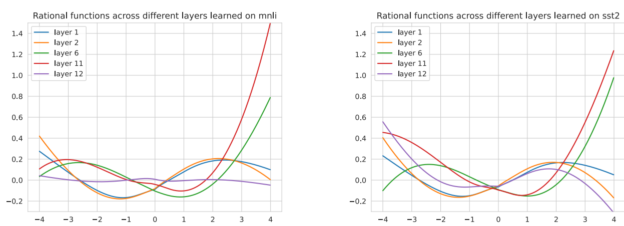


Figure 3: Rational activation functions of RAFT$^{\text{full}}$ among different layers on MNLI and SST2 after fine-tuning

after pre-training to see whether they resemble the common predefined activation functions. Figure 2 shows the plots of learned RAFs across different layers in RAFT after pre-training. As we see, rational functions have different shapes across different layers, none of which are similar to GELU, or other commonly used activation functions in Transformers. This indicates that using the same activation function across all layers may not be beneficial. Moreover, it also shows that some features like monotonicity that 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, Zoph, and Le 2017).

Second, we analyze how the learned RAFs during pre-training change after fine-tuning in RAFT$^{\text{full}}$. Figure 3 shows the learned RAFs after fine-tuning RAFT$^{\text{full}}$ on MNLI and SST2 datasets. We observe that some of the learned RAFs trained on these two tasks differ from each other and the RAFs after pre-training (cf. Figure 2). 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 4 shows the learned RAFs in layer 1 (the first 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 $Layernorm(\mathrm{FFN}(x) + \mathrm{ID}(x))$ consists of two parts: the feedforward branch $\mathrm{FFN}(x)$ and the skip connection branch where $\mathrm{ID}(x)$ is the identity function. 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 when coupled with the skip connection branch. For future work, we will investigate if there are other layers that exhibit a similar shape and if such layers can be pruned with only a minimal loss of performance.

Overall, we find that RAFT provides a new opportunity to analyze and interpret pre-trained language models from the perspective of learned activation functions. It is also worth investigating the similarities and dissimilarities of layers with similar learned activation functions in terms of the learned embedding space or linguistic properties.

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 vanilla BERT. 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. However, the performance gain is minor compared to RAFT$^{\text{fixed}}$ in our experiments. 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.

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 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, Goldberg, and Ravfogel 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 vanilla BERT and RAFT models with BitFit on the GLUE benchmark. We use the same settings as in our previous experiments and test RAFT and vanilla BERT in three configurations in the low-data 100 and full-data scenario: (a) $BitFit_{\mathrm{BERT}}$ uses BitFit with vanilla BERT, (b) $BitFit_{\mathrm{full}}$ uses BitFit with RAFT$^{\text{full}}$, and (c) $BitFit_{\mathrm{fixed}}$ uses BitFit with RAFT$^{\text{fixed}}$. As shown in Table 6, RAFT-based BitFit achieves higher performance than the vanilla BERT on average in both data settings: $BitFit_{\mathrm{fixed}}$ achieves 3.95 points improvements and $BitFit_{\mathrm{full}}$ gets 4.15 points improvements in the low-data scenario while $BitFit_{\mathrm{fixed}}$ performs better with a 2.87 points boost and $BitFit_{\mathrm{full}}$

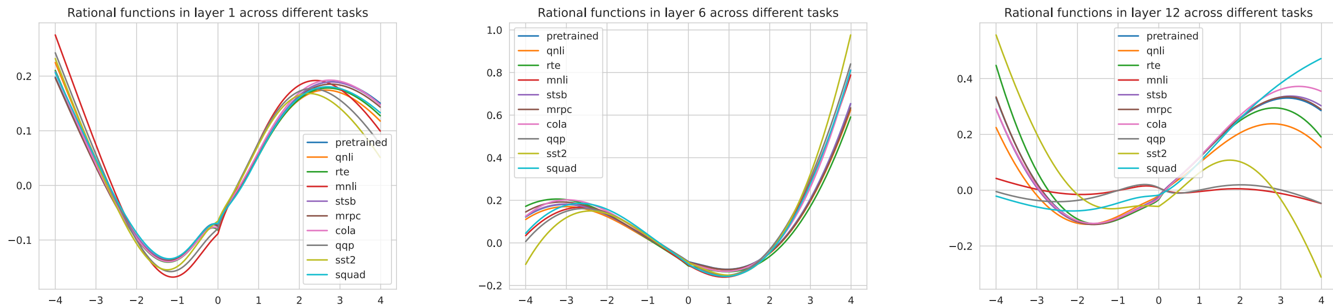


Figure 4: Learned rational activation functions of RAFT$^{\text{full}}$ in layers 1 (bottom), 6, and 12 (top) among different tasks

Model	ColA	SST2	MRPC	QQP	STSB	MNLI-matched/mismatched	QNLI	RTE	Avg.
low data 100 examples[1]									
$BitFit_{\text{BERT}}$	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]									
$BitFit_{\text{BERT}}$	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 6: Comparison between RAFT and vanilla BERT combined with BitFit.

Model	ColA	SST2	MRPC	QQP	STSB	MNLI-matched/mismatched	QNLI	RTE	Avg.
low data 100 examples[1]									
$BitFit^{\text{sub}}_{\text{BERT}}$	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
$BitFit^{\text{sub}}_{\text{RAFT}}$	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]									
$BitFit^{\text{sub}}_{\text{BERT}}$	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
$BitFit^{\text{sub}}_{\text{RAFT}}$	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**

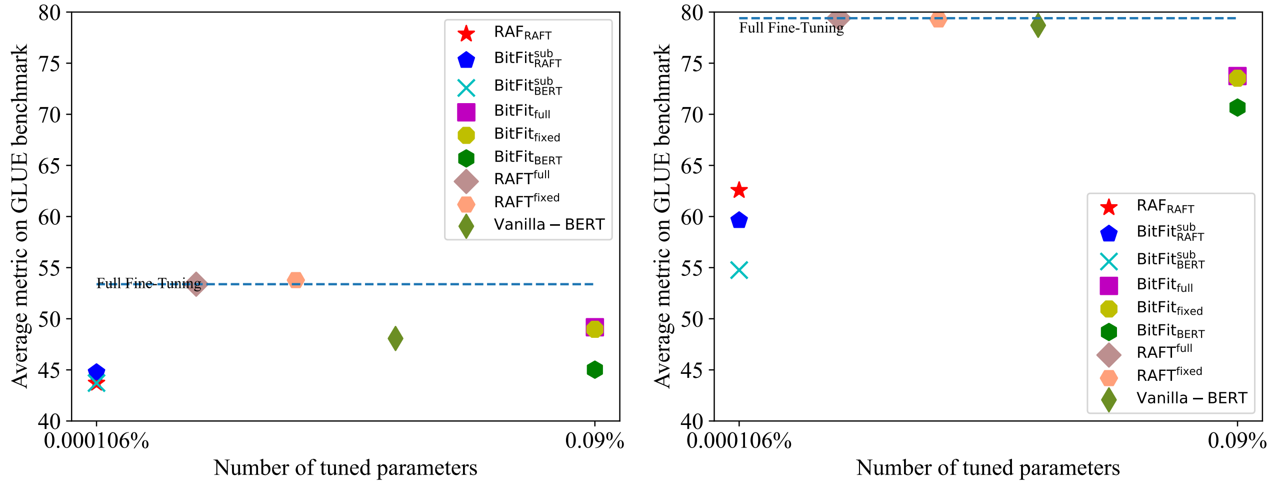
[1] Results are averaged over five random seeds: 5309, 202206, 20220602, 2259, 49

Table 7: Comparison between fine-tuning RAFs and a subset of 117 BitFit parameters with RAFT and vanilla BERT.

performs better with a 3.08 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 $BitFit_{\text{BERT}}$ 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 rational activation functions, we conduct experiments to only tune RAFs on the GLUE benchmark in low- and full-data settings. We call this setup where only 117 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.[8] $BitFit^{\text{sub}}_{\text{BERT}}$ represents tuning the subset of BitFit of vanilla BERT, and $BitFit^{\text{sub}}_{\text{RAFT}}$ represents tuning the subset of BitFit of RAFT. Table 7 presents the results of this comparison. To further compare with the results in Table 3 and Table 6, we plot Figure 5. 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 Vanilla BERT. Only fine-tuning 117 parameters ($BitFit^{\text{sub}}_{\text{BERT}}$, $BitFit^{\text{sub}}_{\text{RAFT}}$ and RAF_{RAFT}) —i.e., a negligible number of parameters compared to 110M parameters in vanilla BERT—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 $BitFit_{BERT}$) 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 $BitFit^{\text{sub}}_{\text{BERT}}$ by 7.8% and performs better than $BitFit^{\text{sub}}_{\text{RAFT}}$ by 2.94% in this setting.

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



(a) Comparison performance in low-data 100 scenario

(b) Comparison performance in full-data scenario

Figure 5: The number of parameters vs. the performance for different fine-tuning methods of RAFT and vanilla BERT.

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 vanilla BERT during pre-training. Our experimental results show that RAFT performs better than vanilla BERT 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, if 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.

Acknowledgments

The authors would like to thank Quentin Delfosse for his continued support and valuable advice regarding the existing implementation of rational activation functions. We further thank Stella Biderman, Fengyu Cai, Nils Dycke, Haau-Sing Li, Andreas Rücklé, Martin Tutek, Kexin Wang, and Neha Warikoo for their fruitful discussions and helpful feedback. This work has been funded by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

Ben Zaken, E.; Goldberg, Y.; and Ravfogel, S. 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)*, 1–9. Dublin, Ireland: Association for Computational Linguistics.

Bingham, G.; Macke, W.; and Miikkulainen, R. 2020. Evolutionary Optimization of Deep Learning Activation Functions. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, 289–296. New York, NY, USA: Association for Computing Machinery. ISBN 9781450371285.

Bingham, G.; and Miikkulainen, R. 2022. Discovering Parametric Activation Functions. *Neural Networks*, 148: 48–65.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 632–642. Lisbon, Portugal: Association for Computational Linguistics.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 1877–1901. Curran Associates, Inc.

Cer, D.; Diab, M.; Agirre, E.; Lopez-Gazpio, I.; and Specia, L. 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), 1–14. Vancouver, Canada: Association for Computational Linguistics.

Clark, J. H.; Garrette, D.; Turc, I.; and Wieting, J. 2022. Canine: Pre-training an Efficient Tokenization-Free Encoder for Language Representation. *Transactions of the Association for Computational Linguistics*, 10: 73–91.

Clark, K.; Luong, M.-T.; Le, Q. V.; and Manning, C. D. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *International Conference*.

Dagan, I.; Glickman, O.; and Magnini, B. 2005. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, 177–190. Springer.

Dauphin, Y. N.; Fan, A.; Auli, M.; and Grangier, D. 2017. Language Modeling with Gated Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 933–941. JMLR.org.

Delfosse, Q.; Schramowski, P.; Molina, A.; and Kersting, K. 2021. Recurrent Rational Networks. *arXiv preprint arXiv:2102.09407*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 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)*, 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics.

Ding, N.; Qin, Y.; Yang, G.; Wei, F.; Yang, Z.; Su, Y.; Hu, S.; Chen, Y.; Chan, C.-M.; Chen, W.; et al. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*.

Dolan, W. B.; and Brockett, C. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Dubey, S. R.; Singh, S. K.; and Chaudhuri, B. B. 2022. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503: 92–108.

Dugas, C.; Bengio, Y.; Bélisle, F.; Nadeau, C.; and Garcia, R. 2000. Incorporating Second-Order Functional Knowledge for Better Option Pricing. In Leen, T.; Dietterich, T.; and Tresp, V., eds., *Advances in neural information processing systems*, volume 13. MIT Press.

Fukushima, K. 1969. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4): 322–333.

He, P.; Liu, X.; Gao, J.; and Chen, W. 2020. DeBERTa: Decoding-Enhanced BERT with disentangled Attention. In *International Conference on Learning Representations*.

Hendrycks, D.; and Gimpel, K. 2016. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR*, abs/1606.08415.

Hoffmann, J.; Borgeaud, S.; Mensch, A.; Buchatskaya, E.; Cai, T.; Rutherford, E.; Casas, D. d. L.; Hendricks, L. A.; Welbl, J.; Clark, A.; et al. 2022. Training Compute-Optimal Large Language Models. *arXiv preprint arXiv:2203.15556*.

Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; and Gelly, S. 2019. Parameter-Efficient Transfer Learning for NLP. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 2790–2799. PMLR.

Iyer, S.; Dandekar, N.; and Csernai, K. 2017. First Quora Dataset Release: Question Pairs.

Izsak, P.; Berchansky, M.; and Levy, O. 2021. How to Train BERT with an Academic Budget. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 10644–10652. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics.

Joshi, M.; Choi, E.; Weld, D.; and Zettlemoyer, L. 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)*, 1601–1611. Vancouver, Canada: Association for Computational Linguistics.

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations*.

Levesque, H.; Davis, E.; and Morgenstern, L. 2012. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; and Zettlemoyer, L. 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*, 7871–7880. Online: Association for Computational Linguistics.

Lieber, O.; Sharir, O.; Lenz, B.; and Shoham, Y. 2021. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs*.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

Manessi, F.; and Rozza, A. 2018. Learning combinations of activation functions. In *2018 24th international conference on pattern recognition (ICPR)*, 61–66. IEEE.

Merchant, A.; Rahimtoroghi, E.; Pavlick, E.; and Tenney, I. 2020. What Happens To BERT Embeddings During Fine-tuning? In *Proceedings of the Third BlackboxNLP Workshop* *on Analyzing and Interpreting Neural Networks for NLP*, 33–44. Online: Association for Computational Linguistics.

Molina, A.; Schramowski, P.; and Kersting, K. 2020. Pad\'e Activation Units: End-to-end Learning of Flexible Activation Functions in Deep Networks. In *International Conference on Learning Representations*.

Moosavi, N.; Delfosse, Q.; Kersting, K.; and Gurevych, I. 2022. Adaptable Adapters. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 3742–3753. Seattle, United States: Association for Computational Linguistics.

Mosbach, M.; Khokhlova, A.; Hedderich, M. A.; and Klakow, D. 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*, 68–82. Online: Association for Computational Linguistics.

Naitzat, G.; Zhitnikov, A.; and Lim, L.-H. 2020. Topology of Deep Neural Networks. *Journal of Machine Learning Research*, 21(184): 1–40.

Nwankpa, C.; Ijomah, W.; Gachagan, A.; and Marshall, S. 2018. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Pfeiffer, J.; Rücklé, A.; Poth, C.; Kamath, A.; Vulić, I.; Ruder, S.; Cho, K.; and Gurevych, I. 2020. AdapterHub: A Framework for Adapting Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 46–54. Online: Association for Computational Linguistics.

Prangemeier, T.; Reich, C.; and Koeppl, H. 2020. Attention-based transformers for instance segmentation of cells in microstructures. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 700–707. IEEE.

Press, O.; and Wolf, L. 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*, 157–163. Valencia, Spain: Association for Computational Linguistics.

Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I.; et al. 2018. Improving language understanding by generative pre-training.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Rae, J. W.; Borgeaud, S.; Cai, T.; Millican, K.; Hoffmann, J.; Song, F.; Aslanides, J.; Henderson, S.; Ring, R.; Young, S.; et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P. J.; et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140): 1–67.

Rajpurkar, P.; Zhang, J.; Lopyrev, K.; and Liang, P. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2383–2392. Austin, Texas: Association for Computational Linguistics.

Ramachandran, P.; Zoph, B.; and Le, Q. V. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Sennrich, R.; Haddow, B.; and Birch, A. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Sharma, S.; Sharma, S.; and Athaiya, A. 2017. Activation functions in neural networks. *towards data science*, 6(12): 310–316.

Shazeer, N. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Shoeybi, M.; Patwary, M.; Puri, R.; LeGresley, P.; Casper, J.; and Catanzaro, B. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Smith, S.; Patwary, M.; Norick, B.; LeGresley, P.; Rajbhandari, S.; Casper, J.; Liu, Z.; Prabhumoye, S.; Zerveas, G.; Korthikanti, V.; et al. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*.

So, D.; Mańke, W.; Liu, H.; Dai, Z.; Shazeer, N.; and Le, Q. V. 2021. Searching for Efficient Transformers for Language Modeling. *Advances in Neural Information Processing Systems*, 34: 6010–6022.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, 1631–1642.

Sütfeld, L. R.; Brieger, F.; Finger, H.; Füllhase, S.; and Pipa, G. 2020. Adaptive blending units: Trainable activation functions for deep neural networks. In *Science and Information Conference*, 37–50. Springer.

Telgarsky, M. 2017. Neural Networks and Rational Functions. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 3387–3393. JMLR.org.

Thoppilan, R.; De Freitas, D.; Hall, J.; Shazeer, N.; Kulshreshtha, A.; Cheng, H.-T.; Jin, A.; Bos, T.; Baker, L.; Du, Y.; et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Wang, H.; Ma, S.; Dong, L.; Huang, S.; Zhang, D.; and Wei, F. 2022. Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*.

Warstadt, A.; Singh, A.; and Bowman, S. R. 2019. Neural Network Acceptability Judgments. *Transactions of the Association for Computational Linguistics*, 7: 625–641.

Williams, A.; Nangia, N.; and Bowman, S. 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)*, 1112–1122. Association for Computational Linguistics.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R. R.; and Le, Q. V. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Zhang, S.; Roller, S.; Goyal, N.; Artetxe, M.; Chen, M.; Chen, S.; Dewan, C.; Diab, M.; Li, X.; Lin, X. V.; et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Zhou, Y.; and Srikumar, V. 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)*, 1046–1061. Dublin, Ireland: Association for Computational Linguistics.

A Hyperparameters Tuning for Pre-training

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%

Table 8 shows hyperparameters we used for pre-training RAFT and vanilla BERT.

Hyperparameters	Vanilla BERT	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
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 8: Hyperparameters for pre-training RAFT and Vanilla BERT

B Data Statistics

Table 9 and Table 10 show data statistics of SQuAD and GLUE benchmark respectively.

C 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.

D Hyperarameters Tuning for BitFit

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: 3-20 epochs

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

Table 9: 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.

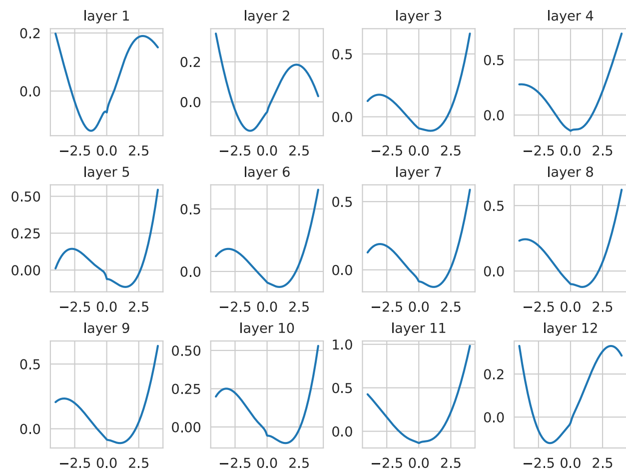


Figure 6: Learned RAFs of different layers after pre-training

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.

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 10: Dataset statistics of the GLUE benchmark

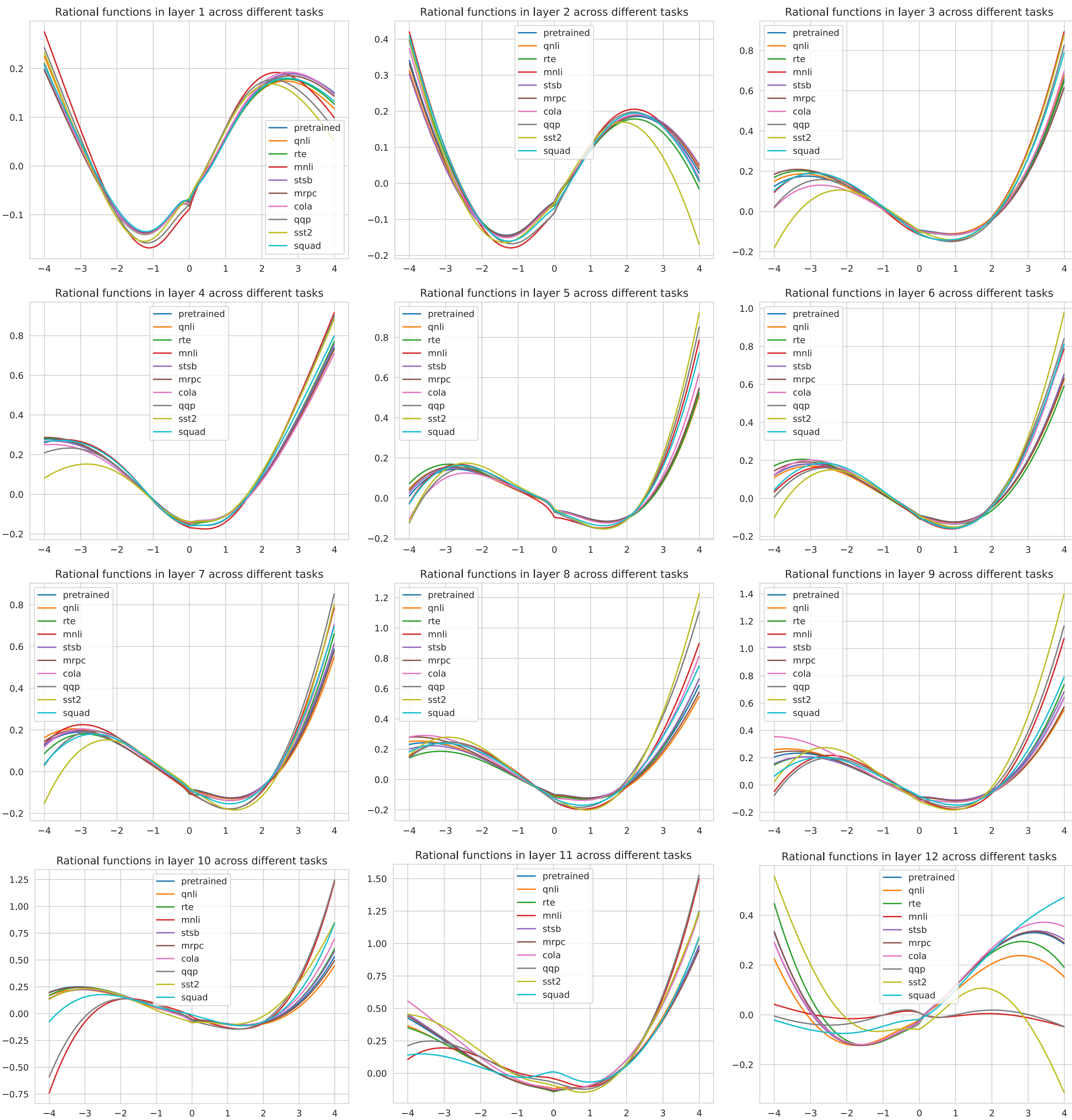


Figure 7: Learned RAFs in 12 layers across different tasks after fine-tuning