Multi-Layer Random Perturbation Training for Improving Model Generalization

Lis Kanashiro Pereira[1], **Yuki Taya**[2], **Ichiro Kobayashi**[3]
Ochanomizu University
kanashiro.pereira@ocha.ac.jp[1], g1620525@is.ocha.ac.jp[2]
koba@is.ocha.ac.jp[3]

Abstract

We propose a simple yet effective Multi-Layer **RA**ndom **P**erturbation **T**raining algorithm (RAPT) to enhance model robustness and generalization. The key idea is to apply randomly sampled noise to each input to generate label-preserving artificial input points. To encourage the model to generate more diverse examples, the noise is added to a combination of the model layers. Then, our model regularizes the posterior difference between clean and noisy inputs. We apply RAPT towards robust and efficient BERT training, and conduct comprehensive fine-tuning experiments on GLUE tasks. Our results show that RAPT outperforms the standard fine-tuning approach, and adversarial training method, yet with 22% less training time.

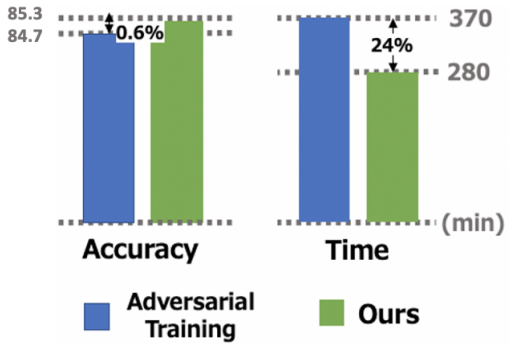

Figure 1: Acurracy and training time comparison between adversarial training (SMART) and RAPT on the MNLI (matched) development dataset. RAPT obains an accuracy gain of 0.6% yet with a training time drop of 24%.

1 Introduction

Although deep learning models have been very successful in various kinds of NLP problems, they are known to be sensitive towards input data distribution change which are pervasive across language tasks. Motivated by this, a recent line of work investigate the adversarial training technique to enhance the model robustness. Adversarial training has proven effective in improving model generalization and robustness in computer vision (Madry et al., 2017; Goodfellow et al., 2014) and natural language processing (NLP) (Zhu et al., 2019; Jiang et al., 2019; Cheng et al., 2019; Liu et al., 2020a; Pereira et al., 2020, 2021; Cheng et al., 2020). It works by augmenting the input with a small perturbation to steer the current model prediction away from the correct label, thus forcing subsequent training to make the model more robust and generalizable. In NLP, the cutting-edge research in adversarial training tends to perform an inner search for the most adversarial direction using gradient steps (Zhu et al., 2019; Jiang et al., 2019; Cheng et al., 2019; Liu et al., 2020a; Pereira et al., 2020, 2021; Cheng et al., 2020). This causes a significant overhead in training time. Moreover, such methods tend to add the adversarial perturbation only to the embedding layer, which might not be optimal.

By contrast, in this paper, we investigate a simpler direction by using only randomly sampled noise to generate label-preserving artificial input points. We thus propose a simple yet effective **RA**ndom **P**erturbation **T**raining algorithm (RAPT) for enhancing model robustness and generalization. For each instance, instead of using gradient steps to generate adversarial examples, RAPT adds randomly sampled noise to the hidden representaions of a randomly chosen layer, among multiple intermediate transformer layers (*i.e.* BERT layers). We hypothesize this might encourage the model to generate more diverse examples, and improve model generalization capability. Our model then regularizes the model posterior difference between clean and noisy inputs.

On the overall GLUE benchmark, RAPT outperforms the standard fine-tuning approach, and

Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pages 303–310

matches or improves the performance of strong adversarial training methods such as SMART (Jiang et al., 2019), yet with a significantly reduced training time. Figure 1 shows the accuracy gain and training time drop of RAPT compared to SMART on the MNLI (matched) development dataset.

2 RAPT

In this paper, we focus on fine-tuning BERT models (Devlin et al., 2019), as this approach has proven very effective for a wide range of NLP tasks.

The standard training algorithm seeks to learn a function $f(x;\theta) : x \rightarrow C$ as parametrized by θ, where C is the class label set. Given a training dataset D of input-output pairs (x, y) and the loss function $l(.,.)$ (*e.g.* cross entropy), the standard training objective would minimize the empirical risk:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim D}[l(f(x;\theta), y)].$$

By contrast, in adversarial training, as pioneered in computer vision (Goodfellow et al., 2014; Hsieh et al., 2019; Madry et al., 2017; Jin et al., 2019), the input would be augmented with a small perturbation that maximize the adversarial loss:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim D}[\max_{\delta} l(f(x+\delta;\theta), y)],$$

where the inner maximization can be solved by projected gradient descent (Madry et al., 2017).

Recently, adversarial training has been successfully applied to NLP as well (Zhu et al., 2019; Jiang et al., 2019; Pereira et al., 2020). In particular, SMART (Jiang et al., 2019) regularizes the standard training objective using *virtual adversarial training* (Miyato et al., 2018), by performing an inner loop to search for the most adversarial direction:

$$\begin{aligned}\min_{\theta} \mathbb{E}_{(x,y)\sim D}[l(f(x;\theta), y)+ \\ \alpha \max_{\delta} l(f(x+\delta;\theta), f(x;\theta))]\end{aligned} \tag{1}$$

Effectively, the adversarial term encourages smoothness in the input neighborhood, and α is a hyperparameter that controls the trade-off between standard errors and adversarial errors.

Current adversarial methods for NLP are slower than standard training, due to the inner maximization. SMART, for instance, requires an additional K projected gradient steps to find the perturbation that maximizes the adversarial loss (violation of local smoothness) (Liu et al., 2020a). In practice, $K = 1$ suffices SMART, and it is roughly 2 times slower compared to standard training. By contrast, RAPT completely removes the adversarial steps that use gradient steps from SMART and instead optimizes for stabilizing the model local smoothness using only randomly sampled noise for regularization:

$$\begin{aligned}\min_{\theta} \mathbb{E}_{(x,y)\sim D}[l(f(x;\theta), y)+ \\ \alpha l(f(x+\delta_r;\theta), f(x;\theta))]\end{aligned} \tag{2}$$

RAPT does not require extra backward computations and empirically works as well as or better than SMART. We consider the posterior regularization using the KL-divergence. For all tasks in this work, an input text sequence is divided into subword units w_t, $t = 1, \ldots, T$. The tokenized input sequence is then transformed into embeddings, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \in \mathbb{R}^n$, through a token encoder, which combines a token embedding, a (token) position embedding and a segment embedding (*i.e.* which text span the token belongs to) by element-wise summation. The embedding layer is used as

Algorithm 1 RAPT

Input: N: the number of training epochs, $\mathcal{D}, \{(x_1, y_1), ..., (x_n, y_n)\}$: the dataset, $\mathcal{X}$: the minibatch of the dataset, $f(x;\theta)$: the machine learning model parametrized by θ, σ^2: the variance of the random noise δ, η: the number to controll the size of the noise, L: the number of transformer based model's layers, f^{layer}: the function that computes the hidden representations of a given layer, $\boldsymbol{h}$: the hidden representations of a layer of the model, δ_r: the noise added to the hidden states of layer r , τ: the global learning rate, α: the hyperparameter for balancing the standard loss and the regularization term, max_layer: the number of the maximum layer where the noise can be added during training.

for $epoch = 1, 2, .., N$ **do**
  for $\mathcal{X} \in \mathcal{D}$ **do**
    Generate a random integer $r \in \{1, ...,$max_layer$\}$
    for $(x, y) \in \mathcal{X}$ **do**
      $\delta \sim N(0, \sigma^2 I)$
      $\delta \leftarrow \eta \frac{\delta}{\|\delta\|_\infty}$
      // x : forward pass to the last layer of the model
      for $layer = 1, 2, ..., L$ **do**
        $\boldsymbol{h} \leftarrow f^{layer}(\boldsymbol{h})$
        if $layer$ is r **then**
          $\boldsymbol{h} \leftarrow \boldsymbol{h} + \delta_r \boldsymbol{I}$
        end if
      end for
      $g_\theta \leftarrow \nabla_\theta l(f(x;\theta), y)$
        $+\alpha \nabla_\theta l(f(x+\delta_r;\theta), f(x;\theta))$
      $\theta \leftarrow \theta - \tau g_\theta$
    end for
  end for
end for

Output: θ

the input to multiple transformer layers (Vaswani et al., 2017) to generate the contextual representations, $\boldsymbol{h}_1^{layer}, \ldots, \boldsymbol{h}_T^{layer} \in \mathbb{R}^d$, which are the hidden states of an intermediate layer of the BERT model.

In RAPT, we sample noise vectors $\delta_1, \ldots, \delta_T$ from $\mathcal{N}(0, \sigma^2 I)$, with mean 0 and variation of σ^2. We first set a maximum layer (among all BERT intermediate layers) where the noise vector can be added. In each epoch, for each mini-batch selected, a layer among the first layer and the maximum layer previously set is randomly chosen. The noise input is then constructed by adding the noise vector δ_r (Equation 2) to the hidden state vector of the ramdomly chosen layer ($\boldsymbol{h}^{layer}$). Specifically, the model first performs a forward pass up to the chosen layer, then the noise vector is added to its hidden states, *i.e.* $\boldsymbol{h}_1^{layer} + \eta\delta_{r1}, \ldots, \boldsymbol{h}_T^{layer} + \eta\delta_{rT}$.

The model is then updated according to the task-specific objective for the task. To preserve the semantics, we constrain the noise to be small, and assume the model's prediction should not change after adding the perturbation. The algorithm of RAPT is shown in Algorithm 1.

3 Experiments

3.1 Datasets

We evaluate our model on the GLUE[1] benchmark, a collection of nine natural language understanding (NLU) tasks. It includes question answering (Rajpurkar et al., 2016), linguistic acceptability (Warstadt et al., 2018), sentiment analysis (Socher et al., 2013), text similarity (Cer et al., 2017), paraphrase detection (Dolan and Brockett, 2005), and natural language inference (NLI) (Dagan et al., 2006; Bar-Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009; Levesque et al., 2012; Williams et al., 2018). The diversity of the tasks makes GLUE very suitable for evaluating the generalization and robustness of NLU models. The GLUE tasks used in our experiments are summarized in Table 1.

3.2 Implementation Details

Our model implementation is based on the MT-DNN[2] framework (Liu et al., 2019, 2020b). We use $\text{BERT}_{\text{BASE}}$(Devlin et al., 2019) as the text encoder. We used ADAM (Kingma and Ba, 2015) as our optimizer with a learning rate in the range $\in \{1 \times 10^{-5}, 2 \times 10^{-5}, 8 \times 10^{-6}\}$ and a batch size $\in \{16, 32\}$. The maximum number of epochs was set to 6. A linear learning rate decay schedule with warm-up over 0.1 was used, unless stated otherwise. To avoid gradient exploding, we clipped the gradient norm within 1. All the texts were tokenized using wordpieces and were chopped to spans no longer than 512 tokens. For SMART , we follow (Jiang et al., 2019) and set the perturbation size to 1×10^{-5}. We choose the step size from $\{1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}, 1, 1.5, 2, 2.5, 3\}$. We set the variance for initializing the perturbation to 1×10^{-5}. The α parameter (Equation 1 and Equation 2) were both set to 1. During RAPT, we select η from $\{0.01, 1.5, 2, 2.3, 2.5\}$. We found that adding the noise to the layers 1 to 3 worked best in our experiments, therefore, the max_layer parameter in Algorithm 1 was set to 3. For more details, please refer to Section 4.

3.3 Main Results

We apply RAPT to $\text{BERT}_{\text{BASE}}$ and evaluate its performance on GLUE. Our results are shown in Table 2. We compare RAPT with the standard fine-tuning approach (Standard) and with the adversarial training method SMART. For our model RAPT, we compare a model that adds the noise to the embedding layer only, RAPT (Embedding), with the model that adds the noise to the other layers. We report the model that uses $\eta = 2$, RAPT ($\eta = 2$), and the model that selects the best η value, RAPT (BEST η).

Overall, we observed that SMART and RAPT were able to outperform standard fine-tuning, without using any additional knowledge source, and without using any additional dataset other than the target task datasets. These results suggest that adding noisy input points during training lead to a more robust model and help generalize better on unseen data. RAPT consistently outperforms standard training (with an average score of 81.9% vs. 81.1% on the test set).

Remarkably, RAPT outperforms SMART on most GLUE tasks, and obtains the highest average score among all tasks on both dev and test sets (85.1% and 81.9%, respectively), yet with a smaller training time (86.7 minutes on average, as shown in Table 3). This indicates that using only randomly sampled noise leads to better results. We also observe pronounced gains of RAPT on the smaller datasets such as RTE (with a test set accuracy of

[1] https://gluebenchmark.com/
[2] https://github.com/namisan/mt-dnn

Corpus	Task	#Train	#Dev	#Test	#Label	Metrics
Single-Sentence Classification						
CoLA	Acceptability	8.5k	1k	1k	2	Matthews corr
SST	Sentiment	67k	872	1.8k	2	Accuracy
Pairwise Text Classification						
MNLI	NLI	393k	20k	20k	3	Accuracy
RTE	NLI	2.5k	276	3k	2	Accuracy
QQP	Paraphrase	364k	40k	391k	2	Accuracy/F1
MRPC	Paraphrase	3.7k	408	1.7k	2	Accuracy/F1
QNLI	QA/NLI	108k	5.7k	5.7k	2	Accuracy
Text Similarity						
STS-B	Similarity	7k	1.5k	1.4k	1	Pearson/Spearman corr

Table 1: Summary information of the GLUE benchmark.

Methods	MNLI-m/mm Acc	QQP Acc/F1	RTE Acc	QNLI Acc	MRPC Acc/F1	CoLA Mcc	SST Acc	STS-B P/S Corr	Average Score
Standard^{dev}	84.1/84.3	90.5/87.3	69.3	90.9	**86.9**/90.7	58.3	92.4	89.9/89.4	84.5
SMART^{dev}	84.7/85.2	90.9/87.9	**70.8**	91.4	86.4/90.5	58.6	92.8	**90.2/89.7**	84.9
RAPT^{dev}(Embedding)	85.2/85.5	**91.2/88.3**	69.5	91.7	86.1/90.1	58.3	**92.9**	90.1/**89.8**	84.9
RAPT^{dev}($\eta = 2$)	85.2/85.5	**91.2**/88.2	70.6	**91.8**	**86.9/90.8**	**58.7**	92.8	89.9/89.6	**85.1**
RAPT^{dev}(BEST η)	**85.3/85.6**	**91.2**/88.2	70.6	**91.8**	**86.9/90.8**	**58.7**	92.8	90.1/89.7	**85.1**
Standard^{test}	84.2/83.2	88.6/70.6	67.9	90.2	**84.0/88.3**	52.1	93.1	86.3/85.0	81.1
SMART^{test}	85.0/84.2	**89.1**/71.8	**68.4**	90.7	83.6/88.1	52.8	93.5	86.9/85.6	81.6
RAPT^{test}(Embedding)	85.3/84.4	89.1/71.8	**68.4**	**91.1**	84.1/88.2	**53.0**	93.7	**87.5/86.4**	**81.9**
RAPT^{test}($\eta = 2$)	85.4/**84.7**	**89.1/71.9**	67.4	**91.1**	**84.0/88.3**	51.1	**93.8**	87.2/86.1	81.7
RAPT^{test}(BEST η)	**85.5/84.7**	**89.1/71.9**	68.0	**91.1**	**84.0/88.3**	52.3	**93.8**	86.8/85.5	81.8

Table 2: Comparison of standard fine-tuning (Standard), adversarial training (SMART) and our methods (RAPT) on GLUE. We use the $\text{BERT}_{\text{BASE}}$ model as the text encoder for all models. For a fair comparison, all these results are produced by ourselves.Theses scores are the average of each model across 5 random seeds. The GLUE test results are scored using the GLUE evaluation server.

68.4) and STS-B (with a test set Pearson/Spearman correlation scores of 87.5/86.4), which illustrates the benefits of RAPT on improving model generalizability.

Adding the perturbation to multiple intermediate layers leads to better results on the dev set than adding the random perturbation to the embedding layer only (average score of 85.1% vs. 84.9%, respectively). However, on the test set, adding the random perturbation to the embedding layer only leads to slightly better results than adding the random perturbation to multiple intermediate layers (81.9% vs. 81.8%, respectively). Still, both settings outperform SMART and standard fine-tuning.

Regarding the training time, on the overall GLUE benchmark, RAPT takes on average 22% less time to train compared to SMART (86.7 vs 110.9 minutes, respectively), as shown in Table 3.

4 Analysis

Here, we take a closer look at the embeddings and hidden representations of the intermediate layers of BERT using standard fine-tuning (Standard), SMART, and RAPT. After training, we extract the embeddings (for Standard and SMART) and the intermediate layer representations (where noise has been added) of the second intermediate layer (for RAPT) of a sentence. Then, we compute the top most similar words for each word in the sentence. We use the cosine similarity for computing the similarity between the vectors. As shown in Table 4, adding the noise to the intermediate layers using RAPT introduces more diversity. For instance, among the top-10 closest words to the word *country*, SMART shares eight words with the Standard fine-tuning methods, while RAPT shares only five.

Time (min)	MNLI	QQP	RTE	QNLI	MRPC	CoLA	SST	STS-B	Average
Standard	220	175	2.5	40	2.5	4.8	17.5	3.8	69.6
SMART	370	270	4	102	4.1	8	31.5	6	110.9
RAPT	280	220	3	80	3.4	6.5	24.5	4.9	86.7
RAPT/SMART	0.76	0.81	0.75	0.78	0.81	0.79	0.77	0.8	0.78

Table 3: Training time comparison between standard fine-tuning (Standard), SMART, and RAPT. RAPT/SMART denotes the ratio between the training time of RAPT and SMART. We used a A100-PCIE-40GB GPU to measure the training time.

Input: [CLS] by law , mexico can only export half the oil it produces to the united states . [SEP] mexico produces more oil than any other **country** . [SEP]

Standard	SMART	RAPT
nation	nation	countries
region	county	nation
county	region	region
island	countries	nations
countries	province	cities
state	island	kingdom
territory	state	global
province	territory	city
countryside	homeland	territory
mountain	trade	countryside

Table 4: Top-10 closest words to the vector of the word **country** using standard fine-tuning (Standard), SMART, and RAPT on the RTE development dataset. The words in red are the words not shared with the Standard fine-tuning method. RAPT has five different words, while SMART has only two.

Although having more diversity, the hidden representations are kept meaningful.

To further illustrate the diversity RAPT introduces to the model's hidden representations, Table 5 and Table 6 compare the embeddings produced by SMART before and after adding the perturbation and the hidden reporesentations produced by RAPT before and after adding the noise. For SMART, we can observe that the top-10 similar words to the word *country* do not change after adding the perturbation, and the cosine similarity scores are kept around the same. In our experiments, increasing the perturbation size leads to a drop in accuracy. For RAPT, we extract the hidden representations from the second layer of BERT. As we can see, the cosine similarity scores change and more diversity is introduced. In our development experiments, the accuracy instead increases, indicating that the hidden representations from the intermediate layers might be less sensitive to noise compared to the embedding layer.

Regarding which layer combination setting is best for adding the noise, we found that adding the noise to the layers 1 to 3 worked best in our experiments, as shown in Figure 2 and Figure 3, for the MNLI and MRPC datasets.

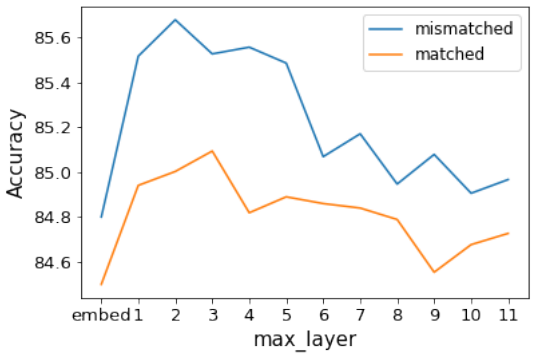

Figure 2: Performance on the MNLI development set as we change the layer combination to add the noise. On the x-axis, $max\_layer = embed$ denotes that the noise is added to the embedding layer only. All the other values denote that, for each mini-batch, a layer among the layer 1 up to this layer value is randomly chosen. The model is then updated according to the task-specific objective for the task.

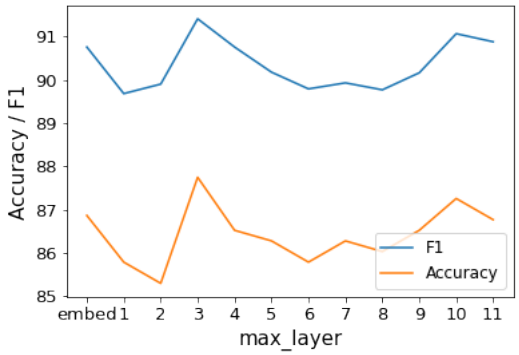

Figure 3: Performance on the MRPC development set as we change the layer combination to add the noise. On the x-axis, $max\_layer = embed$ denotes that the noise is added to the embedding layer only. All the other values denote that, for each mini-batch, a layer among the layer 1 up to this layer value is randomly chosen. The model is then updated according to the task-specific objective for the task.

Input: [CLS] by law , mexico can only export half the oil it produces to the united states . [SEP] mexico produces more oil than any other **country** . [SEP]

SMART		SMART (+noise)		RAPT		RAPT (+noise)	
word	similarity	word	similarity	word	similarity	word	similarity
nation	0.2991	nation	0.2992	countries	0.3976	countries	0.3141
county	0.2942	county	0.2945	nation	0.3328	nation	0.2714
region	0.2939	region	0.2941	region	0.3182	region	0.2546
countries	0.2736	countries	0.2738	nations	0.2934	nations	0.2430
province	0.2517	province	0.2515	kingdom	0.2929	cities	0.2337
island	0.2475	island	0.2473	province	0.2880	kingdom	0.2336
state	0.2449	state	0.2445	island	0.2810	global	0.2324
territory	0.2227	territory	0.2227	city	0.2702	city	0.2295
homeland	0.2106	homeland	0.2103	countryside	0.2682	territory	0.2293
trade	0.2090	trade	0.2087	realms	0.2670	countryside	0.2274

Table 5: Top-10 closest words to the vector of the word **country** using SMART and RAPT on the RTE development dataset. SMART (the leftmost column) denotes the embedding of the target vector (**country**) before adding the perturbation using SMART. SMART (+noise) denotes the embedding of the target vector (**country**) after adding the perturbation using SMART. In both cases, we compute the similarity between the embedding of the target word and the embeddings of the other words in the training set. RAPT (the second column from the right) denotes the hidden state vector of the second layer of BERT of the target vector (**country**) before adding the noise using RAPT. RAPT (+noise) denotes the hidden state vector of the second layer of BERT of the target vector (**country**) after adding the noise using RAPT. In both cases, we compute the similarity between the hidden state of the target word and the hidden states of the other words in the training set.

Input: [CLS] by law , mexico **can** only export half the oil it produces to the united states . [SEP] mexico produces more oil than any other country . [SEP]

SMART		SMART (+noise)		RAPT		RAPT (+noise)	
word	similarity	word	similarity	word	similarity	word	similarity
could	0.5857	could	0.5855	could	0.5656	could	0.4580
may	0.3869	may	0.3873	cannot	0.5162	cannot	0.4269
cannot	0.3774	cannot	0.3776	couldn	0.5098	couldn	0.4213
might	0.3736	might	0.3736	allows	0.3445	allows	0.2859
couldn	0.3651	couldn	0.3650	helps	0.3408	must	0.2759
must	0.3388	must	0.3388	shall	0.3218	may	0.2659
will	0.3312	will	0.3314	should	0.3183	doesn	0.2639
should	0.3119	should	0.3121	must	0.3130	sees	0.2531
would	0.2914	would	0.2915	doesn	0.3106	helps	0.2388
shall	0.2624	shall	0.2626	may	0.3083	allow	0.2318

Table 6: Top-10 closest words to the vector of the word **can** using SMART and RAPT on the RTE development dataset. SMART (the leftmost column) denotes the embedding of the target vector (**can**) before adding the perturbation using SMART. SMART (+noise) denotes the embedding of the target vector (**can**) after adding the perturbation using SMART. In both cases, we compute the similarity between the embedding of the target word and the embeddings of the other words in the training set. RAPT (the second column from the right) denotes the hidden state vector of the second layer of BERT of the target vector (**can**) before adding the noise using RAPT. RAPT (+noise) denotes the hidden state vector of the second layer of BERT of the target vector (**can**) after adding the noise using RAPT. In both cases, we compute the similarity between the hidden state of the target word and the hidden states of the other words in the training set.

5 Conclusion

We proposed RAPT, a simple and efficient random perturbation training algorithm for fine-tuning large scale pre-trained language models. Our experiments demonstrated that it achieves competitive results on GLUE tasks, without relying on

any additional resource other than the target task dataset. Moreover, our model can significanlty reduce the training time compared to adversarial training. RAPT is model-agnostic, and can also be generalized to solve other downstream tasks as well, and we will explore these directions as future work.

Acknowledgments

We thank the reviewers for their helpful feedback. This work has been supported by the project KAKENHI ID: 21K17802.

References

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, and Danilo Giampiccolo. 2006. The second PASCAL recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth pascal recognizing textual entailment challenge. In *In Proc Text Analysis Conference (TAC'09*.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Hao Cheng, Xiaodong Liu, Lis Pereira, Yaoliang Yu, and Jianfeng Gao. 2020. Posterior differential regularization with f-divergence for improving model robustness. *arXiv preprint arXiv:2010.12638*.

Yong Cheng, Lu Jiang, and Wolfgang Macherey. 2019. Robust neural machine translation with doubly adversarial inputs.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, MLCW'05, pages 177–190, Berlin, Heidelberg. Springer-Verlag.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

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

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 1–9, Prague. Association for Computational Linguistics.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Yu-Lun Hsieh, Minhao Cheng, Da-Cheng Juan, Wei Wei, Wen-Lian Hsu, and Cho-Jui Hsieh. 2019. On the robustness of self-attentive models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1520–1529.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2019. Is bert really robust? natural language attack on text classification and entailment. *arXiv preprint arXiv:1907.11932*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR (Poster) 2015*.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Xiaodong Liu, Hao Cheng, Pengcheng He, Weizhu Chen, Yu Wang, Hoifung Poon, and Jianfeng Gao. 2020a. Adversarial training for large neural language models. *arXiv preprint arXiv:2004.08994*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496.

Xiaodong Liu, Yu Wang, Jianshu Ji, Hao Cheng, Xueyun Zhu, Emmanuel Awa, Pengcheng He, Weizhu Chen, Hoifung Poon, Guihong Cao, and Jianfeng Gao. 2020b. The microsoft toolkit of multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:2002.07972*.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. 2018. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993.

Lis Pereira, Xiaodong Liu, Fei Cheng, Masayuki Asahara, and Ichiro Kobayashi. 2020. Adversarial training for commonsense inference. *arXiv preprint arXiv:2005.08156*.

Lis Pereira, Xiaodong Liu, Hao Cheng, Hoifung Poon, Jianfeng Gao, and Ichiro Kobayashi. 2021. Targeted adversarial training for natural language understanding. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 5385–5393 June 6–11, 2021.*

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

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew 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*, pages 1631–1642.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

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.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Thomas Goldstein, and Jingjing Liu. 2019. Freelb: Enhanced adversarial training for language understanding. *arXiv preprint arXiv:1909.11764*.