R-Drop: Regularized Dropout for Neural Networks

Xiaobo Liang[1]* Lijun Wu[2]* Juntao Li[1] Yue Wang[1] Qi Meng[2]
Tao Qin[2] Wei Chen[2] Min Zhang[1] Tie-Yan Liu[2]
[1]Soochow University, [2]Microsoft Research Asia
xbliang3@stu.suda.edu.cn, {ljt,minzhang}@suda.edu.cn, wangyuenlp@gmail.com
{lijuwu,meq,taoqin,wche,tyliu}@microsoft.com

Abstract

Dropout is a powerful and widely used technique to regularize the training of deep neural networks. Though effective and performing well, the randomness introduced by dropout causes unnegligible inconsistency between training and inference. In this paper, we introduce a simple consistency training strategy to regularize dropout, namely R-Drop, which forces the output distributions of different sub models generated by dropout to be consistent with each other. Specifically, for each training sample, R-Drop minimizes the bidirectional KL-divergence between the output distributions of two sub models sampled by dropout. Theoretical analysis reveals that R-Drop reduces the above inconsistency. Experiments on $\mathbf{5}$ widely used deep learning tasks ($\mathbf{18}$ datasets in total), including neural machine translation, abstractive summarization, language understanding, language modeling, and image classification, show that R-Drop is universally effective. In particular, it yields substantial improvements when applied to fine-tune large-scale pre-trained models, e.g., ViT, RoBERTa-large, and BART, and achieves state-of-the-art (SOTA) performances with the vanilla Transformer model on WMT14 English→German translation ($\mathbf{30.91}$ BLEU) and WMT14 English→French translation ($\mathbf{43.95}$ BLEU), even surpassing models trained with extra large-scale data and expert-designed advanced variants of Transformer models. Our code is available at GitHub[2].

1 Introduction

In recent years, deep learning has achieved remarkable success in various areas, e.g., natural language processing, computer vision, speech/audio processing, etc. When training a deep neural network, regularization techniques [61, 64, 27, 3, 71, 62, 23, 75] are indispensable to prevent over-fitting and improve the generalization ability of deep models. Among them, the dropout technique [24], the most widely used one, aims to prevent co-adaptation and performs implicit ensemble by simply dropping a certain proportion of hidden units from the neural network during training. Existing literature [41, 81] has revealed the possible side effect of dropout that there is an unnegligible inconsistency between training and inference stage of dropout models, i.e., the randomly sampled sub model (caused by dropout) during training is inconsistent with the full model (without dropout) during inference. Through imposing L_2 regularization on the inconsistent hidden states [41, 81], current methods can mitigate the inconsistency problem to some extent but are far from being widely used.

In this paper, we introduce a simple yet more effective alternative to regularize the training inconsistency induced by dropout, named as R-Drop. Concretely, in each mini-batch training, each data sample goes through the forward pass twice, and each pass is processed by a different sub model by randomly dropping out some hidden units. R-Drop forces the two distributions for the same

*Equal contribution (listed in alphabetical order).
[2]https://github.com/dropreg/R-Drop

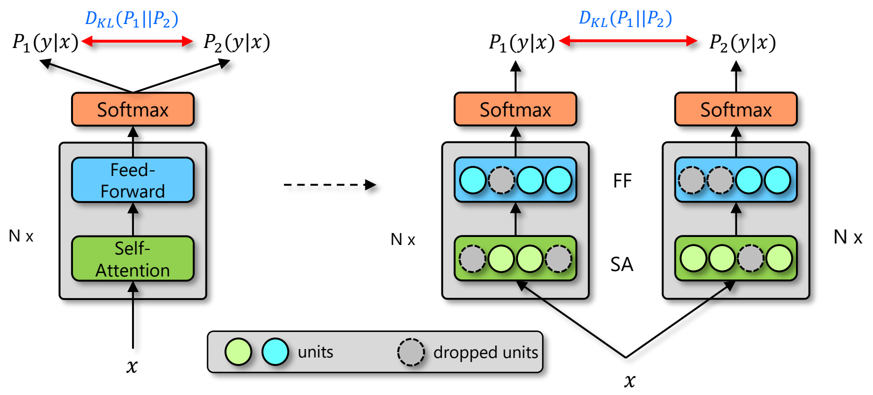

Figure 1: The overall framework of our proposed R-Drop. We take Transformer [63] structure for illustration. The left picture shows that one input x will go through the model twice and obtain two distributions $\mathcal{P}_1$ and $\mathcal{P}_2$, while the right one shows two different sub models produced by dropout.

data sample outputted by the two sub models to be consistent with each other, through minimizing the bidirectional Kullback-Leibler (KL) divergence between the two distributions. That is, R-Drop regularizes the outputs of two sub models randomly sampled from dropout for each data sample in training. In this way, the inconsistency between the training and inference stage can be alleviated. Compared with the dropout strategy in conventional neural network training, R-Drop only adds a KL-divergence loss without any structural modifications.

From the perspective of deep neural network regularization, our proposed R-Drop can be treated as a new variation of dropout. Different from most of the previous methods that merely work on the hidden units of each layer (e.g., the standard dropout [24]) or model parameters (e.g., dropconnect [64]), R-Drop works on both the hidden units and the output of sub models sampled by dropout, which is much more effective. We theoretically analyze the regularization effect of R-Drop, where the result shows that R-Drop can reduce the inconsistency existed in the training and inference.

Though R-Drop regularization is simple, we find it is surprisingly effective through extensive experiments on **5** tasks with **18** datasets, spanning from natural language processing, including language modeling, neural machine translation, abstractive summarization, and language understanding, to computer vision, i.e., image classification. It creates new records on multiple datasets, such as **30.91** BLEU score on WMT14 English$\rightarrow$German and **43.95** on WMT14 English$\rightarrow$French translation tasks while only be simply applied to the training of the vanilla Transformer, and also achieves SOTA results on the CNN/DailyMail summarization dataset. These universal improvements clearly demonstrate the effectiveness of R-Drop.

Our main contributions are summarized as follows:

- We propose R-Drop, a simple yet effective regularization method built upon dropout, which can be universally applied to train different kinds of deep models.
- We theoretically show that our R-Drop can reduce the inconsistency between training and inference of the dropout based models.
- Through extensive experiments on 4 NLP and 1 CV tasks with a total of 18 datasets, we show that R-Drop achieves extremely strong performances, including multiple SOTA results.

2 Approach

The overall framework of our R-Drop regularization method is shown in Figure 1. Before elaborating on the details, we first present some necessary notations. Given the training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, the goal of the training is to learn a model $\mathcal{P}^w(y|x)$, where n is the number of the training samples, (x_i, y_i) is the labeled data pair. x_i is input data and y_i is the label. For example, in NLP, x_i can be the source language sentence in machine translation, and y_i is the corresponding target language sentence. In CV, x_i can be one image, and y_i is the categorical class label. The probability distribution of the mapping function is also denoted as $\mathcal{P}^w(y|x)$, and the Kullback-Leibler (KL) divergence between two distributions $\mathcal{P}_1$ and $\mathcal{P}_2$ is represented by $\mathcal{D}_{KL}(\mathcal{P}_1||\mathcal{P}_2)$. In the following, we will explain our proposed R-Drop, training algorithm, and theoretical analysis, respectively.

2.1 R-Drop Regularization

We introduce our simple regularization method in this part. Given the training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, the main learning objective for a deep learning model is to minimize the negative log-likelihood loss function, which is as follow:

$$\mathcal{L}_{nll} = \frac{1}{n}\sum_{i=1}^{n} -\log \mathcal{P}^w(y_i|x_i). \tag{1}$$

Since the deep neural networks are prone to over-fitting, regularization methods such as dropout [61] are usually adopted during training to reduce the generalization error of the model. Specifically, dropout randomly drops part of units in each layer of the neural network to avoid co-adapting and over-fitting. Besides, dropout also approximately performs to combine exponentially many different neural network architectures efficiently [61], while model combination can always improve the model performance. Though simple and effective, there is a huge inconsistency between training and inference that hinders the model performance. That is, the training stage takes the sub model with randomly dropped units, while the inference phase adopts the full model without dropout. Also, the sub models caused by randomly sampled dropout units are also different without any constraints. Based on above observations and the randomness of the structure brought by dropout, we propose our R-Drop to regularize the output predictions of sub models from dropout.

Concretely, given the input data x_i at each training step, we feed x_i to go through the forward pass of the network twice. Therefore, we can obtain two distributions of the model predictions, denoted as $\mathcal{P}_1^w(y_i|x_i)$ and $\mathcal{P}_2^w(y_i|x_i)$. As discussed above, since the dropout operator randomly drops units in a model, the two forward passes are indeed based on two different sub models (though in the same model). As shown in the right part of Figure 1, the dropped units in each layer of the left path for the output prediction $\mathcal{P}_1^w(y_i|x_i)$ are different from that of the right path for output distribution $\mathcal{P}_2^w(y_i|x_i)$. Thus the distributions of $\mathcal{P}_1^w(y_i|x_i)$ and $\mathcal{P}_2^w(y_i|x_i)$ are different for the same input data pair (x_i, y_i). Then, at this training step, our R-Drop method tries to regularize on the model predictions by minimizing the bidirectional Kullback-Leibler (KL) divergence between these two output distributions for the same sample, which is:

$$\mathcal{L}_{KL}^i = \frac{1}{2}(\mathcal{D}_{KL}(\mathcal{P}_1^w(y_i|x_i)||\mathcal{P}_2^w(y_i|x_i)) + \mathcal{D}_{KL}(\mathcal{P}_2^w(y_i|x_i)||\mathcal{P}_1^w(y_i|x_i))). \tag{2}$$

With the basic negative log-likelihood learning objective $\mathcal{L}_{NLL}^i$ of the two forward passes:

$$\mathcal{L}_{NLL}^i = -\log \mathcal{P}_1^w(y_i|x_i) - \log \mathcal{P}_2^w(y_i|x_i), \tag{3}$$

the final training objective is to minimize $\mathcal{L}^i$ for data (x_i, y_i):

$$\begin{aligned}\mathcal{L}^i = \mathcal{L}_{NLL}^i + \alpha \cdot \mathcal{L}_{KL}^i =& -\log \mathcal{P}_1^w(y_i|x_i) - \log \mathcal{P}_2^w(y_i|x_i) \\ &+ \frac{\alpha}{2}[\mathcal{D}_{KL}(\mathcal{P}_1^w(y_i|x_i)||\mathcal{P}_2^w(y_i|x_i)) + \mathcal{D}_{KL}(\mathcal{P}_2^w(y_i|x_i)||\mathcal{P}_1^w(y_i|x_i))],\end{aligned} \tag{4}$$

where α is the coefficient weight to control $\mathcal{L}_{KL}^i$. In this way, our R-Drop further regularizes the model space beyond dropout and improves the generalization ability of a model. Compared Equation (1) with Equation (4), our R-Drop only adds a KL-divergence loss $\mathcal{L}_{KL}^i$ based on two forward passes in training. Note that our regularization methodology can be universally applied on different model structures if there exists randomness in a model (e.g., dropout) that can produce different sub models or outputs. We leave further explorations as future work.

2.2 Training Algorithm

The overall training algorithm based on our R-Drop is presented in Algorithm 1. As introduced before, at each training step, Line 3-4 show that we go forward the model and obtain output distributions $\mathcal{P}_1^w(y|x)$ and $\mathcal{P}_2^w(y|x)$, then Line 5-6 calculate the negative log-likelihood and the KL-divergence

Algorithm 1 R-Drop Training Algorithm

```
Input: Training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$.
Output: model parameter $w$.
1: Initialize model with parameters $w$.
2: while not converged do
3:     randomly sample data pair $(x_i, y_i) \sim \mathcal{D}$,
4:     repeat input data twice as $[x_i; x_i]$ and obtain the output distribution $[\mathcal{P}_1^w(y_i|x_i), \mathcal{P}_2^w(y_i|x_i)]$,
5:     calculate the negative log-likelihood loss $\mathcal{L}_{NLL}^i$ by Equation (3),
6:     calculate the KL-divergence loss $\mathcal{L}_{KL}^i$ by Equation (2),
7:     update the model parameters by minimizing loss $\mathcal{L}^i$ of Equation (4).
8: end while
```

between the two distributions. It is worth nothing that we do not forward the input data twice, instead, we repeat the input data x and concatenate them ($[x; x]$) in batch-size dimension, which can make forward procedure happen in the same mini-batch to save the training cost. Finally, the model parameters are updated (Line 7) according to the loss of Equation (4). The training will continue over the data epochs till convergence. Compared to the conventional training, our implementation is similar to enlarge the batch size to be double, and one potential limitation is that the computational cost of R-Drop increases at each step. As we show in Section 4.1, similar to other regularization methods (e.g., training w/ or w/o dropout), though R-Drop needs more training to converge, the final optimum is much better with a superior performance. We also show another study of baseline with doubled batch size in Appendix C.1.

2.3 Theoretical Analysis

We analyze the regularization effect of R-Drop in this subsection. Let $h^l(x) \in \mathbb{R}^d$ denote the output of the l-th layer of a neural network with input vector x, and let $\xi^l \in \mathbb{R}^d$ denote a random vector, each dimension of which is independently sampled from a Bernoulli distribution $B(p)$:

$$\xi_i^l = \begin{cases} 1, & \textit{with probability p}, \\ 0, & \textit{with probability 1-p}. \end{cases}$$

Then the dropout operation on $h^l(x)$ can be represented by $h_{\xi^l}^l(x) = \frac{1}{p}\xi^l \odot h^l(x)$, where $\odot$ denotes the element-wised product. Hence, the output distribution of the neural network with parameter w after applying dropout is $\mathcal{P}_\xi^w(y|x) := \texttt{softmax}(\texttt{linear}(h_{\xi^L}^L(\cdots(h_{\xi^1}^1(x_{\xi^0})))))$, where $\xi = (\xi^L, \cdots, \xi^0)$. The objective for R-Drop enhanced training can be formulated as solving the following constrained optimization problem:

$$\min_w \frac{1}{n}\sum_{i=1}^n \mathbb{E}_\xi[-\log \mathcal{P}_\xi^w(y_i|x_i)], \tag{5}$$

$$\textit{s.t.} \quad \frac{1}{n}\sum_{i=1}^n \mathbb{E}_{\xi^{(1)},\xi^{(2)}}[\mathcal{D}_{KL}(\mathcal{P}_{\xi^{(1)}}^w(y_i|x_i)||\mathcal{P}_{\xi^{(2)}}^w(y_i|x_i)))] \leq \epsilon. \tag{6}$$

More precisely, R-Drop optimizes the constrained optimization problem in Equation (5) and Equation (6) in a stochastic manner, i.e., it samples two random vectors $\xi^{(1)}$ and $\xi^{(2)}$ (corresponding to two dropout instantiations) from Bernoulli distribution and one training instance (x_i, y_i), and updates the parameter w according to the stochastic gradient $\nabla_w \mathcal{L}^i$ from Equation (4).

As we presented, one problem for dropout is the inconsistency between the training and inference models. Specifically, the training objective for dropout is the average loss of the sub models, i.e., $\min_w \frac{1}{n}\sum_{i=1}^n \mathbb{E}_\xi[-\log \mathcal{P}_\xi^w(y_i|x_i)]$, while the full model (denoted as $\bar{P}^w(y|x)$) is used for inference. Our proposed R-Drop enhanced training reduces this inconsistency by forcing the sub structures to be similar. The following proposition uses a linear model to demonstrate that with the constraint in Equation (6), the inconsistency gap between the average loss of sub structures and the loss of the full model can be bounded (detailed proof can be found in Appendix B).

Proposition 2.1. *For a linear model $\mathcal{P}^w(y|x) = \texttt{softmax}(\texttt{Norm}(w^T x))$ where $\texttt{Norm}(\cdot)$ denotes the normalization layer and $x \in \mathbb{R}^d$, with the constraint in Equation (6) in the main paper, we have*

$|\mathcal{L}_{nll}(w) - \mathbb{E}_{\xi}[\mathcal{L}_{nll}(w,\xi)]| \leq c\sqrt{\epsilon}$, *where* $\mathcal{L}_{nll}(w), \mathcal{L}_{nll}(w,\xi)$ *are the empirical loss calculated by the full model and a random sub model respectively,* c *is a constant related to the Liptschtz constant of the softmax operator.*

2.4 Discussion

The most related works with our R-Drop are ELD [41] and FD [81], which also study the consistency training with dropout. However, R-Drop has key differences with them. (1) The gap control is from different views. ELD works on directly reducing the gap between the sub model with dropout (train) and the expected full model without dropout (inference), while R-Drop and FD are both working on penalizing the discrepancy between the sub models, the superiority of regularizing the sub models has been proved in FD. (2) The regularization efficiency is different. ELD only back-propagates the gradients through sub model without the full model, which is less efficient than R-Drop that updates both sub models. (3) The regularization effect is different. Both ELD and FD use the L_2 distance on hidden states as the regularization loss function. However, this is far away from the main training objective that minimizes the negative log-likelihood over model output distribution. The distance of hidden states is not in the same space as the probability distribution since log-softmax hugely affects the optimization. In comparison, R-Drop utilizes the KL-divergence between the output probability distributions as the consistency regularization, which is in the same space as the training objective. More analysis and experimental comparisons are shown in Appendix C.4.

3 Experiments

To evaluate our approach and show its universal impact, we conduct experiments on 5 different tasks, including 4 natural language processing (NLP) and 1 computer vision (CV) tasks, which are neural machine translation (NMT) (6 datasets), abstractive summarization (1 dataset), language understanding (8 datasets), language modeling (1 dataset), and image classification (2 datasets). For convenience, we utilize ‘RD’ to represent R-Drop in the tables of experimental results hereinafter. More details of experimental settings for each dataset can be found in Appendix A.

3.1 Application to Neural Machine Translation

Model	En→De	De→En	En→Fr	Fr→En	En→Zh	Zh→En	En→Es	Es→En	Avg
Transformer [63]	28.57	34.64	35.9	36.1	26.3	18.4	39.0	40.6	32.44
Transformer + RD	**30.72**	**37.25**	**38.0**	**38.9**	**28.1**	**19.5**	**41.8**	**43.2**	**34.68**

Table 1: BLEU scores on 8 IWSLT machine translation tasks.

We first evaluate the NMT tasks, which is very important in NLP. To best show the effectiveness of our method, experiments are conducted on both low-resource and rich-resource translation tasks.

Datasets The datasets of low-resource scenario are from IWSLT competitions, which include IWSLT14 English↔German (En↔De), English↔Spanish (En↔Es), and IWSLT17 English↔French (En↔Fr), English↔Chinese (En↔Zh) translations. The rich-resource datasets come from the widely acknowledged WMT translation tasks, and we take the WMT14 English→German and English→French tasks. The IWSLT datasets contain about $170k$ training sentence pairs, $7k$ valid pairs, and $7k$ test pairs. The WMT data sizes are $4.5M$, $36M$ for En→De and En→Fr respectively, valid and test data are from the corresponding newstest data.

Method	En→De	En→Fr
Transformer [63]	29.12	42.69
MUSE [77]	29.90	43.50
Depth Growing [70]	30.07	43.27
Transformer-Admin [38]	30.10	43.80
Data Diversification [48]	30.70	43.70
BERT-fused NMT [80]	30.75	43.78
Transformer + RD	**30.91**	**43.95**

Table 2: BLEU scores on WMT14 En→De and En→Fr machine translation tasks.

Model & Training We take the most popular Transformer [63] network as our model structure. The `transformer_iwslt_de_en` and `transformer_vaswani_wmt_en_de_big` are the configurations for IWSLT and WMT translations respectively. The weight α is set as 5 for all translation tasks. Implementation is developed on Fairseq [50].

Results We calculate the BLEU scores on these tasks for evaluation, following [80]. The IWSLT performances are shown in Table 1 and the rich-resource WMT results are in Table 2. First, we can see that our R-Drop achieves more than 2.0 BLEU score improvements on 8 IWSLT translation tasks, which clearly shows the effectiveness of our method. The results on WMT translations are more impressive. After applying our simple method on the basic Transformer network, we achieve the state-of-the-art (**SOTA**) BLEU score on WMT14 En$\rightarrow$De ($\mathbf{30.91}$) and En$\rightarrow$Fr ($\mathbf{43.95}$) translation tasks, which surpass current SOTA models, such as the BERT-fused NMT [80] model that leverages large-scale monolingual data, and the Data Diversification [48] method trained with many translation models. Note that R-Drop is complementary to the above methods, and we believe stronger results can be achieved if we apply R-Drop on their methods and better backbone models beyond Transformer.

3.2 Application to Language Understanding

Dataset We further evaluate our proposed approach on the language understanding tasks by fine-tuning the pre-trained models[3], which are the standard development sets of GLUE [65] benchmark. The GLUE benchmark includes 8 different text classification or regression tasks, which are MNLI, MRPC, QNLI, QQP, RTE, SST-2, STS-B (regression), CoLA. The detailed statistics are in Appendix.

Model & Training We take the BERT-base [9] and strong RoBERTa-large [39] pre-trained models as our backbones to perform fine-tuning, which are publicly available. For each task, different random seeds and parameter settings are required, thus we dynamically adjust the coefficient α among $\{0.1, 0.5, 1.0\}$ for each setting. Other configurations are following the previous works [9, 39]. For the regression task STS-B, we use MSE instead of KL-divergence to regularize the outputs (see Appendix for MSE regularization details).

Results The evaluation metrics for above 8 tasks are as follows: The result for STS-B is the Pearson correlation; Matthew's correlation is used for CoLA; Other tasks are measured by Accuracy. The results are presented in Table 3. We can see that R-Drop achieves 1.21 points and 0.80 points (on average) improvement over the two baselines BERT-base and RoBERTa-large, respectively, which clearly demonstrate the effectiveness of R-Drop. Specifically, our RoBERTa-large + RD also surpasses the other two strong models: XLNet-large [72] and ELECTRA-large [7], which are specially designed with different model architecture and pre-training task.

Model	**MNLI**	**MRPC**	**QNLI**	**QQP**	**RTE**	**SST-2**	**STS-B**	**CoLA**	**Avg**
BERT-base [9]	83.8	85.3	90.8	91.0	68.2	92.4	89.3	62.3	82.85
BERT-base + RD	85.5	87.3	92.0	91.4	71.1	93.0	89.6	62.6	**84.06**
RoBERTa-large [39]	90.2	90.9	94.7	92.2	86.6	96.4	92.4	68.0	88.93
XLNet-large [72]	90.8	90.8	94.9	92.3	85.9	97.0	92.5	69.0	89.15
ELECRTA-large [7]	90.9	90.8	95.0	92.4	88.0	96.9	92.6	69.1	89.46
RoBERTa-large + RD	90.9	91.4	95.2	92.5	88.4	96.9	92.5	70.0	**89.73**

Table 3: Fine-tuned model performances on GLUE language understanding benchmark.

3.3 Application to Summarization

Dataset Abstractive summarization task is to summarize the long sentence/document into a short sequence/sentence (through generation) with the main content remained. For this generation task, we use the CNN/Daily Mail dataset originally introduced by Hermann et al. [22] to evaluate our method. This dataset contains news documents (source), and their corresponding highlights (target) crawled from CNN and Daily Mail website. It contains 287,226 documents for training, 13,368 documents for validation and 11,490 documents for test. We follow [35] to preprocess the dataset.

Model & Training To mostly show the effectiveness, we take the super strong pre-trained sequence-to-sequence BART [35] model as our backbone and fine-tune it using our method. In this task, the coefficient weight α is set as 0.7 to control the KL-divergence. For other hyper-parameters, we follow the setting of the original paper [35] without modification.

[3]We apply our R-Drop on the fine-tuning stage only in this work. R-Drop can also be applied during pre-training. Due to the computational cost, we leave this as future work.

Results The performance is evaluated by ROUGE F1 score [37]. Specifically, we report the unigram ROUGE-1 (RG-1) and bigram ROUGE-2 (RG-2) overlap to assess the informativeness, and the longest common subsequence ROUGE-L (RG-L) score to assess the fluency. The results are shown in Table 4. We can see that R-Drop based training outperforms the fine-tuned BART model by 0.3 points on RG-1 and RG-2 score and achieves the **SOTA** performance. Specifically, our result also surpasses the PEGASUS method [74], which brings a novel self-supervised paradigm carefully designed for summarization, and the previous best work BART+R3F [1], which introduces a parametric noise sampled from normal or uniform distributions. Instead, our R-Drop does not introduce any extra parameters or model structure changes during training.

Method	RG-1	RG-2	RG-L
Transformer [63]	39.50	16.06	36.63
ProphetNet [54]	44.02	21.17	**41.30**
BART [35]	44.16	21.28	40.90
PEGASUS [74]	44.17	21.47	41.11
BART + R3F [1]	44.38	21.53	41.17
BART + RD	**44.51**	**21.58**	41.24

Table 4: ROUGE results on CNN/Daily Mail summarization dataset. RG-1, RG-2, RG-L stand for ROUGE-1, ROUGE-2, and ROUGE-L scores.

3.4 Application to Language Modeling

Dataset We also evaluate our approach on another widely acknowledged NLP task: language modeling. The dataset we choose for this task is the commonly adopted Wikitext-103 dataset [42], which is the largest available word-level language modeling benchmark with long-term dependency. WikiText-103 contains about $103M$ training tokens from $28K$ articles on Wikipedia, and the average length of tokens per article is about $3.6K$. The data is preprocessed by following [50].

Model & Training We take two models to conduct the language modeling task. One is the basic Transformer decoder [63], another is the more advanced one: Adaptive Input Transformer [5], which introduces adaptive input embeddings into the Transformer model. We use the open-source Fairseq [50] toolkit, and the corresponding model configurations are `transformer_lm_gpt` and `transformer_lm_wiki103` for Transformer and Adaptive Input Transformer. We simply set the weight α to be 1.0 without tuning during training. Other configurations are same as [50] and [5].

Results The evaluation metric for language modeling is perplexity, which can well measure the probability of a sentence. Same as [5], we report the perplexity on both valid and test sets. The results are shown in Table 5. From the table, we can see that our R-Drop based training improves the perplexity on both two different model structures, e.g., 0.80 perplexity improvement on test set over Adaptive Input Transformer. Besides, more improvement can be achieved when the baseline model is not so strong, e.g., 1.79 perplexity gain on valid set and 1.68 on test set above the Transformer baseline.

Method	Valid	Test
Transformer [63]	25.76	26.62
Transformer + RD	**23.97**	**24.94**
Adaptive [5]	18.94	18.87
Adaptive + RD	**18.18**	**18.07**

Table 5: Perplexity results on Wikitext-103 language modeling task. Adaptive refers to Adaptive Input Transformer [5].

3.5 Application to Image Classification

Dataset For image classification, we conduct experiments on two widely acknowledged benchmark datasets, i.e., CIFAR-100 [32] and the ILSVRC-2012 ImageNet dataset [8] (denoted as ImageNet for short). CIFAR-100 dataset consists of $60k$ images of 100 classes, and there are 600 images per class with 500 for training and 100 for testing.

The ImageNet dataset consists of $1.3M$ image samples of $1,000$ categorical classes. We utilize the same data preprocessing strategies with [11], where the details are given in [30].

Model & Training We choose the recent strong and popular Vision Transformer (ViT) [11] model as our backbone. More specifically, we take the two publicly released pre-trained models, ViT-B/16 and ViT-L/16, with $86M$ and $307M$ parameters respectively, and we

Method	CIFAR-100	ImageNet
ViT-B/16 [11]	92.64	83.97
ViT-B/16 + RD	**93.29**	**84.38**
ViT-L/16 [11]	93.44	85.15
ViT-L/16 + RD	**93.85**	**85.57**

Table 6: Accuracy on CIFAR-100 and ImageNet classification tasks.

conduct model fine-tuning on the CIFAR-100 and ImageNet datasets. During fine-tuning, the weight α is set as 0.6 for both models, and we set other hyper-parameters/training details to be same as [11].

Results The classification performance is measured by Accuracy, and the results are presented in Table 6. For CIFAR-100, we achieve about 0.65 accuracy improvement over ViT-B/16 baseline, and 0.41 points over ViT-L/16 model. Similarly, on the large-scale ImageNet dataset, consistent improvements are also obtained. These observations demonstrate that our R-Drop can still benefit the model performance even the baseline is powerful. In a word, through the above NLP tasks and this image classification task, we clearly show R-Drop is effective and can be universally applied.

4 Study

Beyond the superior experimental results, in this section, we conduct extensive studies on different perspectives to better understand our R-Drop method. The analysis experiments are performed on the IWSLT14 De$\rightarrow$En translation task. More studies can be found in Appendix C.

4.1 Regularization and Cost Analysis

We first show the regularization effect of our R-Drop and study the potential limitation of training cost (as discussed in Section 2.2). Hence, we plot the curves of training/valid loss and valid BLEU along the training update number for Transformer and Transformer + RD models. Besides, we also plot the corresponding curves along the training time (minutes). The curves are shown in Figure 2. We can observe: 1) Along with the training, Transformer quickly becomes over-fitting, and the gap between train and valid loss of Transformer is large, while R-Drop has a lower valid loss. This well proves that R-Drop can provide persistent regularization during training. 2) At the early training stage, Transformer improves the BLEU score quickly but converges to bad local optima soon. In comparison, R-Drop gradually improves the BLEU score and achieves a much superior performance. Though it needs more training to converge, the final optimum is better. This is same as other regularization methods (e.g., training w/ or w/o dropout). R-Drop indeed increases the training cost at each step since it requires repeating input x for another computation in a mini-batch. Note that this is similar to batch size doubled training without KL-divergence. In Appendix C.1, we conduct this training and show that R-Drop increases negligible cost but with a much stronger performance.

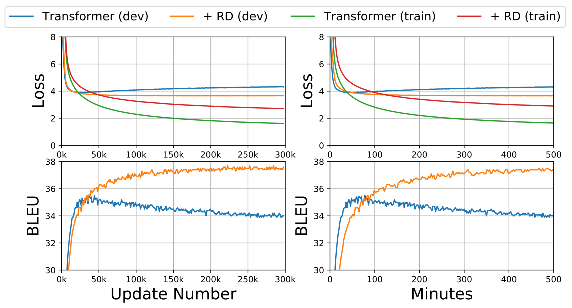

Figure 2: Loss/BLEU curves along with model training.

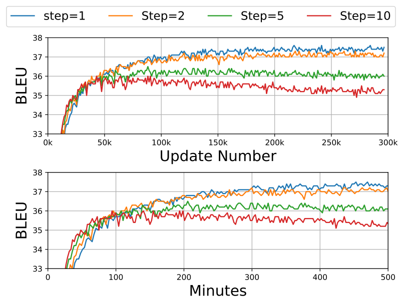

Figure 3: R-Drop with different step.

4.2 k-step R-Drop

The above study shows that R-Drop can achieve much stronger performance, but with a lower convergence, thus we study another training strategy that is to perform R-Drop every k steps to improve the training efficiency, instead of applying at each step. We vary k in $\{1, 2, 5, 10\}$ to see the difference, where $k = 1$ is the current training strategy. The valid BLEU curves along with training update number and training time are presented in Figure 3. From the curves, we can conclude that though the convergence is faster with larger k, the training fails to fall into good optima, which quickly over-fits, and the BLEU scores become worse and worse when we increase k. This proves that our R-Drop at each step can well regularize the training and obtain superior performances.

4.3 m-time R-Drop

Our method regularizes the model output between two distributions $P_1^w(y|x)$ and $P_2^w(y|x)$, and it is also interesting to see whether more improvements can be achieved if we regularize m distributions for the same input data, where $m = 2$ is the current setting. Therefore, we extend our R-Drop to be: $\mathcal{L}_{KL} = \frac{\alpha}{m*(m-1)} \sum_{i,j\in 1,\cdots,m}^{i\neq j} \mathcal{D}_{KL}(\mathcal{P}_i^w(y|x)||\mathcal{P}_j^w(y|x))$, and we take $m = 3$ for a feasible implementation. The BLEU score for IWSLT14 De→En test set is 37.30 when $m = 3$, which is similar to that when $m = 2$ (37.25 BLEU score). This reflects that R-Drop already has a strong regularization effect between two distributions, without the necessity of stronger regularization.

4.4 Two Dropout Rates

Besides the above studies, we investigate R-Drop from another perspective, i.e., the dropout values. In current training, the two distributions are based on the same dropout value (e.g., 0.3 for IWSLT translations). In this study, we utilize two different dropout values for the two output distributions during training (e.g., 0.1 for $P_1^w(y|x)$, 0.3 for $P_2^w(y|x)$) to see the difference. We choose the two dropout rates from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ with total 15 (C_5^2 for two different rates + C_5^1 for two same rates) combinations. The results are shown in Figure 4. Among these different results, we can see that: 1) Dropout rates with the same value $(0.3, 0.3)$ is the best choice (current setting), 2) R-Drop can stably achieve strong results when the two dropout rates are in a reasonable range ($0.3 \sim 0.5$) without a big performance difference. One interesting point is that even the two dropout values are both 0.5, which means half of the units are expected to be dropped, R-Drop can still obtain a satisfied result (36.48 BLEU) compared with the baseline Transformer (34.64 BLEU). These results all confirm the advantage of our R-Drop, and we are interested in studying more in the future.

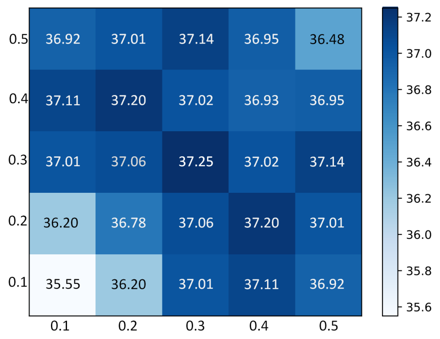

Figure 4: R-Drop with two different dropout rate combinations. Among the 25 numbers, 15 are different since the table is symmetric and triangular.

4.5 Effect of Weight α

Further, we investigate the impact of the KL-divergence loss weight α. As mentioned in Section 3.1, we set $\alpha = 5$ for NMT experiments. Here we vary the α in $\{1, 3, 5, 7, 10\}$ and conduct experiments. As shown in Table 7, small α (e.g., 1) can not perform as good as large α (e.g., 5), which means we should pay more attention to the KL-divergence regularization. However, too much regularization ($\alpha = 10$) is also not good, and the best balanced choice is $\alpha = 5$. Note that the choice of α is distinct for different tasks (e.g., NMT, language understanding), which depends on how easy the over-fitting happens caused by the specific data size and model size of each task.

α	**TF + RD**
$\alpha = 1$	36.05
$\alpha = 3$	36.85
$\alpha = 5$	**37.25**
$\alpha = 7$	37.20
$\alpha = 10$	36.95

Table 7: BLEU scores with different α.

5 Related Work

Regularization Methods. Bigger models always tend to have better performance, especially for various large-scale pre-trained models, e.g., Vision Transformer [11], Swin Transformer [40], GPT families [55, 56, 6], BERT [9], BART [35], Switch Transformers [14], etc. With millions and even billions of parameters, these deep models are prone to over-fitting, thus requiring regularization strategies to improve their generalization ability [34]. To tackle with over-fitting, many regularization techniques have been proposed, e.g., weight decay [33, 31, 28, 67], dropout [24, 64, 4, 66, 61], normalization [27, 57, 3, 26, 71], adding noise [25, 52], layer-wise pre-training and initialization [12, 21], label-smoothing [62], and so on. Among which, dropout and its variants are most popular owing to its effectiveness and moderate cost as well as good compatibility with other regularization methods [46], which has been successfully applied to regularize a wide range of neural network architectures [51], e.g., convolutional neural network layers [68, 10], recurrent neural networks [17, 58, 43], Transformer [73, 79, 69]. The success of dropout methods can be interpreted by preventing co-adaptation of neurons and performing an implicit ensemble of sub models from dropout. Owing to the effect in promoting sparsity of weights and stochastic nature, dropout methods are also adapted to other applications, e.g., contrastive learning for sentence representation learning [18], neural network compression [45, 47] and model uncertainty estimation [16].

Unlike previous researches of designing specific dropout variants or adapting dropout to different applications, we consider to further regularize the model on the success of dropout. Specifically, any two sub models sampled from dropout are encouraged to produce consistent model prediction for an input data by utilizing KL-divergence in the training stage. That is, we conduct regularization on the model output level. In doing so, the sub model outputs produced by the randomness of dropout are regularized to reduce the parameter freedom, which will enhance generalization in inference.

Consistency Training. Besides regularization methods, our work also relates to a few works of consistency training on dropout models or data augmentation. Among them, the most representative methods are ELD [41], FD [81], and Cutoff [60]. As discussed in Section 2.4, ELD only focuses on the inconsistency between the sub model with dropout (train) and the expected full-model without dropout (inference), while FD works between the sub models only (consistence between two sub models). Both ELD and FD utilize L_2 to regularize the hidden space. Instead, our R-Drop performs consistency training on dropout from the output space with a more effective bidirectional KL loss. Unlike the above consistency training method on sub models, Cutoff resembles launching consistency training from a data perspective by regularizing the inconsistency between the original data the augmented samples with part of the information within an input sentence being erased.

Self-distillation. Minimizing the KL-divergence between the output distributions of two different models correlates with knowledge distillation [23, 15, 2, 36, 13, 78], where the two models refer to teacher and student, respectively. In our setting, the teacher and student are the dropout instantiations of the same model, thus it resembles self-knowledge distillation [44] scenario. Different from existing method that exploits dark knowledge from the model itself [20, 19] or distills knowledge between different layers [75], our strategy can be regarded as an instance-wise self-knowledge distillation, i.e., each pair of sampled sub models perform distillation between each other for the same input, which also relates to mutual learning [76] but ours is much more efficient without extra parameters.

6 Conclusions and Future Work

In this paper, we proposed a simple yet very effective consistency training method built upon dropout, namely R-Drop, which minimizes the bidirectional KL-divergence of the output distributions of any pair of sub models sampled from dropout in model training. Experimental results on 18 popular deep learning datasets show that not only can our R-Drop effectively enhance strong models, e.g., ViT, BART, Roberta-large, but also work well on large-scale datasets and even achieve SOTA performances when combined with vanilla Transformer on WMT14 English$\to$German and English$\to$French translations. Due to the limitation of computational resources, for pre-training related tasks, we only tested R-Drop on downstream task fine-tuning in this work. We will test it on pre-training in the future. In this work, we focused on Transformer based models. We will apply R-Drop to other network architectures such as convolutional neural networks.

Acknowledgments and Disclosure of Funding

We would like to thank the reviewers for their constructive comments. Juntao Li is the corresponding author. This work was supported by the National Science Foundation of China (NSFC No. 62036004).

References

[1] Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. Better fine-tuning by reducing representational collapse. *arXiv preprint arXiv:2008.03156*, 2020.

[2] Zeyuan Allen-Zhu and Yuanzhi Li. Towards understanding ensemble, knowledge distillation and self-distillation in deep learning. *arXiv preprint arXiv:2012.09816*, 2020.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Lei Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*, pp. 3084–3092, 2013.

[5] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations*, 2018.

[6] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[7] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2019.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pp. 4171–4186, 2019.

[10] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[12] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Artificial Intelligence and Statistics*, pp. 153–160. PMLR, 2009.

[13] Zhiyuan Fang, Jianfeng Wang, Lijuan Wang, Lei Zhang, Yezhou Yang, and Zicheng Liu. Seed: Self-supervised distillation for visual representation. *International Conference on Learning Representations*, 2021.

[14] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[15] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *International Conference on Machine Learning*, pp. 1607–1616. PMLR, 2018.

[16] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

[17] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. *Advances in neural information processing systems*, 29:1019–1027, 2016.

[18] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*, 2021.

[19] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. In *International Conference on Learning Representations*, 2019.

[20] Sangchul Hahn and Heeyoul Choi. Self-knowledge distillation in natural language processing. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pp. 423–430, 2019.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[22] Karl Moritz Hermann, Tomáš Kočiskỳ, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *arXiv preprint arXiv:1506.03340*, 2015.

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Advances in neural information processing systems*, pp. 529–536, 1995.

[26] Lei Huang, Xianglong Liu, Bo Lang, Adams Yu, Yongliang Wang, and Bo Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

[28] Guoliang Kang, Jun Li, and Dacheng Tao. Shakeout: A new regularized deep neural network training scheme. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. *arXiv preprint arXiv:1912.11370*, 6(2):8, 2019.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[32] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[33] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pp. 950–957, 1992.

[34] Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of dropout methods for deep neural networks. *arXiv preprint arXiv:1904.13310*, 2019.

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

[36] Kevin J Liang, Weituo Hao, Dinghan Shen, Yufan Zhou, Weizhu Chen, Changyou Chen, and Lawrence Carin. Mixkd: Towards efficient distillation of large-scale language models. *International Conference on Learning Representations*, 2021.

[37] Chin-Yew Lin and Eduard Hovy. Manual and automatic evaluation of summaries. In *Proceedings of the ACL-02 Workshop on Automatic Summarization*, pp. 45–51, 2002.

[38] Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. Very deep transformers for neural machine translation. *arXiv preprint arXiv:2008.07772*, 2020.

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

[40] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[41] Xuezhe Ma, Yingkai Gao, Zhiting Hu, Yaoliang Yu, Yuntian Deng, and Eduard Hovy. Dropout with expectation-linear regularization. *arXiv preprint arXiv:1609.08017*, 2016.

[42] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[43] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. In *International Conference on Learning Representations*, 2018.

[44] Hossein Mobahi, Mehrdad Farajtabar, and Peter L Bartlett. Self-distillation amplifies regularization in hilbert space. *arXiv preprint arXiv:2002.05715*, 2020.

[45] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pp. 2498–2507. PMLR, 2017.

[46] Reza Moradi, Reza Berangi, and Behrouz Minaei. A survey of regularization strategies for deep models. *Artificial Intelligence Review*, 53(6):3947–3986, 2020.

[47] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6778–6787, 2017.

[48] Xuan-Phi Nguyen, Shafiq Joty, Kui Wu, and Ai Ti Aw. Data diversification: A simple strategy for neural machine translation. In *Advances in Neural Information Processing Systems*, pp. 10018–10029, 2020.

[49] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pp. 1–9, 2018.

[50] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *NAACL-HLT (Demonstrations)*, 2019.

[51] Hieu Pham and Quoc V Le. Autodropout: Learning dropout patterns to regularize deep networks. *arXiv preprint arXiv:2101.01761*, 2021.

[52] Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831*, 2014.

[53] Matt Post. A call for clarity in reporting bleu scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*. Association for Computational Linguistics, 2018.

[54] Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pp. 2401–2410, 2020.

[55] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training.

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

[57] Tim Salimans and Diederik P Kingma. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 901–909, 2016.

[58] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pp. 1757–1766, 2016.

[59] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, 2016.

[60] Dinghan Shen, Mingzhi Zheng, Yelong Shen, Yanru Qu, and Weizhu Chen. A simple but tough-to-beat data augmentation approach for natural language understanding and generation. *arXiv preprint arXiv:2009.13818*, 2020.

[61] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[62] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6000–6010, 2017.

[64] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pp. 1058–1066. PMLR, 2013.

[65] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 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*, pp. 353–355, 2018.

[66] Sida Wang and Christopher Manning. Fast dropout training. In *international conference on machine learning*, pp. 118–126. PMLR, 2013.

[67] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 2082–2090, 2016.

[68] Haibing Wu and Xiaodong Gu. Towards dropout training for convolutional neural networks. *Neural Networks*, 71:1–10, 2015.

[69] Hongqiu Wu, Hai Zhao, and Min Zhang. Not all attention is all you need. *arXiv preprint arXiv:2104.04692*, 2021.

[70] Lijun Wu, Yiren Wang, Yingce Xia, Fei Tian, Fei Gao, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. Depth growing for neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5558–5563, 2019.

[71] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

[72] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems*, 32:5753–5763, 2019.

[73] Lin Zehui, Pengfei Liu, Luyao Huang, Junkun Chen, Xipeng Qiu, and Xuanjing Huang. Dropattention: A regularization method for fully-connected self-attention networks. *arXiv preprint arXiv:1907.11065*, 2019.

[74] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pp. 11328–11339. PMLR, 2020.

[75] Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3713–3722, 2019.

[76] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4320–4328, 2018.

[77] Guangxiang Zhao, Xu Sun, Jingjing Xu, Zhiyuan Zhang, and Liangchen Luo. Muse: Parallel multi-scale attention for sequence to sequence learning. *arXiv preprint arXiv:1911.09483*, 2019.

[78] Helong Zhou, Liangchen Song, Jiajie Chen, Ye Zhou, Guoli Wang, Junsong Yuan, and Qian Zhang. Rethinking soft labels for knowledge distillation: A bias-variance tradeoff perspective. *International Conference on Learning Representations*, 2021.

[79] Wangchunshu Zhou, Tao Ge, Furu Wei, Ming Zhou, and Ke Xu. Scheduled drophead: A regularization method for transformer models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pp. 1971–1980, 2020.

[80] Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tieyan Liu. Incorporating bert into neural machine translation. In *International Conference on Learning Representations*, 2019.

[81] Konrad Zolna, Devansh Arpit, Dendi Suhubdy, and Yoshua Bengio. Fraternal dropout. 2018.

A Detailed Experimental Settings

We provide more detailed settings for the experiments of each task in this part.

A.1 Neural Machine Translation

For all the NMT tasks, we use the public datasets from IWSLT competitions[4] and WMT competitions[5]. We tokenize all the datasets with byte-pair-encoding (BPE) [59] approach with the dictionary built jointly upon the source and target sentence pairs except the IWSLT17 En$\leftrightarrow$Zh translation dataset that is built separately. After tokenization, the resulted vocabularies for IWSLT datasets are near $10k$, while for WMT datasets, the vocabulary size is about $32k$.

To train the Transformer based NMT models, we use `transformer_iwslt_de_en` configuration for IWSLT translations, which has 6 layers in both encoder and decoder, embedding size 512, feed-forward size $1,024$, attention heads 4, dropout value 0.3, weight decay 0.0001. For the WMT experiments, the `transformer_vaswani_wmt_en_de_big` setting has 6 layers in encoder and decoder, embedding size $1,024$, feed-forward size $4,096$, attention heads 16, dropout value 0.1, attention dropout 0.1 and relu dropout 0.1. The training is optimized with Adam [29] with $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9}$. The learning rate scheduler is `inverse_sqrt` with default learning rate 0.0005 and warmup steps $4,000$. Label smoothing [62] is adopted with value 0.1. Our code implementation is based on open-source Fairseq[6]. We train the IWSLT translations on 1 GEFORCE RTX 3090 card and the WMT translations on 8 GEFORCE RTX 3090 cards.

To evaluate the performance, we use `multi-bleu.perl`[7] to evaluate IWSLT14 En$\leftrightarrow$De and all WMT tasks for a fair comparison with previous works [80, 49]. For other NMT tasks, we use `sacre-bleu`[8] [53] for evaluation. When inference, we follow [63] to use beam size 4 and length penalty 0.6 for WMT14 En$\rightarrow$De, beam size 5 and penalty 1.0 for other tasks. We further report the `sacre-bleu` evaluated BLEU score on WMT14 En$\rightarrow$De and En$\rightarrow$Fr tasks to show advanced comparisons, where the corresponded results are 29.5 and 41.8, respectively.

A.2 Abstrative Summarization

For summarization, we take the pre-trained BART [35] model as backbone and fine-tune on the CNN/DailyMail dataset[9]. BART is a pre-trained sequence-to-sequence model based on the masked source input and autoregressive target output, which contains 12 layers of Transformer encoder and 12 layers of Transformer decoder, the embedding size is $1,024$, and the feed-forward size is $4,096$. Dropout value is 0.1. During fine-tuning, we follow the hyper-parameters used in [35]. The pre-trained model and the backbone implementations are all from Fairseq[10]. The training is conducted on 8 GEFORCE RTX 3090 GPU cards.

A.3 Language Modeling

For language modeling, we train on the Transformer decoder [63] and Adaptive Input Transformer [5] models. The configuration for Transformer is `transformer_lm_gpt`, which contains 12 layers with embedding size 768 and feed-forward size $3,072$, attention heads 12. Dropout and attention dropout are 0.1. For Adaptive Input Transformer, the configuration is `transformer_lm_wiki103` with 16 layers, embedding size $1,024$, feed-forward size $4,096$, attention heads 16, dropout 0.3, attention dropout 0.1, gelu dropout 0.1 and adaptive softmax dropout 0.2. We train Transformer model for $50k$ steps and Adaptive Input Transformer for $286k$ steps. The development is based on the code base Fairseq[11]. The training is on 8 Tesla V100 GPU cards.

[4]`https://iwslt.org/`

[5]`https://www.statmt.org/wmt14/translation-task.html`

[6]`https://github.com/pytorch/fairseq/tree/master/examples/translation`

[7]`https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl`

[8]`https://github.com/mjpost/sacrebleu`

[9]`https://github.com/abisee/cnn-dailymail`

[10]`https://github.com/pytorch/fairseq/tree/master/examples/bart`

[11]`https://github.com/pytorch/fairseq/tree/master/examples/language_model`

Hyper-parameter	**CoLA**	**MRPC**	**RTE**	**SST-2**	**MNLI**	**QNLI**	**QQP**	**STS-B**
Learning Rate	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5
Max Update	5336	2296	3120	20935	123873	33112	113272	3598
Max Sentence (Batch)	16	16	8	32	32	32	32	16
Dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Coefficient α	0.5	1.0	1.0	1.0	0.5	1.0	0.5	1.0

Table 8: Hyper-parameters when fine-tuning our models on GLUE benchmark.

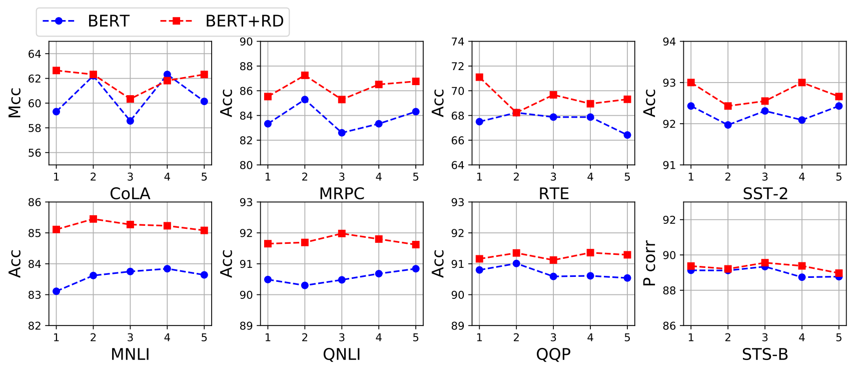

Figure 5: Results on 8 GLUE tasks with different random seeds.

A.4 Language Understanding

For language understanding tasks, we follow the popular pre-training and fine-tuning methodology, and the fine-tuned sets are the GLUE [65] benchmark. We follow previous works [7, 39] to work on the 8 tasks, including singe-sentence classification tasks (CoLA, SST-2), sentence-pair classification tasks (MNLI, QNLI, RTE, QQP, MRPC), and the sentence-pair regression task (STS-B). The detailed data statistics can be found from the original paper [65].

The pre-trained BERT-base model is the Transformer [63] encoder network, which contains 12 layers with embedding size 768, feed-forward size $3,072$ and attention heads 12. Correspondingly, the Roberta-large model contains 24 layers with embedding size $1,024$, feed-forward size $4,096$ and attention heads 16. During fine-tuning, we use Adam [29] as our optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-6}$, and L_2 weight decay of 0.01. We select the learning rate in range $\{5 \times 10^{-6}, 10^{-5}\}$ and batch size in $\{8, 16, 32\}$. Other hyper-parameter settings are mostly same as previous works [39]. The pre-trained model and the backbone implementations are all from Huggingface Transformers[12]. We report the specific settings of several important hyper-parameters in Table 8, including the dropout value. The fine-tuning experiments are conducted on 1 GEFORCE RTX 3090 GPU card.

Further, to give a clear comparison of our R-Drop based fine-tuning and vanilla fine-tuning, we plot the performance changes from different random seeds over the pre-trained BERT model on each GLUE task. The curves are shown in Figure 5. We can see that consistent improvements are achieved on different random seeds, which means our R-Drop can robustly help improve the model generalization and model performance. Besides, we also provide some task performances of different α values, shown in Table 9. This result demonstrated that α indeed is a sensitive hyper-parameter for each GLUE task, while $\alpha = 1.0$ is a good choice for most tasks.

MSE Regularization Our R-Drop is presented under the KL-divergence between two distributions. To extend our method into the regression task, such as STS-B in GLUE, we introduce the MSE-based regularization. For input data (x, y), we forward the x two times similarly as in classification and obtain the two predicted values y'_1 and y'_2. Then we regularize these two predicted values with MSE

[12]`https://github.com/huggingface/transformers`

α	**0.1**	**0.5**	**1.0**	**1.5**
MRPC	84.30	86.03	**86.51**	85.78
SST-2	92.54	92.77	**93.02**	92.43
MNLI	84.20	**84.48**	83.44	82.27
QNLI	91.21	91.92	**92.01**	91.12

Table 9: Comparison of the effect of different α for some GLUE tasks.

as follow:

$$\mathcal{L}_{mse_r} = ||y'_1 - y'_2||_2, \tag{7}$$

and we add $\mathcal{L}_{mse_r}$ with conventional MSE loss: $\mathcal{L}_{mse} = ||y - y'_1||_2 + ||y - y'_2||_2$. The final optimization objective is:

$$\mathcal{L} = \mathcal{L}_{mse} + \alpha \mathcal{L}_{mse_r}. \tag{8}$$

A.5 Image Classification

The image classification task is evaluated with the recent popular Vision Transformer (ViT) [11] model, which is the same as Transformer but with the image patch data as input. We take the two publicly released models[13], ViT-B/16 and ViT-L/16, which are pre-trained on ImageNet-21k [8] dataset with $21k$ classes and $14M$ images in total. ViT-B/16 is a Transformer model with 12 Transformer encoder layers, embedding size 768, feed-forward size $3,072$ and attention heads 12, while ViT-L/16 with 24 layers, $1,024$ embedding size, $4,096$ feed-forward size and 16 attention heads. We only conduct the fine-tuning stage experiments on CIFAR-100 and ImageNet. Note that the ImageNet results are computed without additional techniques (Polyak averaging and 512 resolution images) used to achieve results in [11]. During fine-tuning, the dropout values are 0.1 for both models. Fine-tuning is on 8 GEFORCE RTX 3090 GPU cards.

B Theoretical Discussion of R-Drop

In this section, we provide the proof for Proposition 2.1 in the main paper and make some discussions.

Proposition B.1. *For a linear model $\mathcal{P}^w(y|x) = \texttt{softmax}(\texttt{Norm}(w^T x))$ where $\texttt{Norm}(\cdot)$ denotes the normalization layer and $x \in \mathbb{R}^d$, with the constraint in Equation (6) in the main paper, we have $|\mathcal{L}_{nll}(w) - \mathbb{E}_\xi[\mathcal{L}_{nll}(w,\xi)]| \le c\sqrt{\epsilon}$, where $\mathcal{L}_{nll}(w), \mathcal{L}_{nll}(w,\xi)$ are the empirical loss calculated by the full model and a random sub model respectively, c is a constant related to the Liptschtz constant of the softmax operator.*

Proof: Here, the normalization operator normalize the row of the weight matrix w to be 1. According to the Lipschitz continuity of the loss, we have

$$|\mathcal{L}_{nll}(w) - \mathbb{E}_\xi[\mathcal{L}_{nll}(w,\xi)]| \le c_1 \cdot \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_\xi \|w^T x_i - \frac{1}{p} \cdot (w^T x_i) \odot \xi\| \tag{9}$$

$$= c_1 \cdot \frac{1}{n}\sum_{i=1}^{n} (1-p)\|w^T x_i\| \tag{10}$$

where c_1 is the Lipshitz constant.
According to the condition $\frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\xi^{(1)},\xi^{(2)}}[\mathcal{D}_{KL}(\mathcal{P}^w_{\xi^{(1)}}(y_i|x_i)||\mathcal{P}^w_{\xi^{(2)}}(y_i|x_i)))] \le \epsilon$, we have

$$\frac{1}{2n}\sum_{i=1}^{n} \mathbb{E}_{\xi^{(1)},\xi^{(2)}} \|\mathcal{P}^w_{\xi^{(1)}}(y_i|x_i) - \mathcal{P}^w_{\xi^{(2)}}(y_i|x_i))\|_1 \le \sqrt{\frac{\epsilon}{2}}, \tag{11}$$

because the relation between the KL-divergence and the total variation distance. Since we constrain the norm of w, for fixed x, the softmax operator is a bijection from $w^T x$ to $\texttt{softmax}\left(w^T x\right)$. Suppose

[13]`https://github.com/jeonsworld/ViT-pytorch`

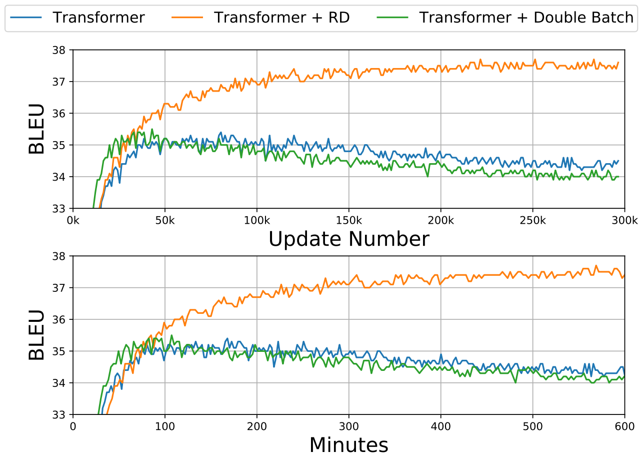

Figure 6: Results of R-Drop and Transformer with a doubled batch size.

c_2 is the Lipschitz constant of the inverse function from $\texttt{softmax}\left(w^T x\right)$ to $w^T x$, we have

$$\frac{1}{2n}\sum_{i=1}^{n}\mathbb{E}_{\xi^{(1)},\xi^{(2)}}\left\|\frac{1}{p}\cdot w^T x_i \odot \xi_1 - \frac{1}{p}\cdot w^T x_i \odot \xi_2\right\| \leq c_2\sqrt{\frac{\epsilon}{2}} \tag{12}$$

For the left term in Eq.(5), we have $\frac{1}{2np}\sum_{i=1}^{n}\mathbb{E}_{\xi^{(1)},\xi^{(2)}}\left\|w^T x_i \odot \xi_1 - w^T x_i \odot \xi_2\right\| = \frac{1-p}{n}\sum_{i=1}^{n}\|w^T x_i\|$, because $\xi^{(1)}, \xi^{(2)}$ independently follow Bernoulli distribution. Then we have $\frac{1}{n}\sum_{i=1}^{n}\|w^T x_i\| \leq \frac{c_2}{1-p}\sqrt{\frac{\epsilon}{2}}$. Combined with Eq.(4), we have

$$|\mathcal{L}_{nll}(w) - \mathbb{E}_{\xi}[\mathcal{L}_{nll}(w,\xi)]| \leq c_1 c_2\sqrt{\frac{\epsilon}{2}} \tag{13}$$

Let $c = \sqrt{\frac{1}{2}}c_1 c_2$, we can get the result.

C More Studies

C.1 Batch Size Doubled Training

As discussed in Section 2.2, we implement the algorithm by repeating input data x once and concatenating the x with repeated one in the same mini-batch to forward once. This is similar to enlarging the batch size to be double at each step. The difference is that half of the data are the same as the other half, while directly doubling the batch size, the data in the same mini-batch are all different. Therefore, we are still interested in the result of directly doubling the batch size to see the performance. We conduct experiments on IWSLT14 De→En translation with Transformer, and the batch size is enlarged from $4,096$ to be $8,192$. The result is 34.93 BLEU score. We can see that though slight improvement is achieved (compared to baseline 34.64), it falls far behind our strong performance 37.25. For the detailed training cost for each step, we present the number here: Transformer + Double Batch costs near 9ms per step, while Transformer + DR costs about 10ms per step. The additional cost is from the KL-divergence loss backward computation. We can see the cost is 1.1 times, which is a negligible cost. We also plot the valid BLEU curves along with the training for this study. The curves are shown in Figure 6. Compared to this batch size doubled training and our R-Drop, we can clearly see the advantage of R-Drop. With similar training costs, R-Drop gradually improves the performance to a much stronger one. In the figures, we also plot the curve for Transformer with the original batch size training (e.g., $4,096$) for a better comparison.

C.2 Importance of KL-Divergence

Our method introduces a KL-divergence between the two distributions from the same sample. In this study, we specifically investigate the importance of KL-divergence loss. Thus, this ablation removes

Minutes	10min	30min	60min	90min	150min	200min	300min
Baseline	27.14	33.62	34.18	34.54	-	-	-
R-Drop	16.06	31.56	33.61	34.91	35.74	36.13	36.39

Table 10: Comparison Baseline and R-Drop (reduced half-batch training) BLEU score along with training time on IWSLT14 De→En translation.

Model	Acc (CIFAR-100)	BLEU (IWSLT14 De→En)
Baseline	77.1	34.78
FD [81]	77.6	35.04
R-Drop	78.13	37.25

Table 11: Comparison of Baseline, FD and R-Drop on IWSLT14 De→En and CIFAR-100 tasks.

the $\mathcal{L}_{KL}$ loss between the two distributions and only keeps the $\mathcal{L}_{NLL}$ loss for training. Similar to other studies, we work on IWSLT14 De→En translation, and the model is Transformer. The result is also 34.93 BLEU score (same as above enlarged batch size), which is slightly better than the Transformer baseline (34.64), but far worse from our R-Drop based training result 37.25 BLEU score. This result well demonstrates the importance and effectiveness of our introduced KL-divergence loss.

C.3 Training Time and Efficiency

To fairly compare the training time between baseline and R-Drop model for the same batch size, We reduce the batch size for the R-Drop model and provide the BLEU score along with training time on IWSLT14 De→En translation tasks, which is shown in Table 10. From the table, we can see that though R-Drop needs more training to converge, the final model is much better than the baseline. We can see that the time cost is comparable for reduced half-batch training when reaching the BLEU score equal to the baseline.

C.4 Experiments for ELD and FD

To give a better understanding of the advantages of R-Drop over hidden space regularization, we also conduct experimental studies. Since FD [81] has shown its advantage over ELD [41], we mainly present experimental comparisons between FD and R-Drop. The experiments are image classification on the cifar-10 dataset (used in FD work) and IWSLT14 De→En Translation (used in R-Drop work). The cifar10 experiments are conducted on the released code[14], and we add R-Drop regularization as our implementation. For the image classification task, the R-Drop model hyper-parameter α needs to reduce linearly with the learning rate implemented by `torch.optim.lr_scheduler.MultiStepLR`. The IWSLT experiments are on our released code, and we implement the FD same as its released code. From Table 11, we can clearly see that R-Drop is superior to FD on both tasks, which can prove the advantages of the KL-divergence consistency regularization.

C.5 Ensemble and Weight Averaging

Model Number / **Model Settings**	**1**	**2**	**3**	**4**	**5**	**6**
Transformer full-model deep ensembling	34.78	36.06	36.37	36.59	36.80	36.92
Transformer full-model weight averaging (model)	34.78	0	0	0	0	0
Transformer full-model weight averaging (checkpoint)	34.78	35.00	35.21	35.26	35.29	35.44
R-Drop full-model weight averaging (checkpoint)	37.25	37.22	37.27	37.30	37.22	37.31

Table 12: Comparison of BLEU scores achieved by model deep ensembling or weight averaging with different model (trained with different random seed) and epoch checkpoint (trained with same seed).

[14]`https://github.com/akamaster/pytorch_resnet_cifar10`

We also compare our R-Drop based training with deep ensembling and weight average methods. (1) Deep ensembling usually utilizes multiple models with different parameters, which incurs additional inference costs, including both numbers of model parameters and inference time. (2) Parameter/weight averaging directly averages the multiple model parameters, and it can only be useful when the parameters are not far away from each other. In contrast, R-Drop aims to make sub models consistent within dropout-based training. It is simple yet effective by adding a consistent regularization loss without any other modifications. The model parameters are not increased, and the inference cost is the same as a single model. We train several independent models with different parameters (each with dropout) and then do deep ensembling, weight averaging on these full models. The results are show in Table 12 . Obviously, R-Drop achieves great performance with a single full model (e.g., 37.25), much better than weight averaging and deep ensembling. Ensemble methods improve the independently trained full models, but the best result is still from R-Drop (at least with 6 models ensembling here). Weight averaging can improve the performance of the base dropout model by averaging nearby checkpoints with the same random seed, but the improvement is relatively small compared with R-Drop. Besides, the weights average obtains an extremely low result for different random seeds trained models (BLEU=0) due to the far difference of model parameters.