HyPe: Better Pre-trained Language Model Fine-tuning with Hidden Representation Perturbation

Hongyi Yuan[12*], **Zheng Yuan**[2], **Chuanqi Tan**[2], **Fei Huang**[2], **Songfang Huang**[2]
[1]Tsinghua University, [2]Alibaba Group
yuanhy20@mails.tsinghua.edu.cn
{yuanzheng.yuanzhen,chuanqi.tcq,f.huang,songfang.hsf}@alibaba-inc.com

Abstract

Language models with the Transformers structure have shown great performance in natural language processing. However, there still poses problems when fine-tuning pre-trained language models on downstream tasks, such as over-fitting or representation collapse. In this work, we propose HyPe, a simple yet effective fine-tuning technique to alleviate such problems by perturbing hidden representations of Transformers layers. Unlike previous works that only add noise to inputs or parameters, we argue that the hidden representations of Transformers layers convey more diverse and meaningful language information. Therefore, making the Transformers layers more robust to hidden representation perturbations can further benefit the fine-tuning of PLMs en bloc. We conduct extensive experiments and analyses on GLUE and other natural language inference datasets. Results demonstrate that HyPe outperforms vanilla fine-tuning and enhances generalization of hidden representations from different layers. In addition, HyPe acquires negligible computational overheads, and is better than and compatible with previous state-of-the-art fine-tuning techniques. Codes are released at https://github.com/Yuanhy1997/HyPe.

1 Introduction

Pretrain-then-finetune has become the mainstream paradigm in recent natural language processing (NLP) practices, and there emerges various pre-trained language models (PLMs) such as BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and XLNet (Yang et al., 2019). Vanilla PLM fine-tuning with common strategies (e.g., dropout (Srivastava et al., 2014) and AdamW (Loshchilov and Hutter, 2019)) can empower PLMs with excellent downstream performance. However, vanilla fine-tuned PLMs acquire performances with large variances on the downstream tasks (Dodge et al., 2020). Such unstable performances may results from over-fitting or representation collapse (Aghajanyan et al., 2021). These problems can be aggravated in low-resource scenarios (Zhang et al., 2021).

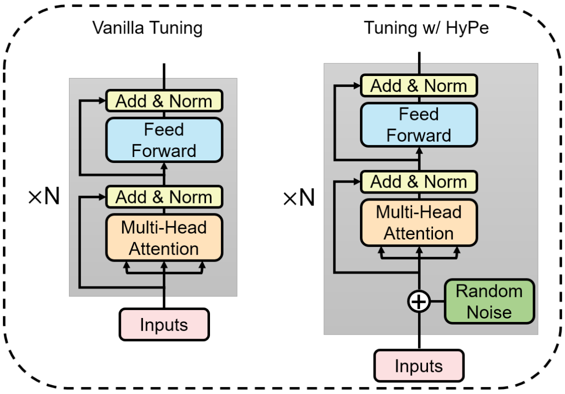


Figure 1: The overview of proposed HyPe fine-tuning technique. The random noise is added to the hidden representations fed into each Transformers layer in the forward computation of PLMs.

In recent literature, effective fine-tuning techniques have been proposed to improve the performance and generalization (transferability) of fine-tuned PLMs (Jiang et al., 2020; Lee et al., 2020; Chen et al., 2020). Besides other explicit regularization, adding noise is a widely-used strategy to smoothen the optimization landscape and mitigate over-fitting. For example, some works apply the perturbation to pre-trained parameter weights (e.g., NoisyTune (Wu et al., 2022)), input embedding features (e.g., R3F (Aghajanyan et al., 2021)) or gradients (e.g., ChildTuning (Xu et al., 2021)) during the fine-tuning process.

Injecting noise to input features is a conventional technique for generalization and can be seen as implicit parameter regularization (Bishop, 1995). Common PLMs are stacked basic neural network layers (i.e., Transformer layers (Vaswani et al., 2017)), and previous research (Tenney et al., 2019) points out that different Transformers layers

[*] Work done at Alibaba DAMO Academy.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 3246–3264
July 9-14, 2023 ©2023 Association for Computational Linguistics

of PLMs resolve different language information which is encoded in hidden representations. We turn to inject noise between layers to enhance the hidden semantic representations for better generalization on Transformers layer level.

Based on the above findings, we propose to improve fine-tuning by perturbing the hidden representations. As shown in Figure 1, we propose a simple yet effective fine-tuning technique named **HyPe** (**Hi(y)**dden representation **Pe**rturbation) that adds random noise to the hidden representations between layers (i.e., the inputs of **each** Transformers layer) to alleviate the performance of fine-tuned layers from degrading. To be concrete, we introduce no inductive biases to the distributions of noise in HyPe and focus on the pivotal influences of noise per se. Although noise can be compatible with auxiliary constrains (Aghajanyan et al., 2021) or include informative priors (Xu et al., 2021), they may lead to non-negligible computational overheads. We simply use the uniform and normal distributions as two variants of noise distributions and denote them as HyPe-U and HyPe-N, respectively. The computational overheads are marginal in HyPe. HyPe can also be regarded as a decoupling analysis of the above methods.

We conduct extensive experiments on GLUE benchmark (Wang et al., 2018) and HyPe improves vanilla fine-tuning up to 1.60 on BERT in terms of average scores of the relatively small datasets MRPC, RTE, CoLA, and STS-B, surpasses previous state-of-the-art techniques (i.e. R-Drop (Liang et al., 2021)) by 0.15, and improves performance in low-resource scenarios. Further analyses demonstrate that HyPe is also compatible with different scales of PLM (Section 5.1) and other fine-tuning techniques (Section 5.2), increases the robustness towards adversarial attacks (Section 5.3), and improves generalization across tasks and domains on different layers (Section 5.4).

To summarize our work, the main contributions are listed as follows:

1. We propose HyPe, a simple yet effective fine-tuning technique requiring little computational overhead to improve the performance and transferability of fine-tuning PLMs.

2. Extensive experimental results show that 1) HyPe improves fine-tuning in the aspect of task performance and generalization and is complementary to PLM scaling; 2) HyPe surpasses and is compatible with current state-of-the-art fine-tuning techniques.

2 Related Works

For large-scale PLMs, fine-tuning on downstream tasks may acquire unstable performances, resulting from over-fitting problems or failed training runs (Dodge et al., 2020; Zhang et al., 2021). Recent research has focused on how to alleviate such problems and effectively improve fine-tuning of PLMs on the downstream tasks.

A general idea is to make the best of the pre-trained weights and constrain the fine-tuned parameters from deviating much from the pre-trained weights. For example, Top-K Tuning (Houlsby et al., 2019) only fine-tunes the top-k layers of PLMs and keeps the lower pre-trained layers intact. Inspired by DropConnect (Wan et al., 2013), mixout (Lee et al., 2020) randomly replaces the weights of parameters with their pre-trained values instead of zero. RecAdam (Chen et al., 2020) introduces L^2 distance to penalize the change of weights from the pre-trained ones. ChildTuning (Xu et al., 2021) applies task-free or task-driven masks on the gradients thus only a subset of parameters are changed during fine-tuning. SAGE (Liang et al., 2022) uses differential updating step sizes for each parameter. Parameters with higher sensitivities are updated less aggressively where the computation of sensitivities is related to the pre-trained parameters in the cases of PLM fine-tuning.

Another line of work use noise to improve fine-tuning. R-Drop (Liang et al., 2021) uses KL divergence to regularize the discrepancy between the noised outputs produced by different dropout (Srivastava et al., 2014) masks during fine-tuning. Recently proposed NoisyTune (Wu et al., 2022) directly adds weight-aware noise to the pre-trained parameters before fine-tuning to improve performance. Based on the ideas of trust regions and adversarial training, FreeLB (Zhu et al., 2019), SMART (Jiang et al., 2020) and R3F (Aghajanyan et al., 2021) are proposed to improve fine-tuning by introducing adversarial noise to the input representations during training. Tong et al. (2022) create noised input representations by interpolating the representations between in-batch samples. The augmented fine-tuning data can alleviate overfitting and help PLMs learn a smoother decision boundary.

Previous research has proven the pivotal role of noise in improving PLM fine-tuning. Our proposed technique looks into the PLMs and adds noise to the hidden representations. Previous works introduce regulations along with the added noise. Generating random noise only requires little computational overheads, while additional regulations can cause non-negligible computational overheads in memory footprints or training time, such as R-Drop requiring two forward computations in each training step (Liang et al., 2021), and Child-Tuning$_D$ (Xu et al., 2021) requiring to pre-compute Fisher information matrices.

3 Hidden Representation Perturbation

HyPe is motivated to improve fine-tuning of PLMs. Perturbing input features for better training performance is proven in effect in wide machine learning applications (Nazaré et al., 2017; Aghajanyan et al., 2021). The structure of PLMs is complicated and different layers may have diverse impacts on understanding languages (Tenney et al., 2019). Therefore, by perturbing the hidden representations, we can improve the performance of each layer hence the whole PLMs in fine-tuning processes.

In the vanilla fine-tuning setting of language models, we denote the mapping of a PLM comprising of n network layers as $f_\theta(\cdot)$ and the classification head for the downstream task as $c_\psi(\cdot)$, where θ stands for the pre-trained parameters of the PLMs and ψ represents the parameters of the classification head on top of the PLM. Here we have the whole forward mapping $\hat{y} = c_\psi(f_\theta(x))$, where x and $\hat{y}$ are the embedded language inputs and predicted target labels respectively. The training objective is $\mathcal{L}(\theta, \psi) = \mathcal{L}\left(c_\psi(f_\theta(x)), y\right)$, where $\mathcal{L}$ is the loss function defined by tasks.

The basic layer block of nowadays PLMs (e.g., BERT) is Transformers (Vaswani et al., 2017) which mainly comprises of multi-head self-attention mechanism and feed-forward neural network. By stacking the Transformers layers, the scales of PLMs can get larger (e.g., the base and large versions of BERT contain 12 and 24 layers respectively). Given the stacking structure of PLMs, $f_\theta(x)$ can be decomposed as:

$$f_\theta(x) = g_{\theta^n} \circ g_{\theta^{n-1}} \circ \cdots g_{\theta^1}(x),$$

where $g_{\theta^i}(\cdot)$ is the mapping function of the i-th Transformers layer of the PLM, θ^i represents the parameters within layer i and we have $\cup_{i=1}^{n} \theta^i = \theta$. Let h^i represents the hidden states fed into the layer i, then $h^{i+1} = g_{\theta^i}(h^i)$. As the input sequences may comprise multiple word tokens, without the loss of generality, we omit the token position and sample index marks for x, y and h^i for simplicity.

Algorithm 1 Forward Propagation with HyPe

```
Input: Word Token Sequences $x$
$h^1$ = EmbeddingLayer($x$)
for each $i$ in layer number $n$ do
    Generate $\varepsilon^i$ from $\mathcal{N}(0, \sigma^2)$ or $\mathcal{U}(-\sigma, \sigma)$,
    $h^i = h^i + \varepsilon^i$,
    // ▷ Add Random Noise to Hidden States
    $h^{i+1} = g_{\theta^i}(h^i)$,
end for
$\hat{y} = c_\psi(h^n)$.
return $\hat{y}$
```

During fine-tuning, HyPe injects parameter-independent noise to the hidden states (representations) of each layer, then for the i-th layer:

$$\begin{aligned} h^{i+1} &= g_{\theta^i}(h^i + \varepsilon^i) \\ &:= g_{\theta^i}^{\varepsilon^i}(h^i), \end{aligned}$$

therefore the whole feed-forward process of the PLM becomes:

$$f_\theta^{\text{HyPe}}(x) = g_{\theta^n}^{\varepsilon^n} \circ g_{\theta^{n-1}}^{\varepsilon^i} \circ \cdots g_{\theta^1}^{\varepsilon^1}(x),$$

where ε^i is the random noise for layer i and each entry is distributed as $\mathcal{N}(0, \sigma^2)$ or $\mathcal{U}(-\sigma, \sigma)$. With HyPe, the training objective is simply:

$$\mathcal{L}^{\text{HyPe}}(\theta, \psi) = \mathcal{L}\left(c_\psi(f_\theta^{\text{HyPe}}(x)), y\right).$$

As shown above, HyPe is a simple and straightforward fine-tuning technique. It can be easily applied to different tasks and PLMs.

4 Experiments

In this section, we empirically demonstrate the effectiveness of HyPe through extensive experiments. We use GLUE benchmark (Wang et al., 2018) to illustrate the performance of HyPe in comparison to vanilla fine-tuning.

4.1 Datasets

GLUE GLUE is a widely-used benchmark designed for evaluating the natural language understanding abilities of models. Tasks in GLUE cover different aspects of language understanding including sentiment analysis, language acceptability, etc.

Dataset	STS-B	COLA	MRPC	RTE	AVG	STS-B	CoLA	MRPC	RTE	AVG
			BERT					XLNet		
Vanilla	$90.07_{0.67}$	$63.63_{1.82}$	$90.67_{0.92}$	$72.24_{2.18}$	79.15	$91.68_{0.06}$	$30.91_{24.99}$	$92.12_{0.40}$	$75.57_{11.63}$	72.57
HyPe-N	$\mathbf{90.37}_{0.43}$	$\mathbf{66.26}_{1.90}$	$91.98_{1.11}$	$74.37_{1.64}$	**80.75**	$91.87_{0.06}$	$\mathbf{64.40}_{0.72}$	$\mathbf{92.66}_{0.12}$	$83.15_{0.90}$	**83.02**
HyPe-U	$90.31_{0.41}$	$65.48_{0.45}$	$\mathbf{92.12}_{0.28}$	$\mathbf{74.49}_{0.95}$	80.60	$\mathbf{91.97}_{0.10}$	$58.05_{2.53}$	$92.40_{0.24}$	$\mathbf{83.27}_{1.04}$	81.42
			RoBERTa					ELECTRA		
Vanilla	$91.90_{0.11}$	$65.55_{0.36}$	$92.09_{0.16}$	$81.71_{2.13}$	82.81	$92.27_{0.16}$	$46.41_{32.83}$	$93.49_{0.86}$	$88.33_{0.45}$	80.13
HyPe-N	$92.22_{0.12}$	$\mathbf{66.04}_{1.83}$	$92.04_{0.58}$	$82.79_{1.51}$	83.27	$\mathbf{92.37}_{0.06}$	$\mathbf{68.88}_{0.98}$	$\mathbf{94.00}_{0.61}$	$\mathbf{88.45}_{1.56}$	**85.93**
HyPe-U	$\mathbf{92.29}_{0.06}$	$65.77_{1.22}$	$\mathbf{92.60}_{0.71}$	$\mathbf{84.12}_{0.29}$	**83.70**	$92.20_{0.16}$	$51.01_{25.34}$	$93.91_{0.44}$	$\mathbf{88.45}_{1.18}$	81.39

Table 1: Comparison results of HyPe and vanilla fine-tuning on relatively small datasets using different PLMs. The best results are in **bold**. The standard deviations for each results are shown in the subscripts. AVG means the average score of the four datasets. Vanilla fine-tuning on CoLA using XLNet and ELECTRA is highly unstable hence resulting in low average scores with high variances.

Dataset	SST2	QNLI	QQP	MNLI	AVG
Vanilla	$95.83_{0.30}$	$93.43_{0.77}$	$88.99_{0.12}$	$\mathbf{90.58}_{0.07}$	92.21
HyPe-N	$\mathbf{96.06}_{0.05}$	$93.98_{0.27}$	$89.15_{0.13}$	$90.32_{0.07}$	92.38
HyPe-U	$96.02_{0.19}$	$\mathbf{94.19}_{0.24}$	$\mathbf{89.25}_{0.15}$	$90.25_{0.13}$	**92.43**

Table 2: Comparison results of HyPe and vanilla fine-tuning on large GLUE datasets using RoBERTa. The best results are in **bold**. The standard deviations for each results are shown in the subscripts. AVG means the average score of the four datasets.

Following Xu et al. (2021), we mainly use four relatively small datasets STS-B (Cer et al., 2017), MRPC (Dolan and Brockett, 2005), RTE (Socher et al., 2013a) and CoLA (Warstadt et al., 2019), as the over-fitting problem is more notable in the small data settings (Dodge et al., 2020). We also use other larger datasets SST2 (Socher et al., 2013b), QNLI (Rajpurkar et al., 2016), QQP[1] and MNLI (Williams et al., 2018) to further illustrate the performance of HyPe. We report performance on the development set since the test set labels are not released. The statistics of GLUE are listed in Appendix B.

4.2 Experiment Settings

For all experiments listed in the following, we do grid search on the learning rates and report the average results over three different random seeds. We use the hidden representations of the first special token (e.g., [CLS] in BERT) for sentence representation. For our HyPe, we conduct experiments on two variants with different distributions of noise, denoted as HyPe-N where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ and HyPe-U where $\varepsilon \sim \mathcal{U}(-\sigma, \sigma)$. HyPe is only added during training. When using HyPe, we empirically find that turning off dropout will improve the technique's performance, which will be discussed in Section 5.5. Therefore, we run experiments with HyPe using no dropout on hidden representations. For the more detailed settings concerning individual experiments, we list them in Appendix A∼G.1.

4.3 Performance on GLUE

To illustrate the generality of HyPe, we conduct experiments on the GLUE benchmark with four popular PLMs, BERT-large (Devlin et al., 2019), RoBERTa-large (Liu et al., 2019), ELECTRA-large (Clark et al., 2020) and XLNet-large (Yang et al., 2019). We use the PLMs from Huggingface Hub[2] (Wolf et al., 2020).

We first evaluate HyPe on the four relatively small datasets from GLUE. As shown in Table 1, both variants of HyPe with different noise consistently improve the performance over vanilla fine-tuning. On average scores across tasks, the improvements are 1.60 on BERT, 0.89 on RoBERTa, 7.45 on XLNet, and 5.80 on ELECTRA, respectively. In addition, HyPe can help the model converge better on the CoLA dataset using XLNet and ELECTRA with smaller standard deviations.

We also evaluate HyPe on relatively large datasets. We fine-tune RoBERTa on the larger datasets of GLUE benchmark, with and without HyPe. The results listed in Table 2 also show that HyPe improves performance with large amounts of fine-tuning samples. The average gains across datasets are 0.22 and 0.17 for HyPe-U and HyPe-N

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

[2] For these four PLMs, we use `bert-large-cased`, `roberta-large`, `google/electra-large-discriminator`, and `xlnet-large-cased`, respectively.

Dataset	Vanilla	HyPe-N	HyPe-U
STS-B	$89.28_{0.07}$	$89.33_{0.59}$	$\mathbf{89.77}_{0.41}$
CoLA	$43.20_{12.26}$	$55.34_{1.70}$	$\mathbf{56.34}_{2.23}$
MRPC	$88.02_{0.80}$	$\mathbf{89.74}_{1.48}$	$88.49_{0.11}$
RTE	$61.61_{6.95}$	$74.61_{7.32}$	$\mathbf{78.58}_{5.02}$
SST2	$92.47_{0.68}$	$\mathbf{92.97}_{1.12}$	$92.51_{0.47}$
QNLI	$84.94_{1.14}$	$\mathbf{85.39}_{1.61}$	$84.86_{1.19}$
QQP	$73.92_{3.59}$	$74.97_{1.69}$	$\mathbf{76.38}_{0.77}$
MNLI	$60.90_{11.89}$	$79.90_{1.49}$	$\mathbf{80.17}_{0.73}$
MNLI-mm	$62.56_{11.43}$	$80.97_{1.49}$	$\mathbf{81.43}_{0.63}$
AVG	72.99	80.36	**80.95**

Table 3: Two variants of HyPe results in low-resource scenarios in comparison to vanilla fine-tuning. Best results are in **bold**, and standard deviations are marked in subscripts.

respectively.

In summary of the aforementioned results, we can conclude that HyPe improves and stabilizes fine-tuning consistently across different datasets and PLMs. In addition, we observe that the improvements are more significant on small datasets, which indicates that HyPe has the capability of mitigating the over-fitting problem of PLM fine-tuning.

4.4 Performance with Low Resources

As the amount of training data becomes smaller, the over-fitting problem can be more severe. Since HyPe shows good performance in mitigating over-fitting on relatively small GLUE datasets, we create a low-resource setting to further illustrate the performance of HyPe. We follow previous research (Xu et al., 2021) for the low-resource setting. In detail, we subsample the training samples of each dataset in GLUE benchmark to a training subset with 1k samples, and evaluate the performance using the original development set.

As shown in Table 3, both variants of HyPe with RoBERTa-large outperform vanilla consistently. On average, the improvements brought by HyPe-N and HyPe-U are up to 7.37 and 7.96 respectively. On some datasets, the improvements are significant: for example, the improvements of HyPe-N and HyPe-U are up to 13.00 and 16.97 on RTE respectively. In summary, HyPe can effectively prevent PLMs from over-fitting when fine-tuning in low-resource scenarios.

5 Further Analysis

We provide further analyses and discussions on the performances of HyPe for model scaling, methods comparison and combination, adversarial attacks, and hyper-parameters in this section.

	STS-B	COLA	MRPC	RTE	Avg. Imp.
Base	91.58	63.81	92.34	84.84	-
/w HyPe	91.86	65.08	93.07	85.44	+0.72
Large	92.39	67.01	93.34	90.97	-
/w HyPe	92.68	67.92	93.17	91.10	+0.29
XL	92.62	69.12	92.97	91.34	-
/w HyPe	92.56	70.74	93.33	91.94	+0.63
XXL	93.02	70.24	93.80	92.06	-
/w HyPe	93.23	70.76	94.26	92.42	+0.39

Table 4: Performances on models with different parameter sizes. Avg. Imp. represents Averaged Improvements.

5.1 Performance on Parameter Scaling

We investigate how HyPe performs as parameters of PLM scale up. We experiment on DeBERTa (He et al., 2021) with 4 sizes: base, large, XL, and XXL. The experimental details are shown in Appendix F. Results in Table 4 show that HyPe uniformly improves vanilla fine-tuning across different model sizes. The averaged improvements are +0.72, +0.29, +0.63, and +0.39 as the size scales up. This demonstrates that HyPe is complimentary to PLMs parameter scaling.

5.2 Methods Comparison

To compare HyPe with previous techniques for effective fine-tuning, we review and compare with the following baselines: (1) **Top-K Tuning** (Houlsby et al., 2019); (2) **Mixout** (Lee et al., 2020); (3) **RecAdam** (Chen et al., 2020); (4) **R3F** (Aghajanyan et al., 2021); (5) **ChildTuning** (Xu et al., 2021); (6) **R-Drop** (Liang et al., 2021); (7) **LNSR** (Hua et al., 2021); (8) **NoisyTune** (Wu et al., 2022). The comparison experiments are conducted on the GLUE datasets STS-B, CoLA, MRPC, and RTE.

Comparison From the results shown in Table 5, HyPe achieves the best results on STS-B and CoLA, and consistently outperforms Top-K Tuning, Mixout, RecAdam, Child-Tuning$_F$, and NoisyTune across different datasets. HyPe-N achieves the best average score of four tasks and surpasses the previous state-of-the-art R-Drop by 0.15. On MRPC and RTE, HyPe achieves competitive results with R3F, R-Drop, and Child-Tuning$_D$. However, R3F and R-Drop include a KL divergence regularization objective and need to make two forward computations in a fine-tuning step. Both methods may have additional computational overhead. Take GPU memory footprints as an example, under the same training setting (e.g., batch size of 16), R3F and R-Drop require 16GB of memory while HyPe only requires about 11GB of memory. Child-Tuning$_D$ is a task-specific method and needs additional computation

Dataset	STS-B	COLA	MRPC	RTE	Average
Vanilla	$90.07_{0.67}$	$63.63_{1.82}$	$90.67_{0.92}$	$72.24_{2.18}$	79.31
Top-K Tuning*	89.97	62.63	91.09	70.90	78.65
Mixout*	89.99	63.60	91.29	72.15	79.26
RecAdam*	89.86	64.33	90.85	71.63	79.17
LNSR*	90.23	63.35	88.50	73.31	78.85
Child-Tuning$_F$	$90.24_{0.45}$	$63.86_{1.60}$	$91.43_{1.11}$	$73.77_{2.09}$	79.83
Child-Tuning$_D$	$\underline{90.34}_{0.55}$	$64.48_{1.29}$	$91.43_{0.24}$	$73.65_{0.51}$	79.97
R-Drop	$90.29_{0.37}$	$65.06_{0.35}$	$91.84_{0.54}$	$\mathbf{75.21}_{0.90}$	$\underline{80.60}$
R3F	$90.21_{0.54}$	$64.90_{1.50}$	$\mathbf{92.23}_{0.67}$	$\underline{74.73}_{2.41}$	80.52
NoisyTune	$90.22_{0.55}$	$64.67_{0.27}$	$91.46_{0.64}$	$73.89_{1.78}$	80.06
HyPe-N	$\mathbf{90.37}_{0.43}$	$\mathbf{66.26}_{1.90}$	$91.98_{1.11}$	$74.37_{1.64}$	**80.75**
HyPe-U	$90.31_{0.41}$	$\underline{65.48}_{0.45}$	$\underline{92.12}_{0.28}$	$74.49_{0.95}$	$\underline{80.60}$

Table 5: Results comparing to other effective fine-tuning techniques using BERT-large. Best results are **bold**, and the second best results are underlined. Standard deviations are provided in subscripts. * are reported from Xu et al. (2021) as the experiment settings are similar.

advGLUE	SST-2	MNLI(m/mm)	RTE	QNLI	QQP
Vanilla	33.03	28.72/27.05	40.46	39.77	37.91
HyPe	**34.45**	**32.51/27.78**	**48.56**	**47.97**	**40.17**

Table 6: Accuracy results on the adversarial attacked testing samples from advGLUE using BERT-large. Detailed data introduction and experiment settings are in Appendix E. MNLI(m/mm) stands for MNLI-match/mismatch.

of the Fisher information matrix. HyPe only adds task-agnostic random noise to the hidden representations, and is more computationally efficient.

Compatibility To show the complementarity of HyPe with other effective fine-tuning techniques, we conduct experiments on the combination of techniques. We integrate HyPe-N with four recently proposed state-of-the-art techniques, R-Drop, R3F, Child-Tuning$_D$, and NoisyTune. We use MRPC, STS-B, CoLA, and RTE datasets and apply different combinations to RoBERTa and BERT. The average results of the four tasks in Figure 2 show that combining HyPe with other effective fine-tuning techniques can further boost performance. This illustrates that the improvements brought by adding noise to hidden representations do not overlap with other techniques, thus another advantage of HyPe is being compatible with others. The details of experiment settings and results are shown in Appendix D.

5.3 Performance on Adversarial Samples

Fine-tuning PLMs may prone to bad generalization of adversarial attacks. Results listed in Table 6 on textually crafted adversarial samples from advGLUE (Wang et al., 2021) show that vanilla fine-tuned PLMs suffer from adversarial attacks, and compared to vanilla, the performance gains brought by HyPeN are up to +1.42, +3.79/+0.73, +8.10, +8.20 and +2.26 on advSST-2, advMNLI(m/mm), advRTE, advQNLI and advQQP respectively. The results demonstrate that injecting noise into the hidden representations can increase the robustness of fine-tuning towards adversarial attacks.

5.4 Performance on Generalization

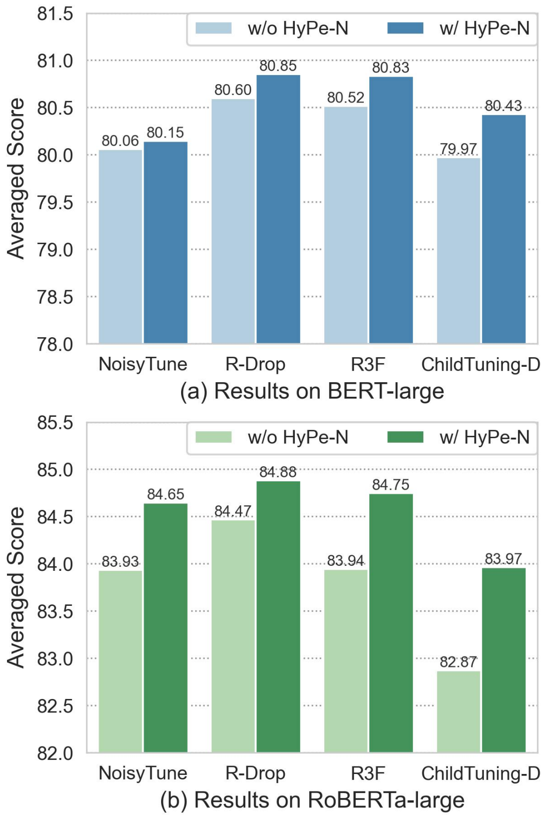
Probings on generalization abilities is another scope to access the over-fitting problem of fine-tuning (Xu et al., 2021; Aghajanyan et al., 2021). In this subsection, we discuss the transferability of HyPe fine-tuned PLMs from the perspective of task generalization and domain generalization.

Task Generalization Probing One side effect of over-fitting is the degeneration of the dense representations of PLMs after fine-tuning, and the phenomenon is named representation collapse (Aghajanyan et al., 2021). We probe fine-tuned PLMs task generalization by training a PLM on one task and then evaluating on another with parameters fixed. Previous works freeze the whole parameters of PLMs and only tune a linear classifier for other tasks (Aghajanyan et al., 2021; Xu et al., 2021). As HyPe perturbs hidden representations among layers, we extend this experiment by training separated linear classifiers for hidden representation of each layer, and show their representational abilities.

We use MRPC, STS-B, RTE, and CoLA for the target tasks and start from the checkpoints of RoBERTa fine-tuned on SST2. As depicted in Figure 3, it is shown that 1) both variants of HyPe achieve better performance than vanilla fine-tuning overall; 2) the improvement is more significant on higher layers of the PLM. In the lower layers, the three lines seem entangled. This is reasonable as the lower layers of PLMs are changed less in fine-

	Fine-tune on MNLI				Fine-tune on SNLI			
	Vanilla	HyPe-N	Δ	HyPe-U	Δ	Vanilla	HyPe-N	Δ	HyPe-U	Δ
SNLI	90.67	91.30	**+0.63**	90.77	**+0.10**	92.99	93.60	**+0.61**	93.49	**+0.50**
SICK	90.30	89.76	-0.54	89.16	-1.14	87.74	89.09	**+1.35**	90.30	**+2.56**
SciTaiL	80.04	81.40	**+1.36**	80.44	**+0.40**	79.58	80.71	**+1.13**	80.83	**+1.25**
QQP	75.84	76.22	**+0.38**	76.04	**+0.20**	74.12	75.12	**+1.00**	74.90	**+0.78**
MNLI	89.91	90.42	**+0.51**	90.01	**+0.10**	86.66	87.63	**+0.97**	87.40	**+0.74**
MNLI-mm	90.73	91.12	**+0.39**	90.82	**+0.09**	87.28	88.44	**+1.16**	88.03	**+0.75**

Table 7: Comparison results on domain generalization. Δ represents the change of performance over vanilla fine-tuning. Improvements of two HyPe variants over vanilla fine-tuning are in **bold**. All evaluation datasets are out-of-domain except the dataset from the training set itself.



(a) Results on BERT-large

(b) Results on RoBERTa-large

Figure 2: Results for combining HyPe-N with other effective fine-tuning techniques. The result numbers are the average evaluation scores of task MRPC, STS-B, CoLA, and RTE.

tuning, as discussed by previous research (Durrani et al., 2021). The results show that PLMs fine-tuned with HyPe maintain better representation ability across layers, thus demonstrating that they suffer less from the over-fitting problem.

Domain Generalization Probing Besides generalization across tasks, Xu et al. (2021) also experiments on transferability across domains for the same. Good domain generalization may indicate that PLMs are fine-tuned to learn general semantic features and not easily over-fit the domain-specific information within training data. Following their work, we use natural language inference (NLI) tasks from different domains. Beyond NLI datasets MNLI and QQP in GLUE, we additionally introduce datasets SNLI (Bowman et al., 2015), SciTaiL (Khot et al., 2018) and SICK (Marelli et al., 2014). For MNLI, we use both development sets of MNLI-match (MNLI) and MNLI-mismatch (MNLI-mm) for evaluation. Following previous research, we fine-tune RoBERTa-large with different techniques on a 5k sample subset of MNLI and SNLI datasets, respectively. Then, we test the fine-tuned PLMs on the aforementioned datasets to show the domain generalization ability. The detailed introductions of the datasets, experiment settings, and necessary label mappings are shown in Appendix C.

The results listed in Table 7 illustrate that both variants of HyPe outperform vanilla fine-tuned models on most of the out-of-domain datasets, except for SICK when fine-tuned on MNLI. This shows that HyPe can mitigate model over-fitting to domain-related features. Therefore when the domain of downstream tasks varies, PLMs fine-tuned with HyPe can still have good performance.

Both generalization probing experiments above demonstrate that HyPe can help PLMs avoid representation collapse and over-fitting to the fine-tuning data, hence obtaining good generalization across tasks and domains.

5.5 Discussions

Do the noise forms and scales matter? Here we discuss how performance varies given different noise distributions and scales σ.

In Table 8, we can conclude from the results that 1) given different distributions and scales, HyPe consistently outperforms vanilla fine-tuning; 2) for different tasks the best choice for distributions and scales may differ: for example, on CoLA, the language acceptability task, the best choice is using a normal distribution with small scale $\sigma = 10^{-5}$, while on MRPC, the semantical equivalence task, it

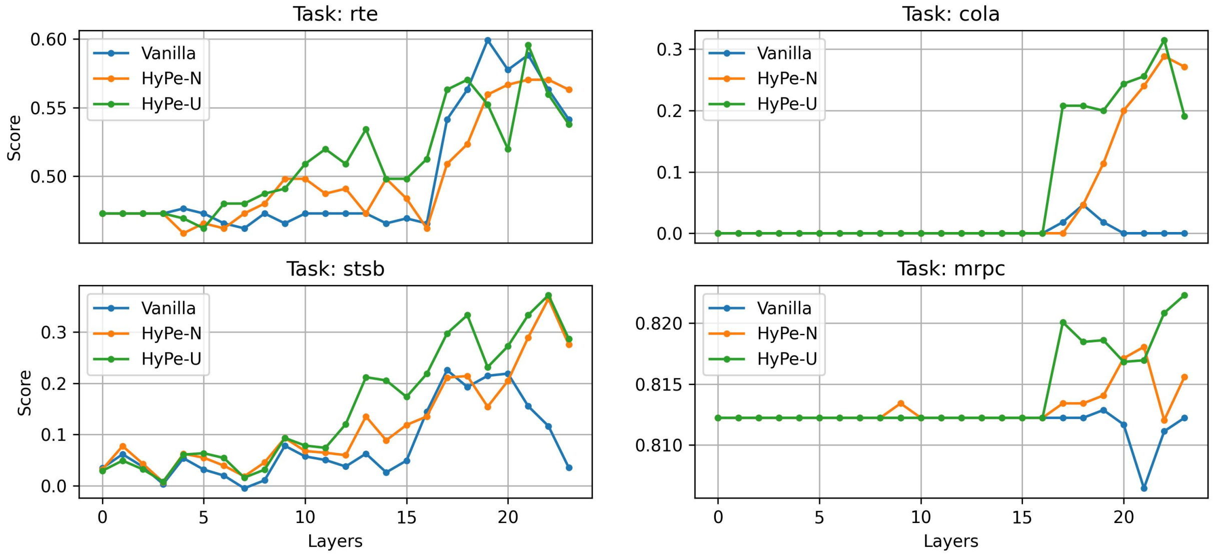


Figure 3: Linear probings on the hidden representations across layers based on the RoBERTa-large checkpoint fine-tuned on SST2.

Dataset	STS-B	CoLA	MRPC	RTE
Vanilla	90.07	63.63	90.67	72.24
HyPe-N				
$\sigma = 10^{-5}$	**90.37**	**66.26**	91.14	74.37
$\sigma = 10^{-4}$	90.29	64.71	91.98	73.16
$\sigma = 10^{-3}$	**90.37**	64.94	91.73	72.80
$\sigma = 10^{-2}$	90.36	64.60	91.61	74.13
HyPe-U				
$\sigma = 10^{-5}$	90.24	65.48	**92.12**	73.65
$\sigma = 10^{-4}$	90.31	65.13	91.83	**74.49**

Table 8: Results of analysis experiments on the distribution forms and scales of the noise. Experiments conducted on BERT-large.

is better to use uniform distribution with the scale of $\sigma = 10^{-5}$.

Relation with Dropout Note that in the aforementioned experiments we turn off dropout when using HyPe. When combining HyPe-N with dropout, we empirically find that the performance degrades. The average score drops from 80.75 to 79.92, as shown in Table 9. The possible explanation is that the improvement brought by dropout and that by HyPe partly overlap, since dropout randomly sets entries of hidden representations to zero, which can be regarded as a *discrete* form of 0/1 noise *multiplied* to different hidden representations where each entry of noise obeys a Bernoulli distribution. In terms of HyPe, we *add continuous* random noise to the hidden representations. Empirically our HyPe shows superior performance than dropout, as in vanilla fine-tuning we apply 0.1 dropout rate. Therefore, adding continuous noise to the hidden representations in HyPe can be a good alternative for the discrete noise of dropout.

Dataset	STS-B	CoLA	MRPC	RTE	AVG
Vanilla	90.07	63.63	90.67	72.24	79.15
HyPe-N	**90.37**	**66.26**	**91.98**	**74.37**	**80.75**
HyPe-N+DP	90.21	64.52	91.53	73.41	79.92

Table 9: Results with and without dropout when using HyPe on BERT-large. DP represents dropout.

We leave the discussions of adding noise only to hidden representations of a subset of layers and adding additional noise to the representations of self-attention mechanism outputs inside each Transformers layer to Appendix G.

6 Conclusion

To conclude, we introduce HyPe, a technique to improve PLM fine-tuning. HyPe enhances fine-tuning by perturbing the intermediate hidden representations of a PLM with task and model agnostic random noise. Through experiments on GLUE and other NLI tasks, we demonstrate that PLMs fine-tuned with HyPe have better performance and transferability in comparison to vanilla fine-tuning, especially in a low-resource scenario. In further analyses, without additional regulation like KL-divergence and computational overheads, HyPe obtains superior performances compared to existing state-of-the-art fine-tuning techniques, and can further boost fine-tuning combined with others. Fine-tuning with HyPe improves hidden representations across different layers and provide stable improvements for generalization, adversarial attack and different model scales.

Limitations

Collapsed fine-tuning runs mostly occur in the low resource scenario where PLMs may easily overfit to the small data. The improvement with the proposed technique becomes marginal when the amount of training data scales up, as shown in Table 2. The other limitation is that HyPe introduces two new hyper-parameters: The noise distribution form and the scale of variance. To achieve the best performance, we may need to search for different combinations of hyper-parameters.

Ethic Statement and Broader Impact

As the parameter scale of PLMs and the pre-training cost get much larger hence showing better brilliant performance in language modeling, it is necessary to improve the fine-tuning performance of the language model in an effective and efficient way. Our proposed HyPe improves large PLM fine-tuning by only adding noise to the hidden representations. Unlike previous works, we do not include additional regulations since additional regulations may require non-negligible computational resources which may increase as the scale of PLM gets larger. It is important to develop effective fine-tuning techniques that are efficient and easy to implement. Through extensive discussions of HyPe, we illustrate that including perturbations in the features or representations could be the key part of why previous techniques work. Besides, we show that our HyPe can be a good continuous noise alternative for the widely-used dropout which can be regarded as 0/1 discrete noise multiplied to hidden representations. How and where to include perturbations and which forms of perturbations to apply to the fine-tuning of language models is worth studying and would be beneficial for advancing NLP frontiers.

Acknowledgments

This work was supported by Alibaba Group through Alibaba Research Intern Program.

References

Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. 2021. Better fine-tuning by reducing representational collapse. In *ICLR*.

Chris M. Bishop. 1995. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1):108–116.

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*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

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.

Sanyuan Chen, Yutai Hou, Yiming Cui, Wanxiang Che, Ting Liu, and Xiangzhan Yu. 2020. Recall and learn: Fine-tuning deep pretrained language models with less forgetting. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7870–7881, Online. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

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.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah A. Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *ArXiv*, abs/2002.06305.

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

Nadir Durrani, Hassan Sajjad, and Fahim Dalvi. 2021. How transfer learning impacts linguistic knowledge in deep NLP models? In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4947–4957, Online. Association for Computational Linguistics.

Kawin Ethayarajh. 2019. How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 55–65, Hong Kong, China. Association for Computational Linguistics.

Jun Gao, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tieyan Liu. 2019. Representation degeneration problem in training natural language generation models. In *International Conference on Learning Representations*.

Yichen Gong, Heng Luo, and Jian Zhang. 2018. Natural language inference over interaction space. In *International Conference on Learning Representations*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

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.

Hang Hua, Xingjian Li, Dejing Dou, Chengzhong Xu, and Jiebo Luo. 2021. Noise stability regularization for improving BERT fine-tuning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3229–3241, Online. Association for Computational Linguistics.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and efficient fine-tuning for pretrained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2177–2190, Online. Association for Computational Linguistics.

Tushar Khot, Ashish Sabharwal, and Peter Clark. 2018. Scitail: A textual entailment dataset from science question answering. In *AAAI*.

Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. 2020. Mixout: Effective regularization to finetune large-scale pretrained language models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Chen Liang, Haoming Jiang, Simiao Zuo, Pengcheng He, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Tuo Zhao. 2022. No parameters left behind: Sensitivity guided adaptive learning rate for training large transformer models. In *International Conference on Learning Representations*.

Xiaobo* Liang, Lijun* Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, and Tie-Yan Liu. 2021. R-drop: Regularized dropout for neural networks. In *NeurIPS*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).

Tiago Santana Nazaré, G. B. P. D. Costa, Welinton A. Contato, and Moacir P. Ponti. 2017. Deep convolutional neural networks and noisy images. In *CIARP*.

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. 2013a. 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, Seattle, Washington, USA. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013b. 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, Seattle, Washington, USA. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Yixuan Su, Fangyu Liu, Zaiqiao Meng, Tian Lan, Lei Shu, Ehsan Shareghi, and Nigel Collier. 2021. Tacl: Improving bert pre-training with token-aware contrastive learning.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Shoujie Tong, Qingxiu Dong, Damai Dai, Yifan song, Tianyu Liu, Baobao Chang, and Zhifang Sui. 2022. Robust fine-tuning via perturbation and interpolation from in-batch instances.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page III–1058–III–1066. JMLR.org.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Boxin Wang, Chejian Xu, Shuohang Wang, Shuohang Wang, Zhe Gan, Yu Cheng, Jianfeng Gao, Ahmed Awadallah, and Bo Li. 2021. Adversarial glue: A multi-task benchmark for robustness evaluation of language models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

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, New Orleans, Louisiana. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Chuhan Wu, Fangzhao Wu, Tao Qi, and Yongfeng Huang. 2022. NoisyTune: A little noise can help you finetune pretrained language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 680–685, Dublin, Ireland. Association for Computational Linguistics.

Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. 2021. Raise a child in large language model: Towards effective and generalizable fine-tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9514–9528, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Sangwon Yu, Jongyoon Song, Heeseung Kim, Seongmin Lee, Woo-Jong Ryu, and Sungroh Yoon. 2022. Rare tokens degenerate all tokens: Improving neural text generation via adaptive gradient gating for rare token embeddings. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 29–45, Dublin, Ireland. Association for Computational Linguistics.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. 2021. Revisiting few-sample bert fine-tuning. *ArXiv*, abs/2006.05987.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. 2019. Freelb: Enhanced adversarial training for language understanding. *CoRR*, abs/1909.11764.

A General Experiment Settings

On each experiment with each PLM, we run for three different random seeds for the averaged results and we grid search on learning rates of $\{1, 2, 3, 4\} \times 10^{-5}$ for the best results. Across different PLMs and tasks, we use AdamW (Loshchilov and Hutter, 2019) as the optimizer with Adam β of (0.9,0.99), Adam ϵ of 1×10^{-5} and 0.1 weight decay. For the learning rate scheduler, we use a linear decay scheme. We truncate all the inputs to a length of 128 tokens. In vanilla fine-tuning, we use 0.1 dropout rate. For HyPe-N and HyPe-U, we use the best results of the scale 10^{-4} and 10^{-5} and turn off dropout if not otherwise specified.

All our experiments are conducted on 32G NVIDIA V100 GPU in a single GPU setting.

B Experiments on GLUE

B.1 Data Introduction

Dataset	Train. Size	Dev. Size	Metric
MRPC	3.7k	408	F1
RTE	2.5k	277	Accuracy
STS-B	5.7k	1.5k	Pearson-Spearman Corr
CoLA	8.5k	1.0k	Matthew's Corr
QNLI	108k	5.7k	Accuracy
QQP	364k	40k	F1
SST2	67k	872	Accuracy
MNLI	393k	9.8k	Accuracy
MNLI-mm	-	9.8k	Accuracy

Table 10: The summary statistics of GLUE benchmark.

The summary statistics of GLUE and the reported evaluation metric is listed in Table 10. The license for GLUE is CC-BY-4.0.

B.2 Experiment Settings

For different fine-tuning techniques, we experiment with the same hyper-parameter setting, which are listed in Table 11.

B.3 GLUE Test Set Results

The conventional evaluation procedures of the previous research (R3F, RDrop, ChildTuning, NoisyTune) only report results on development sets of GLUE. Here we compare the vanilla fine-tuned results with HyPe fine-tuned results on test sets. Results listed in Table 12 show that on the averaged scores (column AVG.-ALL) of 8 GLUE tasks except WNLI and AX, HyPe-N and HyPe-U achieve 82.27 and 82.20 for BERT, as well as 85.71 and 86.02 for RoBERTa, which is obviously better than vanilla fine-tuning of 81.40 for BERT and 84.94 for RoBERTa. The improvements are more higher on 4 relatively small datasets (column AVG.), and HyPe-(N/U) achieves 2.30/1.90 and 1.17/1.92 for BERT and RoBERTa respectively. The results are consistent with those in Table 1 and 2 where HyPe can bring more performance gains on small data setting, since PLMs are prone to over-fitting more given small data.

C Generalization Probings

C.1 Dataset Introduction

The summary statistics of the NLI datasets SNLI, SICK and ScitaiL used in domain generalization probing experiments are presented in Table 13. The licenses for SICK and ScitaiL are CC-BY-NC-SA-3.0 and Apache-2.0 respectively.

C.2 Experiment Settings

Task Generalization We freeze the model parameters fine-tuned on SST2 except for fine-tuning a re-initialized linear head for each task. For each experiment, we use a learning rate of 0.001 for 3 epochs and batch size 16 for tuning the linear heads.

Domain Generalization We train on the subsets for 3 epochs with batch size 16. For different datasets we used, their label spaces are different as shown in Table 14. Therefore, we follow the experiment settings in Xu et al. (2021). Since SciTaiL only contains two labels `entailment` and `neutral` in their label spaces, we map the `contradiction` label in MNLI, MNLI-mm, SICK and SNLI to `neutral` to reduce their label space to `entailment` and `neutral`. For QQP, following Gong et al. (2018), we map `duplicate` to `entailment` and `not duplicate` to `contradiction`. With the above procedures, we create a consistent label space for each dataset to run evaluations. Besides, for some samples in SNLI, there exists no golden labels, and we filter them for training and evaluation. For the datasets used, we use their corresponding development sets for evaluation.

D With Other Techniques

D.1 Baseline Techniques

Different previously proposed effective fine-tuning techniques have exclusive hyper-parameters, we list the hyper-parameters we used in our re-implementation in Table 15. For each, we

Dataset	Batch Size	Update Steps	Warm-up Steps
BERT			
MRPC	16	3 epochs	10% of total steps
RTE	16	3 epochs	10% of total steps
STS-B	16	3 epochs	10% of total steps
CoLA	16	3 epochs	10% of total steps
RoBERTa			
MRPC	16	3 epochs	10% of total steps
RTE	16	3 epochs	10% of total steps
STS-B	16	3 epochs	10% of total steps
CoLA	16	3 epochs	10% of total steps
SST2	16	3 epochs	10% of total steps
QNLI	16	3 epochs	10% of total steps
QQP	16	3 epochs	10% of total steps
MNLI	16	3 epochs	10% of total steps
ELECTRA			
MRPC	32	3 epochs	10% of total steps
RTE	32	10 epochs	10% of total steps
STS-B	32	10 epochs	10% of total steps
CoLA	32	3 epochs	10% of total steps
XLNet			
MRPC	32	800 steps	200 steps
RTE	32	800 steps	200 steps
STS-B	32	3000 steps	500 steps
CoLA	64	1200 steps	120 steps

Table 11: Experiment settings used for different GLUE datasets and PLMs.

	CoLA	STS-B	MRPC	RTE	AVG.(Δ)	SST-2	QNLI	QQP	MNLI-m/mm	AVG.-ALL(Δ)
Vanilla	62.3	90.7	90.8	79.9	80.93(-)	96.6	91.9	73.3	89.6/89.3	84.94(-)
HyPe-N	65.5	90.9	91.0	81.0	82.10(+1.17)	96.5	94.1	73.0	89.8/89.6	85.71(+0.77)
HyPe-U	65.2	91.1	92.3	82.8	82.85(+1.92)	96.4	93.8	73.1	89.9/89.6	86.02(+1.08)

Table 12: Test set results on GLUE for RoBERTa-large. We use $\sigma = 10^{-5}$ for HyPe-N and HyPe-U.

Dataset	Train. Size	Dev. Size	Test Size	Metric
SNLI	550,152	10,000	10,000	Accuracy
ScitaiL	23,596	1,304	2,126	Accuracy
SICK	4,439	495	4,906	Accuracy

Table 13: The summary statistics of NLI datasets used in domain generalization probing experiments.

Dataset	Label Space
MNLI	entailment/neutral/contradiction
MNLI-mm	entailment/neutral/contradiction
SNLI	entailment/neutral/contradiction
SciTaiL	entailment/neutral
SICK	entailment/neutral/contradiction
QQP	duplicate/not duplicate

Table 14: The label spaces for datasets used in domain generalization experiments of Section 5.4.

follow the best settings reported in their papers. For ChildTuning, we use the Python code implementation from https://github.com/alibaba/AliceMind/tree/main/ChildTuning. For R-Drop, we use the implementation in https://github.com/dropreg/R-Drop. For R3F, we use the implementation from https://github.com/facebookresearch/fairseq/tree/main/examples/rxf. Note that in the original R3F implementation, they leave out STS-B task as this is a regression task and is not compatible with KL divergence. In our implementation, for STS-B task, we use mean squared error (MSE) in place of KL divergence for regulation.

D.2 Combination Experiments

We use the HyPe variant HyPe-N with scale $\sigma = 10^{-5}$ to integrate with others. When combining with Child-Tuning$_D$, we add HyPe to the forward computations. When combining with R3F, we use HyPe for the noised forward computation. When combining with R-Drop, we add HyPe to two forward computations in a training step with no dropout. When combining with NoisyTune, we add the noise to the parameters before fine-tuning with HyPe. For the combination experiments, we also search on the same ranges of hyper-parameters for the best result.

D.3 Detailed Results for Technique Combination

The detailed results for Figure 2 are listed in Table 16.

Technique	Hyper-parameters	Values
Child-Tuning$_F$	Gradient Mask Probability p	$\{0.2, 0.3, 0.4\}$
Child-Tuning$_D$	Gradient Mask Probability p	$\{0.1, 0.2, 0.3\}$
R-Drop	Regularization Weight α	$\{0.1, 0.5, 1.0\}$
R3F	Noise Distribution	$\mathcal{N}(0, \sigma^2)$
	Noise Scale σ	10^{-5}
	Regularization Weight λ	$\{0.1, 0.5, 1.0\}$
NoisyTune	Noisy Intensity λ	$\{0.1, 0.15, 0.2\}$

Table 15: The exclusive hyper-parameter settings for each baselines. For multiple values, we use the best results searched on these numbers.

Dataset	STS-B	COLA	MRPC	RTE	average	Δ
Detailed results on BERT						
RDrop	$90.29_{0.37}$	$65.06_{0.35}$	$91.84_{0.54}$	$75.21_{0.90}$	80.60	-
HyPe-N+RDrop	$90.45_{0.33}$	$65.23_{0.43}$	$91.80_{0.26}$	$75.93_{0.85}$	80.85	+0.25
R3F	$90.21_{0.56}$	$64.90_{1.50}$	$92.23_{0.67}$	$74.73_{2.41}$	80.52	-
HyPe-N+R3F	$90.36_{0.37}$	$65.58_{0.52}$	$91.82_{0.44}$	$75.57_{0.85}$	80.83	+0.31
Child-Tuning$_D$	$90.34_{0.55}$	$64.48_{1.29}$	$91.43_{0.24}$	$73.65_{0.51}$	79.97	-
HyPe-N+Child-Tuning$_D$	$90.75_{0.65}$	$65.18_{1.17}$	$91.77_{0.30}$	$74.01_{0.29}$	80.43	+0.46
NoisyTune	$90.22_{0.55}$	$64.67_{0.27}$	$91.46_{0.64}$	$73.89_{1.78}$	80.06	-
HyPe-N+NoisyTune	$90.37_{0.51}$	$65.12_{2.12}$	$91.45_{0.20}$	$73.65_{0.29}$	80.15	+0.09
Detailed results on RoBERTa						
RDrop	$92.26_{0.12}$	$67.03_{0.42}$	$93.03_{0.64}$	$85.56_{0.59}$	84.47	-
HyPe-N+RDrop	$92.34_{0.03}$	$68.77_{3.59}$	$93.21_{0.90}$	$85.20_{2.36}$	84.88	+0.41
R3F	$92.13_{0.08}$	$67.32_{1.72}$	$92.32_{0.68}$	$84.00_{1.62}$	83.94	-
HyPe-N+R3F	$92.29_{0.07}$	$68.25_{0.42}$	$92.64_{0.72}$	$85.80_{1.70}$	84.75	+0.81
Child-Tuning$_D$	$91.95_{0.15}$	$63.66_{0.71}$	$92.01_{0.77}$	$83.87_{3.97}$	82.87	-
HyPe-N+Child-Tuning$_D$	$92.05_{0.28}$	$67.38_{1.35}$	$92.31_{0.37}$	$84.12_{0.51}$	83.97	+1.10
NoisyTune	$92.07_{0.21}$	$66.15_{0.13}$	$92.31_{1.02}$	$85.20_{0.59}$	83.93	-
HyPe-N+NoisyTune	$92.34_{0.12}$	$67.71_{0.83}$	$93.09_{0.09}$	$85.44_{0.95}$	84.65	+0.72

Table 16: Detailed results of HyPe-N combining with other effective fine-tuning techniques. The standard deviations are shown in the subscripts.

E Experiment Details for advGLUE

AdvGLUE (Wang et al., 2021) contains the five adversarial perturbed datasets in GLUE which are SST-2, QQP, MNLI, RTE and QNLI. For MNLI there are MNLI-match and MNLI-mismatch. They use the original training data from the corresponding datasets in GLUE for model training. In our experiments, each results listed in Table 6 are averaged out of 3 random seed runs.

F Experiment Details for Parameter Scaling Experiments

When using vanilla fine-tuning schemes as settings listed in Table 11 will lead to corrupted and sub-optimal performances for DeBERTa. To reproduce a strong vanilla baseline for solid comparison, (1) we extend the training epochs to 6 and use a fixed warm-up step 100; (2) for MRPC, RTE and STS-B, we fine-tune based on MNLI-tuned models, which are `deberta-base-mnli`, `deberta-large-mnli`, `deberta-v2-xlarge-mnli` and `deberta-v2-xxlarge-mnli` from Huggingface repository, and for CoLA, we use the origin pre-trained versions , which are `deberta-base`, `deberta-large`, `deberta-v2-xlarge` and `deberta-v2-xxlarge` from Huggingface repository; (3) for the xlarge and xxlarge versions of DeBERTa's, we additionally search for best results on learning rates $\{1 \times 10^{-6}, 3 \times 10^{-6}, 5 \times 10^{-6}, 8 \times 10^{-6}\}$.

G More Discussions

G.1 Token Representation Similarity

As mentioned above in the generalization probing experiments, the representation abilities of hidden states are ameliorated. To further investigate how HyPe improves PLMs fine-tuning, we investigate the change of hidden representations. As illustrated by previous research (Ethayarajh, 2019; Gao et al., 2019), PLMs may suffer from the problem of anisotropic distribution of token representations (i.e., the representations only distributed in a narrow cone of the entire high-dimensional space). Research finds a correlation between isotropic distribution of representations and downstream performance (Su et al., 2021; Yu et al., 2022). Isotropic-

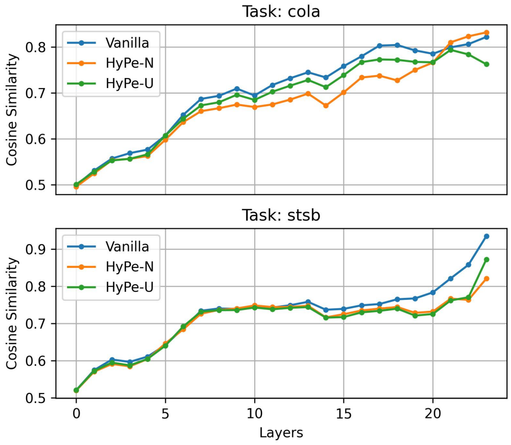


Figure 4: Token representation cosine similarity within samples across different layers. We use the fine-tuned RoBERTa checkpoints for all the results.

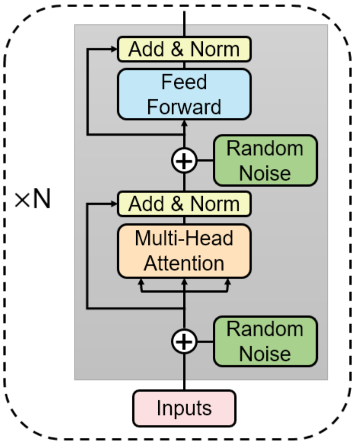


Figure 5: Besides adding noise to the hidden representations fed into Transformers layers, noise can also be added to between self-attention and feed-forward network within Transformers.

distributed hidden representation is a good property in terms of good representation abilities. Representation anisotropy can be accessed by calculating the token-wise cosine similarity within a sample. The lower similarity indicates a more isotropic distribution.

For the calculation of layer-wise token cosine similarity, we denote the index of each sample as i, the token index in each sample as j. The layer index is denoted as l. The calculation of similarity score $\mathcal{S}_i^l$ for layer l and sample i is:

$$\mathcal{S}_i^l = \frac{2}{n_i(n_i-1)} \sum_{1\leq j_a<j_b\leq n_i} \cos(h_{ij_a}^l, h_{ij_b}^l),$$

where n_i is the token count of sample i, h_{ij}^l stands for the hidden representation of token j in sample i in layer l and cos stands for the cosine similarity $\cos(q,p) = \frac{q^T p}{\|q\|\|p\|}$. Then the score is averaged over different samples:

$$\mathcal{S}^l = \frac{1}{M}\sum_{i=1}^{M} \mathcal{S}_i^l,$$

where M is the number of samples.

With isotropic distribution where similarity values are larger, transformers layers do not show degeneration and maintain good representation capacities. Hidden states may carry diverse useful information to each token in the next layer throught attention mechanism. We investigate the similarity to provide insight on how HyPe improve final results.

In Figure 4, we provide a line plot on how hidden presentation similarity varies across layers. For each point, the results are averaged across samples and 3 different runs. We can see that the anisotropic distribution problem gets severe for the higher layers. Models fine-tuned with HyPe have lower hidden representation similarity compared to vanilla fine-tuned PLMs on the top layers. For the lower layers, three lines are entangled, and this finding is consistent with that in Section 5.4.

It is worth noticing that for token similarity on CoLA, although HyPe-U has lower similarity on the last layer, while has lower performance than HyPe-N in Table 1. There may seem a contradiction between results. However, HyPe-N achieves better similarity on other higher layers. As HyPe is added to all different layers and information from intermediate layers influences that from the last layer, the results are also consistent.

In summary, inspired by previous research on interpreting PLMs, we empirically provide an insight that HyPe may improve fine-tuning by making hidden representations isotropic-distributed.

Adding noise after self-attentions. In HyPe, we add noise to the hidden representations between each Transformer layer, and compared to dropout, HyPe empirically shows better performance. These findings lead to this discussion of adding noise to the representations between self-attention and feed-forward network within Transformers layer like dropout, as illustrated in Figure 5. We run experiments on CoLA, STS-B, MRPC, and RTE with different schemes of adding noise. Experiments are conducted on BERT-large.

As shown in Table 17, in terms of average scores, HyPe-N with scale $\sigma = 10^{-5}$ (i.e., only adding noise between Transformers layers) shows the best performance, while adding noise only within Transformers shows the worst result among the three. When combining both positions to add noise, the performance shows no improvements on performances.

Adding noise to a subset of hidden representations. HyPe adds random noise to the hidden representations of all Transformers layers. We run further analyses by only adding noise to hidden representations fed into a subset of layers. We add normal noise with scale $\sigma = 10^{-5}$ to the hidden representations in the higher 6/12 layers and lower 6/12 layers of BERT-large. The higher layers mean the layers near the classifier head, while the lower layers mean the layers near the token embedding layer. As shown in Table 18, from the average scores across MRPC, STS-B, CoLA, and RTE datasets, we can conclude that 1) when adding noise on the higher layers is better than adding on the lower layers; 2) Noise added to more layers will obtain better performance.

	STS-B	CoLA	MRPC	RTE	AVG
HyPe	90.37	66.26	91.14	74.37	80.54
HyPe+Adding within Transformers	90.42	65.35	91.42	73.65	80.21
Adding within Transformers	90.54	65.53	91.59	71.84	79.88

Table 17: Results of analysis experiments on the distribution forms and scales of the noise.

	STS-B	CoLA	MRPC	RTE	AVG
Vanilla	90.07	63.63	90.67	72.24	79.31
HyPe on lower layers					
Lower 6 Layers	90.57	62.76	91.16	73.65	79.54
Lower 12 Layers	90.20	65.04	91.63	72.80	79.92
HyPe on higher layers					
Higher 6 Layers	90.25	64.37	91.36	73.65	79.90
Higher 12 Layers	90.27	64.36	91.53	74.73	80.22
HyPe	90.37	66.26	91.14	74.37	80.54

Table 18: HyPe noise added to hidden representations of different subsets of layers.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitation section in main texts

☑ A2. Did you discuss any potential risks of your work?
Ethic Statement and Broader Impact section in main texts

☑ A3. Do the abstract and introduction summarize the paper's main claims?
In Introduction and Abstract section in main texts

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 4.1 and Appendix B

☑ B1. Did you cite the creators of artifacts you used?
Section 4.1 and Appendix B C1

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Section 4.1 and Appendix B C1

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Section 4.1 and Appendix B C1

□ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Section 4.1 and Appendix B C1

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Section 4.1 and Appendix B C1

C ☑ Did you run computational experiments?

Section 4.1 and Appendix A

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Section 4.1 and Appendix A

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 4.1 and Appendix A

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Section 4, 5 Appendix A

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 4.3 and Appendix F

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

□ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

□ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

□ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

□ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

□ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.