Weight Distillation: Transferring the Knowledge in Neural Network Parameters

Ye Lin[1]*, Yanyang Li[2]*, Ziyang Wang[1], Bei Li[1], Quan Du[1], Tong Xiao[1,3], Jingbo Zhu[1,3]†
[1]NLP Lab, School of Computer Science and Engineering,
Northeastern University, Shenyang, China
[2]The Chinese University of Hong Kong, Hong Kong, China
[3]NiuTrans Research, Shenyang, China
{linye2015,blamedrlee,libeineu,duquanneu}@outlook.com
{wangziyang}@stumail.neu.edu.cn
{xiaotong,zhujingbo}@mail.neu.edu.cn

Abstract

Knowledge distillation has proven to be effective in model acceleration and compression. It transfers knowledge from a large neural network to a small one by using the large neural network predictions as targets of the small neural network. But this way ignores the knowledge inside the large neural networks, e.g., parameters. Our preliminary study as well as the recent success in pre-training suggests that transferring parameters are more effective in distilling knowledge. In this paper, we propose *Weight Distillation* to transfer the knowledge in parameters of a large neural network to a small neural network through a parameter generator. On the WMT16 En-Ro, NIST12 Zh-En, and WMT14 En-De machine translation tasks, our experiments show that weight distillation learns a small network that is 1.88∼2.94× faster than the large network but with competitive BLEU performance. When fixing the size of the small networks, weight distillation outperforms knowledge distillation by 0.51∼1.82 BLEU points.

1 Introduction

Knowledge Distillation (KD) is a popular model acceleration and compression approach (Hinton et al., 2015). It assumes that a lightweight network (i.e., *student* network, or student for short) can learn to generalize in the same way as a large network (i.e., *teacher* network, or teacher for short). To this end, a simple method is to train the student network with predicted probabilities of the teacher network as its targets.

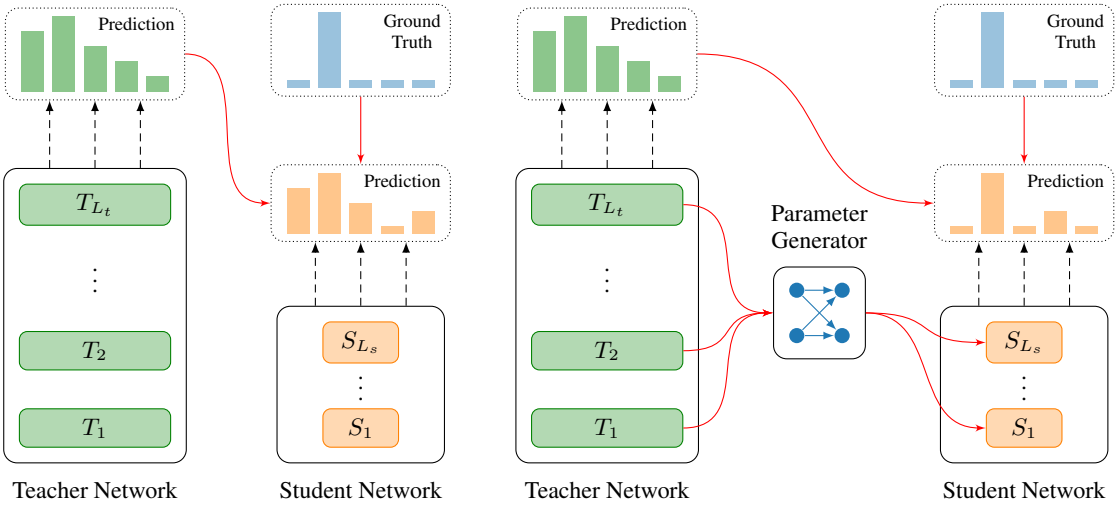
But KD has its limitation: the student network can only access the knowledge in the predictions of the teacher network. It does not consider the knowledge in the teacher network parameters. These parameters contain billions of entries for the teacher network to make predictions. Yet in KD the student only learns from those predictions with at most thousands of categories. This way results in an inferior student network, since it learns from the limited training signals. Our analysis in Section 5.1 shows that KD performs better if we simply cut off parts of parameters from the teacher to initialize the student. This fact implies that the knowledge in parameters is complementary to KD but missed. It also agrees with the recent success in pre-training (Yang et al., 2019; Liu et al., 2019; Devlin et al., 2019), where parameters reusing plays the main role. Based on this observation, a superior student is expected if all parameters in the teacher network could be exploited. However, this imposes a great challenge as the student network is too small to fit in the whole teacher network.

To fully utilize the teacher network, we propose *Weight Distillation* (WD) to transfer all the parameters of the teacher network to the student network, even if they have different numbers of weight matrices and (or) these weight matrices are of different shapes. We first use a parameter generator to predict the student network parameters from the teacher network parameters. After that, a fine-tuning process is performed to improve the quality of the transferred parameters. See Fig. 1 for a comparison of KD and WD.

We test the WD method in a well-tuned Transformer-based machine translation system. The experiments are run on three machine translation benchmarks, including the WMT16 English-Roman (En-Ro), NIST12 Chinese-English (Zh-En), and WMT14 English-German (En-De) tasks. With a similar speedup, the student network trained by WD achieves BLEU improvements of 0.51∼1.82 points over KD. With similar BLEU performance, the student network trained by WD is 1.11∼1.39× faster than KD. More interestingly, it is found that WD is very effective in improving the student net-

*Authors contributed equally.
†Corresponding author.

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 2076–2088
August 1–6, 2021.



(a) Knowledge Distillation (b) Weight Distillation

Figure 1: A comparison of Knowledge Distillation and Weight Distillation (Solid red lines denote the knowledge transfer. T_1 and S_1 are the teacher and student weight matrices at the 1st layer and so on. L_t and L_s are the numbers of layers in the teacher and student networks.).

work when its model size is close to the teacher network. On the WMT14 En-De test data, our WD-based system achieves a strong result (a BLEU score of 30.77) but is 1.88× faster than the big teacher network.

2 Background

2.1 Transformer

In this work, we choose Transformer (Vaswani et al., 2017) for study because it is one of the state-of-the-art neural models in natural language processing. Transformer is a Seq2Seq model, which consists of an encoder and a decoder. The encoder maps an input sequence to a sequence of continuous representations and the decoder maps these representations to an output sequence. Both the encoder and the decoder are composed of an embedding layer and multiple hidden layers. The decoder has an additional output layer at the end.

The hidden layer in the encoder consists of a self-attention sub-layer and a feed-forward network (FFN) sub-layer. The decoder has an additional encoder-decoder attention sub-layer between the self-attention and the FFN sub-layers. For more details, we refer the reader to (Vaswani et al., 2017).

2.2 Knowledge Distillation

KD encourages the student network to produce outputs close to the outputs of the teacher network. KD achieves this by:

$$\bar{\mathcal{S}} = \arg\min_{\mathcal{S}} \mathcal{L}(y_{\mathcal{T}}, y_{\mathcal{S}}) \quad (1)$$

where $\mathcal{L}$ is the cross-entropy loss, $y_{\mathcal{T}}$ is the teacher prediction, $\mathcal{T}$ is the teacher parameters, $y_{\mathcal{S}}$ is the student prediction and $\mathcal{S}$ is the student parameters. In practice, Eq. 1 serves as a regularization term.

A more effective KD variant for Seq2Seq models is proposed by Kim and Rush (2016). They replace the predicted distributions $y_{\mathcal{T}}$ by the generated sequences from the teacher network.

3 Weight Distillation

3.1 The Parameter Generator

The proposed parameter generator transforms the teacher parameters $\mathcal{T}$ to the student parameters $\mathcal{S}$. It is applied to the encoder and decoder separately.

The process is simple: it first groups weight matrices in the teacher network into different subsets, and then each subset is used to generate a weight matrix in the student network. Though using all teacher weights to predict student weights is possible, its efficiency becomes an issue. For instance, the number of parameters in a simple linear transformation will be the product of the numbers of entries in its input and output, where in our case these input and output contain billions of entries (from the teacher and student weights), making it intractable to keep this simple linear transformation in the memory. Grouping is an effective way

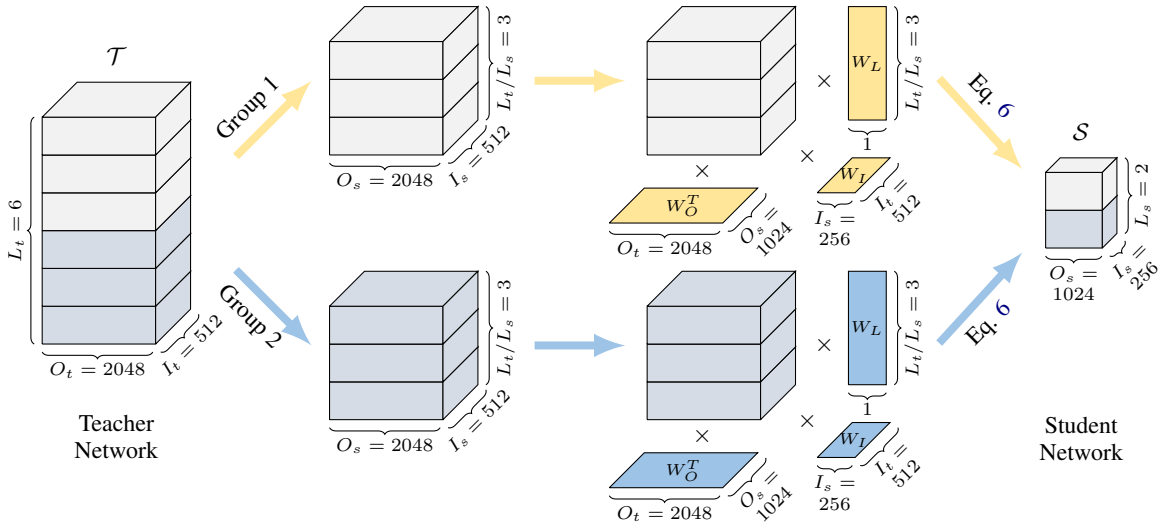


Figure 2: A running example of the Parameter Generator. We take the transformation of W_1 in Eq. 2 from the teacher to the student as an example. The teacher (stacked large cubes in the left) contains $L_t = 6$ weights (W_1) with each weight from different layers. W_1 (a single cube) in the teacher has an input dimension I_t of 512 and an output dimension O_t of 2048. The student (stacked small cubes in the right) contains only $L_s = 2$ weights (W_1) with input dimension $I_s = 256$ and output dimension $O_s = 1024$.

to reduce it to light-weighted transformation problems. Here we take the encoder as an example for the following discussion.

3.1.1 Weight Grouping

The left of Fig. 2 shows an example of weight grouping for one group with two subsets.

Before the discussion, we define the weight *class* as a weight matrix from the network formulation, and the weight *instance* as the instantiation of a weight class. Take the FFN for an example. Its formulation is defined as:

$$\mathrm{FFN}(x) = \max(xW_1 + b_1, 0)W_2 + b_2 \quad (2)$$

where W_1, b_1, W_2 and b_2 are learnable weight matrices. In this case, W_1 in Eq. 2 defines a weight class. Then all the corresponding weight matrices from FFNs in different layers of the network are the instantiations of this W_1 weight class.

From this sense, a weight class determines the role of its instantiations in design, e.g., extracting features for W_1 in Eq. 2. This means that when transferring parameters, different weight classes will contribute little to each other as they have different roles. Therefore, when predicting a student weight matrix, it is sufficient to consider the teacher weight matrices with the same weight class only, which makes the prediction efficient. So our parameter generator groups the teacher weight matrices by the weight class they belong to, i.e., different weight classes clusters all their instantiations to form their own groups. In the previous example, the W_1 weight class will form a group $[T_1, T_2, \cdots, T_{L_t}]$, where each T_i is the W_1 weight instance in the i-th FFN and L_t is the number of layers in the teacher network. These weight matrices are then used to generate the W_1 weight instances in the student network.

The parameter generator further divides each group into smaller subsets with weight matrices from adjacent layers, because the adjacent layers function similarly (Jawahar et al., 2019) and so as their weights. This way additionally makes the later transformation more light-weighted. Namely, given a group of L_t weight matrices, the parameter generator splits it into L_s subsets, where L_s is the number of layers in the student network. For example, the i-th subset of the group of W_1 weight class in the previous example will be $\left[T_{(i-1)*L_t/L_s+1}, T_{(i-1)*L_t/L_s+2}, \cdots, T_{i*L_t/L_s}\right]$. This subset is used to generate the weight matrix S_i, which corresponds to W_1 weight instance in the i-th FFN of the student network.

3.1.2 Weight Transformation

Given a subset of teacher weight matrices, the parameter generator then transforms them to the desired student weight matrix, as shown in the right of Fig. 2.

Let us see the process of generating the

weight matrix $S \in \mathbb{R}^{I_s \times O_s}$ from the subset $\left[T_1, T_2, \cdots, T_{L_t/L_s}\right]$ with each $T_i \in \mathbb{R}^{I_t \times O_t}$, where I_s and O_s are the input and output dimensions of the student weight matrix, I_t and O_t are the input and output dimensions of the teacher weight matrix. The parameter generator first stacks all weight matrices in this subset into a tensor $\hat{T} \in \mathbb{R}^{I_t \times O_t \times L_t/L_s}$. Then it uses three learnable weight matrices, $W_I \in \mathbb{R}^{I_t \times I_s}, W_O \in \mathbb{R}^{O_t \times O_s}, W_L \in \mathbb{R}^{L_t/L_s \times 1}$, to transform $\hat{T}$ to the shape $I_s \times O_s \times 1$ sequentially:

$$\hat{T}_{\cdot jk} \leftarrow \hat{T}_{\cdot jk} W_I, \forall j \in [1, O_t], k \in [1, L'] \quad (3)$$

$$\hat{T}_{j \cdot k} \leftarrow \hat{T}_{j \cdot k} W_O, \forall j \in [1, I_s], k \in [1, L'] \quad (4)$$

$$\hat{T}_{jk \cdot} \leftarrow \hat{T}_{jk \cdot} W_L, \forall j \in [1, I_s], k \in [1, O_s] \quad (5)$$

where $L' = L_t/L_s$.

Finally we transform $\hat{T}$ (with 1 in its shape get eliminated) to produce S, as follows:

$$S = \tanh(\hat{T}) \odot W + B \quad (6)$$

where W and B are learnable weight matrices of the parameter generator and have the same shape as $\hat{T}$. $\odot$ denotes the Hadamard product. The tanh function provides non-linearity. W and B are used to scale and shift the tanh output to any desirable value. Note that we do not share W_I, W_O, W_L, W and B when generating different S. If the encoder is of the same size in both the teacher and student networks, only Eq. 6 is needed to map each weight matrix from the teacher network to the student network.

3.2 Training

There are two training phases in WD: In the first phase (Phase 1), we train the parameter generator $\pi = \{W_I, W_O, W_L, W, B\}$ to predict the student network $\mathcal{S}$; In the second phase (Phase 2), we fine-tune the generated student network $\mathcal{S}$ to obtain better results. Phase 2 is necessary because the parameter generator is simply a feed-forward network with one hidden layer and thus has no enough capacity to produce a good enough student network at once. A more sophisticated parameter generator is an alternative, but it is expensive due to its large input and output spaces.

The task of Phase 1 is to minimize the loss of the student network with parameters $\mathcal{S}$ predicted by the parameter generator π from the teacher parameters $\mathcal{T}$. The objective of Phase 1 is:

$$\bar{\pi} = \arg\min_{\pi}[(1-\alpha)\mathcal{L}(y_{\mathcal{T}}, y_\pi) + \alpha\mathcal{L}(y, y_\pi)] \quad (7)$$

where $\mathcal{L}$ is the cross-entropy loss, $y_{\mathcal{T}}$ is the teacher prediction, y_π is the prediction of the student network generated by the parameter generator π, y is the ground truth, and α is a hyper-parameter that balances two losses and is set to 0.5 by default. The first term of Eq. 7 is the KD loss as in Eq. 1, and the second term is the standard loss.

The objective of Phase 2 has the same form as Eq. 7, except that it optimizes $\mathcal{S}$ instead of π, like this:

$$\bar{\mathcal{S}} = \arg\min_{\mathcal{S}}[(1-\alpha)\mathcal{L}(y_{\mathcal{T}}, y_{\mathcal{S}}) + \alpha\mathcal{L}(y, y_{\mathcal{S}})] \quad (8)$$

4 Experiments

4.1 Datasets

We evaluate our methods on the WMT16 English-Roman (En-Ro), NIST12 Chinese-English (Zh-En), and WMT14 English-German (En-De) tasks.

For the En-Ro task, we use the WMT16 English-Roman dataset (610K pairs). We choose *newsdev-2016* as the validation set and *newstest-2016* as the test set. For the Zh-En task, we use 1.8M sentence Chinese-English bitext provided within NIST12 OpenMT[1]. We choose the evaluation data of *mt06* as the validation set, and *mt08* as the test set. For the En-De task, we use the WMT14 English-German dataset (4.5M pairs). We share the source and target vocabularies. We choose *newstest-2013* as the validation set and *newstest-2014* as the test set.

For all datasets, we tokenize every sentence using the script in the Moses toolkit and segment every word into subword units using Byte-Pair Encoding (Sennrich et al., 2016). The number of the BPE merge operations is set to 32K. We remove sentences with more than 250 subword units (Xiao et al., 2012). In addition, we evaluate the results using `multi-bleu.perl`.

[1]LDC2000T46, LDC2000T47, LDC2000T50, LDC2003E14, LDC2005T10, LDC2002E18, LDC2007T09, LDC2004T08

	System	Depth	Width	Test	Δ_{BLEU}	Valid	Params	Speed	Speedup
WMT16 En-Ro	Teacher	6	512	31.64	-	32.07	83M	138.35 sent./s	1.00×
	TINY	1	256	29.65	-	29.73	45M	323.26 sent./s	2.34×
	+ KD	1	256	30.03	0.00	29.98	45M	347.07 sent./s	2.51×
	+ WD	1	256	30.89	+0.86	30.89	45M	359.53 sent./s	2.60×
	SMALL	2	512	31.22	-	31.19	66M	281.31 sent./s	2.03×
	+ KD	2	512	30.97	0.00	30.77	66M	289.11 sent./s	2.09×
	+ WD	2	512	31.65	+0.68	31.27	66M	289.80 sent./s	2.09×
NIST12 Zh-En	Teacher	6	512	45.14	-	51.91	102M	88.42 sent./s	1.00×
	TINY	1	256	41.90	-	48.28	60M	225.46 sent./s	2.55×
	+ KD	1	256	42.78	0.00	49.71	60M	214.06 sent./s	2.42×
	+ WD	1	256	44.60	+1.82	51.56	60M	247.90 sent./s	2.80×
	SMALL	2	512	44.30	-	50.83	85M	194.23 sent./s	2.20×
	+ KD	2	512	44.89	0.00	51.87	85M	199.74 sent./s	2.26×
	+ WD	2	512	46.20	+1.31	53.04	85M	199.29 sent./s	2.25×
WMT14 En-De	Teacher	6	512	27.47	-	26.79	96M	158.29 sent./s	1.00×
	TINY	1	256	24.62	-	24.88	55M	321.79 sent./s	2.03×
	+ KD	1	256	26.51	0.00	26.01	55M	412.91 sent./s	2.61×
	+ WD	1	256	27.12	+0.61	26.42	55M	406.68 sent./s	2.57×
	SMALL	2	512	26.68	-	26.07	80M	281.97 sent./s	1.78×
	+ KD	2	512	27.47	0.00	26.54	80M	306.91 sent./s	1.94×
	+ WD	2	512	28.18	+0.71	26.97	80M	309.11 sent./s	1.95×

Table 1: Results of Transformer-base on different tasks (sent./s: translated sentences per second).

4.2 Model Setup

Our baseline system is based on the open-source implementation of the Transformer model presented in Ott et al. (2019)'s work. For all machine translation tasks, we experiment with the Transformer-base (base) setting. We additionally run the Transformer-big (big) (Vaswani et al., 2017) and Transformer-deep (deep) (Wang et al., 2019; Zhang et al., 2020) settings on the large En-De dataset. All systems consist of a 6-layer encoder and a 6-layer decoder, except that the Transformer-deep encoder has 48 layers (depth) (Li et al., 2020). The embedding size (width) is set to 512 for Transformer-base/deep and 1,024 for Transformer-big. The FFN hidden size equals to 4× embedding size in all settings. We stop training until the model stops improving on the validation set. All experiments are done on 8 NVIDIA TITIAN V GPUs with mixed-precision training (Micikevicius et al., 2018). At test time, the model is decoded with a beam of width 4/6/4, a length normalization weight of 1.0/1.0/0.6 and a batch size of 64 for the En-Ro/Zh-En/En-De tasks with half-precision.

Note that our method can also be seen as an advanced version of Tucker Decomposition (Tucker, 1966). So we also implement a baseline based on Tucker Decomposition. Unfortunately, this model does not converge to a good optima and performs extremely poor.

For the KD baseline, we adopt Kim and Rush (2016)'s method, which has proven to be the most effective for Seq2Seq models (Kim et al., 2019). It generates the pseudo data from the source side of the bilingual corpus. The choices of student networks are based on the observation that the encoder has a greater impact on performance and the decoder dominates the decoding time (Kasai et al., 2020). Therefore we vary the depth and width of the decoder. We test two student network configurations: TINY halves the decoder width and uses a 1-layer decoder (the fastest WD student network with the performance close to the teacher network); SMALL uses a 2-layer decoder whose width is the same as the teacher network (the fastest KD student network with the performance close to the teacher network).

All hyper-parameters of WD are identical to the baseline system, except that WD uses 1/4 warmup steps in Phase 2. For the parameter generator initialization, we use Glorot and Bengio (2010)'s method to initialize W_I, W_O, W_L in Eqs. 3 - 5. W and B in Eq. 6 are initialized to constants 1 and 0 respec-

	System	Depth	Width	Test	Δ_{BLEU}	Valid	Params	Speed	Speedup
big	Teacher	6	1024	29.11	-	27.66	281M	123.92 sent./s	1.00×
	TINY	1	512	25.83	-	25.33	150M	353.42 sent./s	2.85×
	+ KD	1	512	27.70	0.00	26.52	150M	353.82 sent./s	2.86×
	+ WD	1	512	28.60	+0.90	26.83	150M	364.67 sent./s	2.94×
	SMALL	2	1024	27.62	-	26.78	214M	252.46 sent./s	2.04×
	+ KD	2	1024	29.01	0.00	27.54	214M	261.78 sent./s	2.11×
	+ WD	2	1024	29.52	+0.51	27.97	214M	260.34 sent./s	2.10×
deep	Teacher	6	512	29.43	-	27.82	229M	134.26 sent./s	1.00×
	TINY	1	256	26.34	-	26.05	187M	270.30 sent./s	2.01×
	+ KD	1	256	29.36	0.00	27.39	187M	308.57 sent./s	2.30×
	+ WD	1	256	29.92	+0.56	27.99	187M	285.43 sent./s	2.13×
	SMALL	2	512	28.06	-	26.51	212M	245.82 sent./s	1.83×
	+ KD	2	512	29.83	0.00	28.02	212M	258.45 sent./s	1.92×
	+ WD	2	512	30.77	+0.94	28.33	212M	252.69 sent./s	1.88×

Table 2: Results of Transformer-big/deep on WMT14 En-De (sent./s: translated sentences per second).

System	Test	Δ_{BLEU}	Valid	Δ_{BLEU}
TINY (KD)	42.78	0.00	49.71	0.00
+ Init	43.36	+0.58	50.32	+0.61
+ WD	44.60	+1.82	51.56	+1.85
SMALL (KD)	44.89	0.00	51.87	0.00
+ Init	45.66	+0.77	52.57	+0.70
+ WD	46.20	+1.31	53.04	+1.17

Table 3: Initialization study (Init: initialize the student network with the teacher parameters).

tively. All results are the average of three identical runs with different random seeds.

4.3 Results

Table 1 shows the results of different approaches on different student networks with Transformer-base as the teacher network. In all three tasks and different sized student networks, WD outperforms KD by 0.77, 1.57, and 0.66 BLEU points on En-Ro, Zh-En, and En-De on average. Our method (TINY) can obtain similar performance to the teacher network with only half of its parameters and is 2.57∼2.80× faster, while KD (SMALL) uses more parameters and has only a 1.94∼2.26× speedup in the same case. We attribute the success of WD to that the parameter generator uses parameters of the teacher network to provide a good initialization for the student network, as Phase 1 behaves like the initialization, and the effectiveness of a good initialization has been widely proven (Erhan et al., 2010; Mishkin and Matas, 2016). Interestingly, both KD and WD surpass the teacher network when the student network size is close to the teacher network (SMALL). This is due to that KD has a form similar to data augmentation (Gordon and Duh, 2019).

Table 2 shows the results of larger networks, i.e., Transformer-big/deep. The phenomenon here is similar to that in Table 1. The acceleration on Transformer-big is more obvious than on Transformer-base (2.94× vs. 2.57× for TINY and 2.10× vs. 1.95× for SMALL in WD). This is because the decoder in Transformer-big occupies a larger portion of the decoding time than in Transformer-base. But the acceleration on Transformer-deep is less obvious than on Transformer-base (2.13× vs. 2.57× for TINY and 1.88× vs. 1.95× for SMALL in WD), as a deeper encoder consumes more inference time. Moreover, compared with such a strong Transformer-deep teacher, WD (SMALL) can still outperform it by 1.34 BLEU points with a 1.88× speedup, achieving the state-of-the-art.

5 Analysis

To better understand WD, we conduct a series of experiments on the NIST12 Zh-En validation set with the Transformer-base teacher.

5.1 Initialization Study

To test whether KD misses knowledge in parameters, we initialize the student network with the teacher parameters. If the teacher and student networks have different depths, we initialize the student network with the bottom layers of the teacher network (Sanh et al., 2019). If they have different

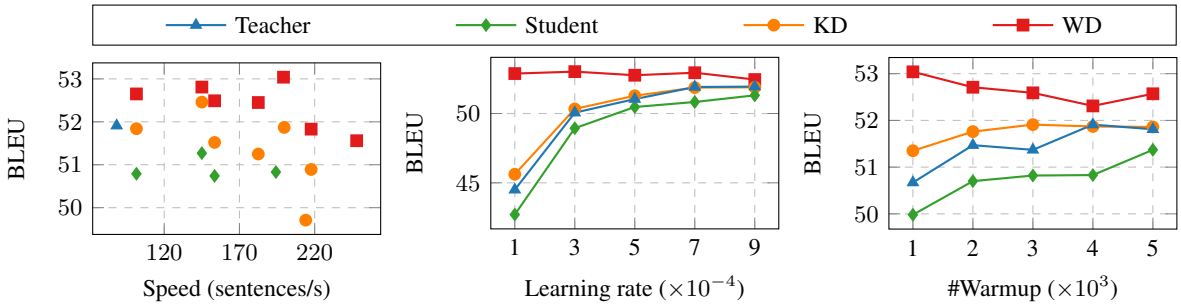


Figure 3: Sensitivity analysis on SMALL.

System	Test	Δ_{BLEU}	Valid	Δ_{BLEU}
SMALL	44.30	0.00	50.83	0.00
+ KD	44.89	+0.59	51.87	+1.04
+ Encoder	45.40	+1.10	51.62	+0.79
+ Decoder	45.26	+0.96	51.34	+0.51
+ Embed (Enc)	44.67	+0.37	51.22	+0.39
+ Embed (Dec)	45.06	+0.76	51.26	+0.43
+ Output	45.10	+0.80	51.28	+0.45

Table 4: Ablation study of using different weight matrices solely.

W / D	256		512	
	$\text{BLEU}_{\text{KD/WD}}$	Params	$\text{BLEU}_{\text{KD/WD}}$	Params
1	38.46/40.34	30M	43.51/45.39	65M
2	45.33/47.21	32M	50.02/50.45	72M
3	47.30/49.09	34M	51.18/51.99	80M
4	47.90/50.08	36M	51.05/52.05	87M
5	48.87/50.70	38M	52.15/52.00	94M
6	49.78/50.73	40M	52.40/53.09	102M

Table 5: Compression study with various depth (D) and width (W) of both the encoder and decoder.

widths, we slice the teacher weight matrices to fit the student network (Wang et al., 2020). Table 3 shows that initializing the student networks with the teacher parameters improves KD, supporting our claim that knowledge in parameters is complementary to KD but missed. We also see that WD outperforms this simple initialization, which implies that using all teacher parameters helps to obtain a better student.

5.2 Sensitivity Analysis

The left part of Fig. 3 studies how sensitive the performance (BLEU) of different methods are to various levels of inference speedup (obtained by varying decoder depth and width). It shows that WD distributes on the upper right of the figure, which means that WD produces student networks that are consistently faster and better.

We also investigate how sensitive different methods are to the training hyper-parameters, i.e., the learning rate and warmup steps. Here we focus on Phase 2 of WD, as it directly impacts the final performance. The middle part of Fig. 3 shows that WD can endure learning rates in a wide range, because its performance does not vary much. However, a very large learning rate still negatively impacts the performance. The right part of Fig. 3 is the opposite, where WD is more sensitive to the warmup steps than the learning rate. This is because more warmup steps will run the network with a high learning rate in a longer period. A high learning rate has been proven to be harmful as shown in the middle part of Fig. 3.

5.3 Ablation Study

Table 4 studies which weight matrices in the teacher network are the most effective. It is achieved by training the parameter generator with only the intended weight matrices and without the KD loss term in Eq. 7. We see that using any weight matrix brings a significant improvement over the baseline. This observation shows that weight matrices in the teacher network do contain abundant knowledge. Among these, the encoder weight matrices produce the most significant result, which agrees with the previous study claiming that the encoder is more important than the decoder (Wang et al., 2019; Bapna et al., 2018).

5.4 Compression Study

As the previous experiments focus on a lightweight decoder for acceleration, the compression is limited as the encoder remains large. To examine the effectiveness of WD on model compression, we shrink the depth and width of the encoder and decoder simultaneously. As shown in Table 5, WD consistently outperforms KD by about 1 BLEU

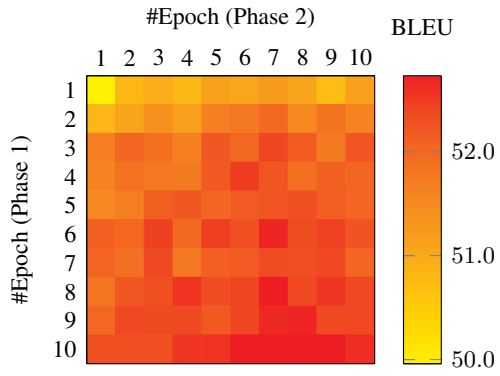


Figure 4: Training efficiency of WD on SMALL.

point under various compression ratios (ranging from 1.00× to 3.40×). Note that decreasing the width brings more significant compression. This is because a large portion of the parameters is from the embedding matrices and the output projection. The sizes of these matrices are determined by the width and a fixed vocabulary size.

5.5 Training Efficiency

Fig. 4 studies the training efficiency of WD by comparing the final BLEU scores when two training phases end in different epochs. As shown in Fig. 4, Phase 1 has little impact on Phase 2, because Phase 2 converges to optimums with similar BLEU scores once Phase 1 runs for a few epochs (say, 3 epochs). If we run Phase 1 longer, then Phase 2 converges faster. This phenomenon suggests that Phase 1 already transfers the knowledge in the teacher parameters within the first few epochs, and the remaining epochs merely do the fine-tuning (Phase 2) job. This implies that the training of WD is efficient, since we can just train the parameter generator for several epochs first, then fine-tune the generated network as in KD, and finally obtain a much better result than KD.

Though we could train the parameter generator for just a few epochs as suggested, Phase 1 is still time-consuming. The reasons are two folds: 1) the parameter generator consumes a lot of memory and we have to resort to gradient accumulation; 2) the parameter generator involves many large matrix multiplications. For the experiments in Table 1 and Table 2, it takes us 0.66 days for WD to finish training on average, whereas 0.55 days for the teacher network baseline and 0.31 days for both the student network baseline and KD.

6 Related Work

6.1 Knowledge Distillation

Knowledge distillation (Hinton et al., 2015; Freitag et al., 2017) is a widely used model acceleration and compression technique (Jiao et al., 2019; Sanh et al., 2019; Liu et al., 2020). It treats the network predictions as the knowledge learned by the teacher network, since these predicted distributions contain the ranking information on similarities among categories. It then transfers this knowledge to the student network by enforcing the student network to have similar predictions. The followed work extends this idea by providing more knowledge from different sources to the student network. FitNets (Romero et al., 2015) uses not only the predictions but also the intermediate representations learned by the teacher network to supervise the student network. For the Seq2Seq model, Kim and Rush (2016) proposes to use the generated sequences as the sequence-level knowledge to guide the student network training. Moreover, self-knowledge distillation (Hahn and Choi, 2019) even shows that knowledge (representations) from the student network itself can improve the performance.

Our weight distillation, on the other hand, explores a new source of knowledge and a new way to leverage this knowledge. It transfers the knowledge in parameters of the teacher network to the student network via a parameter generator. Therefore, it is orthogonal to other knowledge distillation variants.

6.2 Transfer Learning

Transfer learning aims at transferring knowledge from a source domain to a target domain. Based on what knowledge is transferred to the model in the target domain, transfer learning methods can be classified into three categories (Pan and Yang, 2010): instance-based methods reuse certain parts of the data in the source domain (Jiang and Zhai, 2007; Dai et al., 2007); feature-based methods use the representation from the model learned in the source domain as the input (Peters et al., 2018; Gao et al., 2008); parameter-based methods directly fine-tune the model learned in the source domain with the target domain data (Yang et al., 2019; Liu et al., 2019; Devlin et al., 2019).

Perhaps the most related work is Platanios et al. (2018)'s work. Their method falls into the parameter-based category. They use a universal parameter generator to share the knowledge among translation tasks. This parameter generator pro-

duces a translation model from a given language-specific embedding. Though we similarly employ the idea of a parameter generator, our weight distillation aims at transferring knowledge from one model to another rather than from one translation task to another. Therefore our parameter generator takes a model instead of a language-specific embedding as its input and is only used once.

7 Conclusion

In this work, we propose weight distillation to transfer knowledge in the parameters of the teacher network to the student network. It generates the student network from the teacher network via a parameter generator. Our experiments on three machine translation tasks show that weight distillation consistently outperforms knowledge distillation by producing a faster and better student network.

Acknowledgments

This work was supported in part by the National Science Foundation of China (Nos. 61876035 and 61732005), the National Key R&D Program of China (No. 2019QY1801), and the Ministry of Science and Technology of the PRC (Nos. 2019YFF0303002 and 2020AAA0107900). The authors would like to thank anonymous reviewers for their comments.

References

Ankur Bapna, Mia Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. 2018. Training deeper neural machine translation models with transparent attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3028–3033, Brussels, Belgium. Association for Computational Linguistics.

Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. 2007. Boosting for transfer learning. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 193–200. ACM.

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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Dumitru Erhan, Aaron C. Courville, Yoshua Bengio, and Pascal Vincent. 2010. Why does unsupervised pre-training help deep learning? In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 201–208. JMLR.org.

Markus Freitag, Yaser Al-Onaizan, and Baskaran Sankaran. 2017. Ensemble distillation for neural machine translation. *CoRR*, abs/1702.01802.

Jing Gao, Wei Fan, Jing Jiang, and Jiawei Han. 2008. Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 283–291. ACM.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org.

Mitchell A. Gordon and Kevin Duh. 2019. Explaining sequence-level knowledge distillation as data-augmentation for neural machine translation. *CoRR*, abs/1912.03334.

Sangchul Hahn and Heeyoul Choi. 2019. Self-knowledge distillation in natural language processing. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing, RANLP 2019, Varna, Bulgaria, September 2-4, 2019*, pages 423–430. INCOMA Ltd.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3651–3657. Association for Computational Linguistics.

Jing Jiang and ChengXiang Zhai. 2007. Instance weighting for domain adaptation in NLP. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*. The Association for Computational Linguistics.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling BERT for natural language understanding. *CoRR*, abs/1909.10351.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *CoRR*, abs/2006.10369.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1317–1327. The Association for Computational Linguistics.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation@EMNLP-IJCNLP 2019, Hong Kong, November 4, 2019*, pages 280–288. Association for Computational Linguistics.

Bei Li, Ziyang Wang, Hui Liu, Quan Du, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2020. Learning light-weight translation models from deep transformer. *CoRR*, abs/2012.13866.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling BERT with adaptive inference time. *CoRR*, abs/2004.02178.

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. *CoRR*, abs/1907.11692.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed precision training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Dmytro Mishkin and Jiri Matas. 2016. All you need is a good init. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*, pages 48–53. Association for Computational Linguistics.

Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.*, 22(10):1345–1359.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.

Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom M. Mitchell. 2018. Contextual parameter generation for universal neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 425–435. Association for Computational Linguistics.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. Fitnets: Hints for thin deep nets. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Ledyard R Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311.

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

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. Hat: Hardware-aware transformers for efficient natural language processing.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2,*

2019, Volume 1: Long Papers, pages 1810–1822. Association for Computational Linguistics.

Tong Xiao, Jingbo Zhu, Hao Zhang, and Qiang Li. 2012. Niutrans: An open source toolkit for phrase-based and syntax-based machine translation. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the System Demonstrations, July 10, 2012, Jeju Island, Korea*, pages 19–24. The Association for Computer Linguistics.

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

Yuhao Zhang, Ziyang Wang, Runzhe Cao, Binghao Wei, Weiqiao Shan, Shuhan Zhou, Abudurexiti Reheman, Tao Zhou, Xin Zeng, Laohu Wang, Yongyu Mu, Jingnan Zhang, Xiaoqian Liu, Xuanjun Zhou, Yinqiao Li, Bei Li, Tong Xiao, and Jingbo Zhu. 2020. The niutrans machine translation systems for WMT20. In *Proceedings of the Fifth Conference on Machine Translation, WMT@EMNLP 2020, Online, November 19-20, 2020*, pages 338–345. Association for Computational Linguistics.

A Appendices

Hyper-parameters. We tune the learning rate and warmup steps in Phase 2 of WD. We use the grid search to select the learning rate in $[1 \times 10^{-4}, 3 \times 10^{-4}, 5 \times 10^{-4}, 7 \times 10^{-4}, 9 \times 10^{-4}]$ and warmup steps in $[1000, 2000, 3000, 4000, 5000]$ that have the best average BLEU performance in all validation sets.

Datasets. Detailed data statistics as well as the URLs of three machine translation tasks we used, including WMT16 English-Roman (En-Ro), NIST12 Chinese-English (Zh-En), and WMT14 English-German (En-De), are shown in Table 6.

For En-Ro, the training set consists of 0.6M bilingual sentence pairs. The validation set *newsdev-2016* contains 1999 pairs and the test set *newstest-2016* contains 1999 pairs. For Zh-En, the training set consists of 1.8M bilingual sentence pairs. The validation set *mt06* contains 1,664 pairs and the test set *mt08* contains 1,357 pairs. For En-De, the training set consists of 4.5M bilingual sentence pairs. The validation set *newstest-2013* contains 3,000 pairs and the test set *newstest-2014* contains 3,003 pairs.

Runtime. To compare the average runtime for each approach, Table 7 shows the actual number of updates and runtime. For the baseline models (i.e., Teacher, TINY and SMALL) and KD, we record their runtime in the Phase 1 entry because they only need to be trained once.

Lang.	Train		Test		Valid	
	Sent.	Word	Sent.	Word	Sent.	Word
En-Ro	0.6M	33M	1999	112K	1999	118K
Zh-En	1.8M	115M	1357	247K	1664	280K
En-De	4.5M	262M	3003	164K	3000	156K

Table 6: Date statistics.

One can observe that in Table 7, Phase 2 of WD generally consumes similar or less time as well as the number of updates than other approaches. This is because the model is already close to the optimum before the fine-tuning (Phase 2). Table 7 also shows that the number of updates in Phase 1 of WD is much less than other approaches, yet its training time is much longer. This phenomenon is more obvious in Transformer-deep models. This is because one step in Phase 1 of WD is 2.11× slower than in Phase 2 of WD.

Decoder. We also investigate how WD's performance (on the validation set) and speed change given different decoder depths and widths. We choose the speed of WD to compute the speedup of different decoder depths and widths. Although the actual speedup of KD will not be exactly the same as the one of WD due to their different decoding results, they are close.

As shown in Table 8, WD is robust to different sized decoders, with both BLEU and speed significantly outperform KD. WD consistently outperforms KD by about 1 BLEU point under various decoder depths and widths. Interestingly, we find that pruning the layers degrades the performance more than shrinking its width, but it provides a higher speedup. Taking the student network with depth 2 and width 512 as an example, if we shrink the depth from 2 to 1, there is a decrease of 1.21 BLEU points in WD but with 1.12× speedup. When we shrink the width from 512 to 256, it leads to a moderate decrease of 0.59 BLEU points yet with only 1.06× speedup. This might be because layers are computed sequentially and wider matrices enjoy the parallel computation acceleration provided by modern GPUs.

Loss. In Table 7, we observe that WD generates student networks that are superior to KD. We believe that this is because WD converges to a better

	System	Depth	Width	Test	Valid	Phase 1		Phase 2	
						#Update	Time	#Update	Time
WMT16 En-Ro	Teacher (base)	6	512	31.64	32.07	70K	0.06	-	-
	TINY	1	256	29.65	29.73	70K	0.03	-	-
	+ KD	1	256	30.03	29.98	70K	0.03	-	-
	+ WD	1	256	30.89	30.89	47K	0.04	70K	0.03
	SMALL	2	512	31.22	31.19	70K	0.04	-	-
	+ KD	2	512	30.97	30.77	70K	0.04	-	-
	+ WD	2	512	31.65	31.27	47K	0.06	70K	0.04
NIST12 Zh-En	Teacher (base)	6	512	45.14	51.91	30K	0.08	-	-
	TINY	1	256	41.90	48.28	30K	0.05	-	-
	+ KD	1	256	42.78	49.71	30K	0.05	-	-
	+ WD	1	256	44.60	51.56	20K	0.07	30K	0.05
	SMALL	2	512	44.30	50.83	30K	0.06	-	-
	+ KD	2	512	44.89	51.87	30K	0.06	-	-
	+ WD	2	512	46.20	53.04	20K	0.09	30K	0.06
WMT14 En-De	Teacher (base)	6	512	27.47	26.79	100K	0.24	-	-
	TINY	1	256	24.62	24.88	100K	0.14	-	-
	+ KD	1	256	26.51	26.01	100K	0.14	-	-
	+ WD	1	256	27.12	26.42	50K	0.18	80K	0.11
	SMALL	2	512	26.68	26.07	100K	0.19	-	-
	+ KD	2	512	27.47	26.54	100K	0.19	-	-
	+ WD	2	512	28.18	26.97	50K	0.25	80K	0.15
	Teacher (big)	6	1024	29.11	27.66	200K	1.71	-	-
	TINY	1	512	25.83	25.33	200K	0.58	-	-
	+ KD	1	512	27.70	26.52	200K	0.58	-	-
	+ WD	1	512	28.60	26.83	67K	0.57	100K	0.29
	SMALL	2	1024	27.62	26.78	200K	0.79	-	-
	+ KD	2	1024	29.01	27.54	200K	0.79	-	-
	+ WD	2	1024	29.52	27.97	67K	0.55	100K	0.40
	Teacher (deep)	6	512	29.43	27.82	60K	0.67	-	-
	TINY	1	256	26.34	26.05	60K	0.57	-	-
	+ KD	1	256	29.36	27.39	60K	0.57	-	-
	+ WD	1	256	29.92	27.99	30K	1.51	30K	0.29
	SMALL	2	512	28.06	26.51	60K	0.60	-	-
	+ KD	2	512	29.83	28.02	60K	0.60	-	-
	+ WD	2	512	30.77	28.33	30K	1.53	30K	0.30

Table 7: Results of Transformer on different tasks (Time is measured by GPU days).

W / D	256				512			
	KD	WD	Δ_{BLEU}	Speedup	KD	WD	Δ_{BLEU}	Speedup
1	49.71	51.56	+1.85	2.80×	50.89	51.83	+0.94	2.53×
2	51.25	52.45	+1.20	2.12×	51.87	53.04	+1.17	2.25×
3	51.52	52.49	+0.97	1.78×	52.46	52.81	+0.35	1.68×
4	51.41	52.42	+1.01	1.62×	52.07	53.66	+1.59	1.56×
5	51.27	52.71	+1.44	1.33×	52.07	52.74	+0.67	1.30×
6	50.79	52.65	+1.86	1.18×	51.91	53.09	+1.18	1.02×

Table 8: BLEU and speed vs. decoder depth and width (Transformer-base, NIST12 Zh-En).

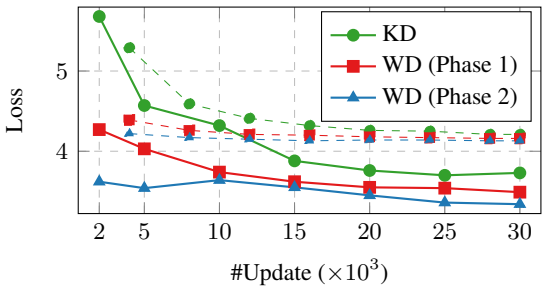


Figure 5: Train (solid)/valid (dash) loss of SMALL.

optimum. To examine this hypothesis, we study its loss in Fig. 5. As can be seen, WD does obtain much lower train and valid losses than KD. We also see that Phase 1 already outperforms KD at the end. Given the fact that Phase 1 does the initialization job for Phase 2 and Phase 2 is KD exactly, the way WD works can be treated as providing a good start.