TextBrewer: An Open-Source Knowledge Distillation Toolkit for Natural Language Processing

Ziqing Yang†, Yiming Cui‡†, Zhipeng Chen†,
Wanxiang Che‡, Ting Liu‡, Shijin Wang†§, Guoping Hu†
†State Key Laboratory of Cognitive Intelligence, iFLYTEK Research, China
‡Research Center for Social Computing and Information Retrieval (SCIR),
Harbin Institute of Technology, Harbin, China
§iFLYTEK AI Research (Hebei), Langfang, China
†§{zqyang5,ymcui,zpchen,sjwang3,gphu}@iflytek.com
‡{ymcui,car,tliu}@ir.hit.edu.cn

Abstract

In this paper, we introduce **TextBrewer**, an open-source knowledge distillation toolkit designed for natural language processing. It works with different neural network models and supports various kinds of supervised learning tasks, such as text classification, reading comprehension, sequence labeling. TextBrewer provides a simple and uniform workflow that enables quick setting up of distillation experiments with highly flexible configurations. It offers a set of predefined distillation methods and can be extended with custom code. As a case study, we use TextBrewer to distill BERT on several typical NLP tasks. With simple configurations, we achieve results that are comparable with or even higher than the public distilled BERT models with similar numbers of parameters. [1]

1 Introduction

Large pre-trained language models, such as GPT (Radford, 2018), BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019b) and XLNet (Yang et al., 2019) have achieved great success in many NLP tasks and greatly contributed to the progress of NLP research. However, one big issue of these models is the high demand for computing resources — they usually have hundreds of millions of parameters, and take several gigabytes of memory to train and inference — which makes it impractical to deploy them on mobile devices or online systems. From a research point of view, we are tempted to ask: is it necessary to have such a big model that contains hundreds of millions of parameters to achieve a high performance? Motivated by the above considerations, recently, some researchers in the NLP community have tried to design lite models (Lan et al., 2019), or resort to knowledge distillation (KD) technique to compress large pre-trained models to small models.

KD is a technique of transferring knowledge from a teacher model to a student model, which is usually smaller than the teacher. The student model is trained to mimic the outputs of the teacher model. Before the birth of BERT, KD had been applied to several specific tasks like machine translation (Kim and Rush, 2016; Tan et al., 2019) in NLP. While the recent studies of distilling large pre-trained models focus on finding general distillation methods that work on various tasks and are receiving more and more attention (Sanh et al., 2019; Jiao et al., 2019; Sun et al., 2019a; Tang et al., 2019; Liu et al., 2019a; Clark et al., 2019; Zhao et al., 2019).

Though various distillation methods have been proposed, they usually share a common workflow: firstly, train a teacher model, then optimize the student model by minimizing some losses that are calculated between the outputs of the teacher and the student. Therefore it is desirable to have a reusable distillation workflow framework and treat different distillation strategies and tricks as plugins so that they could be easily and arbitrarily added to the framework. In this way, we could also achieve great flexibility in experimenting with different combinations of distillation strategies and comparing their effects.

In this paper, we introduce **TextBrewer**, a PyTorch-based distillation toolkit for NLP that aims to provide a unified distillation workflow, save the effort of setting up experiments and help users to distill more effective models. TextBrewer provides simple-to-use APIs, a collection of distillation methods, and highly customizable configurations. It has also been proved able to distill BERT models efficiently and reproduce the state-of-the-art results on typical NLP tasks. The main features of TextBrewer are:

[1] TextBrewer: http://textbrewer.hfl-rc.com

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 9–16

- **Versatility in tasks and models**. It works with a wide range of models, from the RNN-based model to the transformer-based model, and works on typical natural language understanding tasks. Its usability in tasks like text classification, reading comprehension, and sequence labeling has been fully tested.
- **Flexibility in configurations**. The distillation process is configured by configuration objects, which can be initialized from JSON files and contain many tunable hyperparameters. Users can extend the configurations with new custom losses, schedulers, etc., if the presets do not meet their requirements.
- **Including various distillation methods and strategies**. KD has been studied extensively in computer vision (CV) and has achieved great success. It would be worthwhile to introduce these studies to the NLP community as some of the methods in these studies could also be applied to texts. TextBrewer includes a set of methods from both CV and NLP, such as flow of solution procedure (FSP) matrix loss (Yim et al., 2017), neuron selectivity transfer (NST) (Huang and Wang, 2017), probability shift and dynamic temperature (Wen et al., 2019), attention matrix loss, multi-task distillation (Liu et al., 2019a). In our experiments, we will show the effectiveness of applying methods from CV on NLP tasks.
- **Being non-intrusive and simple to use**. *Non-intrusive* means there is no need to modify the existing code that defines the models. Users can re-use the most parts of their existing training scripts, such as model definition and initialization, data preprocessing and task evaluation. Only some preparatory work (see Section 3.3) are additionally required to use TextBrewer to perform the distillation.

TextBrewer also provides some useful utilities such as model size analysis and data augmentation to help model design and distillation.

2 Related Work

Recently some distilled BERT models have been released, such as DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2019), and ERNIE Slim[2]. DistilBERT performs distillation on the pre-training task, i.e., masked language modeling. TinyBERT performs transformer distillation at both the pre-training and task-specific learning stages. ERNIE Slim distills ERNIE (Sun et al., 2019b,c)on a sentiment classification task. Their distillation code is publicly available, and users can replicate their experiments easily. However, it is laborious and error-prone to change the distillation method or adapt the distillation code for some other models and tasks, since the code is not written for general distillation purposes.

There also exist some libraries for general model compression. Distiller (Zmora et al., 2018) and PaddleSlim[3] are two versatile libraries supporting pruning, quantization and knowledge distillation. They focus on models and tasks in computer vision. In comparison, TextBrewer is more focused on knowledge distillation on NLP tasks, more flexible, and offers more functionalities. Based on PyTorch, It provides simple APIs and rich customization for fast and clean implementations of experiments.

3 Architecture and Design

Figure 1 shows an overview of the main functionalities and architecture of TextBrewer. To support different models and different tasks and meanwhile stay flexible and extensible, TextBrewer provides *distillers* to conduct the actual experiments and configuration classes to configure the behaviors of the distillers.

3.1 Distillers

Distillers are the cores of TextBrewer. They automatically train and save models and support custom evaluation functions. Five distillers have been implemented: `BasicDistiller` is used for single-task single-teacher distillation; `GeneralDistiller` in addition supports more advanced intermediate loss functions; `MultiTeacherDistiller` distills an ensemble of teacher models into a single student model; `MultiTaskDistiller` distills multiple teacher models of different tasks into a single multi-task student model (Clark et al., 2019; Liu et al., 2019a). We also have implemented `BasicTrainer` for training teachers on labeled data to unify the workflows of supervised learning and distillation. All the distillers share the same interface and usage. They can be replaced by each other easily.

[2]https://github.com/PaddlePaddle/ERNIE

[3]https://github.com/PaddlePaddle/PaddleSlim

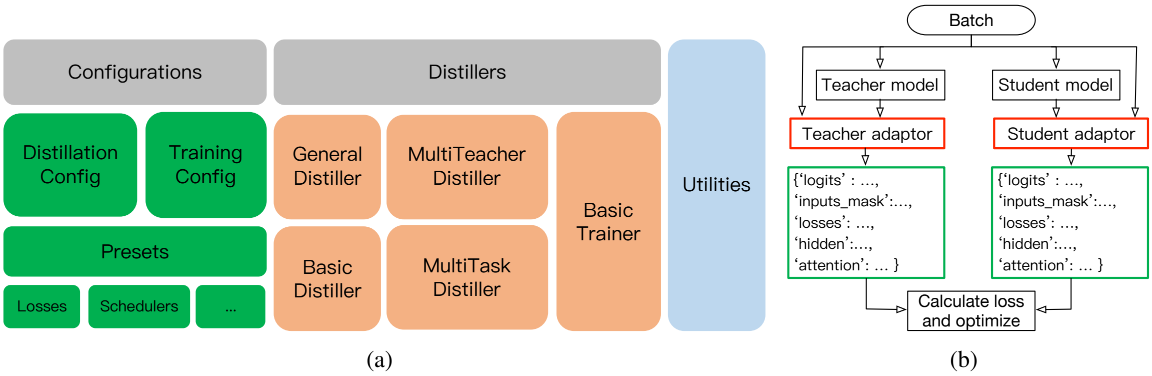

Figure 1: (a) An overview of the main functionalities of TextBrewer. (b) A sketch that shows the function of adaptors inside a distiller.

3.2 Configurations and Presets

The general training settings and the distillation method settings of a distiller are specified by two configurations: `TrainingConfig` and `DistillationConfig`.

TrainingConfig defines the settings that are general to deep learning experiments, including the directory where logs and student model are stored (`log_dir`, `output_dir`), the device to use (`device`), the frequency of storing and evaluating student model (`ckpt_frequencey`), etc.

DistillationConfig defines the settings that are pertinent to distillation, where various distillation methods could be configured or enabled. It includes the type of KD loss (`kd_loss_type`), the temperature and weight of KD loss (`temperature` and `kd_loss_weight`), the weight of hard-label loss (`hard_label_weight`), probability shift switch, schedulers and intermediate losses, etc. Intermediate losses are used for computing the losses between the intermediate states of teacher and student, and they could be freely combined and added to the distillers. Schedulers are used to adjust loss weight or temperature dynamically.

The available values of configuration options such as loss functions and schedulers are defined as dictionaries in presets. For example, the loss function dictionary includes hidden state loss, cosine similarity loss, FSP loss, NST loss, etc.

All the configurations can be initialized from JSON files. In Figure 3 we show an example of `DistillationConfig` for distilling BERT$_{\text{BASE}}$, to a 4-layer transformers. See Section 4 for more details.

```python
from textbrewer import GeneralDistiller
from textbrewer import TrainingConfig, DistillationConfig

# We omit the initialization of models, optimizer, and dataloader.
teacher_model : torch.nn.Module = ...
student_model : torch.nn.Module = ...
dataloader : torch.utils.data.DataLoader = ...
optimizer : torch.optim.Optimizer = ...
scheduler : torch.optim.lr_scheduler = ...

def simple_adaptor(batch, model_outputs):
    # We assume that the first element of model_outputs
    # is the logits before softmax
    return {'logits': model_outputs[0]}

train_config = TrainingConfig()
distill_config = DistillationConfig()
distiller = GeneralDistiller(
    train_config=train_config, distill_config = distill_config,
    model_T = teacher_model, model_S = student_model,
    adaptor_T = simple_adaptor, adaptor_S = simple_adaptor)

distiller.train(optimizer, scheduler,
    dataloader, num_epochs, callback=None)
```

Figure 2: A code snippet that demonstrates the minimal TextBrewer workflow.

3.3 Workflow

Before distilling a teacher model using TextBrewer, some preparatory works have to be done:

1. Train a teacher model on a labeled dataset. Users usually train the teacher model with their own training scripts. TextBrewer also provides `BasicTrainer` for supervised training on a labeled dataset.

2. Define and initialize the student model.

3. Build a dataloader of the dataset for distillation and initialize the optimizer and learning rate scheduler.

The above steps are usually common to all deep learning experiments. To perform distillation, take the following additional steps:

1. Initialize training and distillation configurations, and construct a distiller.

2. Define *adaptors* and a *callback* function.

3. Call the `train` method of the distiller.

A code snippet that shows the minimal workflow is presented in Figure 2. The concepts of callback and adaptor will be explained below.

```
{"temperature": 8,
 "temperature_scheduler": 'none'
 "hard_label_weight": 0,
 "hard_label_weight_scheduler": 'none',
 "kd_loss_type": "ce",
 "kd_loss_weight": 1,
 "kd_loss_weight_scheduler": 'none',
 "probability_shift": False,
 "intermediate_matches": [
 {'layer_T':0, 'layer_S':0, 'feature':'hidden',
  'loss': 'hidden_mse', 'weight' : 1,'proj':['linear',312,768]},
 {'layer_T':3, 'layer_S':1, 'feature':'hidden',
 'loss': 'hidden_mse',  'weight' : 1,'proj':['linear',312,768]},
 {'layer_T':6, 'layer_S':2, 'feature':'hidden',
 'loss': 'hidden_mse',  'weight' : 1,'proj':['linear',312,768]},
 {'layer_T':9, 'layer_S':3, 'feature':'hidden',
 'loss': 'hidden_mse',  'weight' : 1,'proj':['linear',312,768]},
 {'layer_T':12, 'layer_S':4, 'feature':'hidden',
 'loss': 'hidden_mse',  'weight' : 1,'proj':['linear',312,768]},
 {'layer_T':[0,0], 'layer_S':[0,0], 'feature':'hidden',
  'loss': 'nst', 'weight': 1}
 {'layer_T':[3,3], 'layer_S':[1,1], 'feature':'hidden',
  'loss': 'nst', 'weight': 1}
 {'layer_T':[6,6], 'layer_S':[2,2], 'feature':'hidden',
  'loss': 'nst', 'weight': 1}
 {'layer_T':[9,9], 'layer_S':[3,3], 'feature':'hidden',
  'loss': 'nst', 'weight': 1}
 {'layer_T':[12,12],'layer_S':[4,4], 'feature':'hidden',
  'loss': 'nst', 'weight': 1}]}
```

Figure 3: An example of distillation configuration. This configuration is used to distill a 12-layer $\text{BERT}_{\text{BASE}}$ to a 4-layer T4-tiny.

3.3.1 Callback Function

To monitor the performance of the student model during training, people usually evaluate the student model on a development set at some checkpoints besides logging the loss curve. For example, in the early stopping strategy, users choose the best model weights checkpoint based on the performance of the student model on the development set at the end of each epoch. TextBrewer supports such functionality by providing the callback function argument in the `train` method, as shown in line 24 of Figure 2. The callback function takes two arguments: the student model and the current training step. At each checkpoint step (determined by `num_train_epochs` and `ckpt_frequencey`), the distiller saves the student model and then calls the callback function.

Since it is impractical to implement evaluation metrics and evaluation procedures for all NLP tasks, we encourage users to implement their own evaluation functions as the callbacks for the best practice.

3.3.2 Adaptor

The distiller is model-agnostic. It needs a translator to translate the model outputs into meaningful data. Adaptor plays the role of translator. An Adaptor is an interface and responsible for explaining the inputs and outputs of the teacher and student for the distiller.

Adaptor takes two arguments: the model inputs and the model outputs. It is expected to return a dictionary with some specific keys. Each key explains the meaning of the corresponding value, as shown in Figure 1 (b). For example, `logits` is the logits of final outputs, `hidden` is intermediate hidden states, `attention` is the attention matrices, `inputs_mask` is used to mask padding positions. The distiller only takes necessary elements from the outputs of adaptors according to its distillation configurations. A minimal adaptor only needs to explain logits, as shown in lines 11–14 of Figure 2.

3.4 Extensibility

TextBrewer also works with users' custom modules. New loss functions and schedulers can be easily added to the toolkit. For example, to use a custom loss function, one first implements the loss function with a compatible interface, then adds it to the loss function dictionary in the presets with a custom name, so that the new loss function becomes available as a new option value of the configuration and can be recognized by distillers.

4 Experiments

In this section, we conduct several experiments to show TextBrewer's ability to distill large pretrained models on different NLP tasks and achieve results are comparable with or even higher than the public distilled BERT models with similar numbers of parameters. [4]

4.1 Settings

Datasets and tasks. We conduct experiments on both English and Chinese datasets. For English datasets, We use MNLI (Wang et al., 2019) for text classification task, SQuAD1.1 (Rajpurkar et al., 2016) for span-extraction machine reading comprehension (MRC) task and CoNLL-2003 (Tjong Kim Sang and De Meulder, 2003) for named entity recognition (NER) task. For Chinese datasets, we use the Chinese part of XNLI

[4] More results are presented in the online documentation: https://textbrewer.readthedocs.io

Dataset	Task	Metrics	#Train	#Dev
MNLI	Classification	Acc	393K	20K
SQuAD	MRC	EM/F1	88K	11K
CoNLL-2003	NER	F1	23K	6K
XNLI	Classification	Acc	393K	2.5K
LCQMC	Classification	Acc	293K	8.8K
CMRC 2018	MRC	EM/F1	10K	3.4K
DRCD	MRC	EM/F1	27K	3.5K

Table 1: A summary of the datasets used in experiments. The size of CoNLL-2003 is measured in number of entities.

(Conneau et al., 2018), LCQMC (Liu et al., 2018), CMRC 2018 (Cui et al., 2019b) and DRCD (Shao et al., 2018). XNLI is the multilingual version of MNLI. LCQMC is a large-scale Chinese question matching corpus. CMRC 2018 and DRCD are two span-extraction machine reading comprehension datasets similar to SQuAD. The statistics of the datasets are listed in Table 1.

Models. All the teachers are BERT$_{\texttt{BASE}}$-based models. For English tasks, teachers are initialized with the weights released by Google[5] and converted into PyTorch format via Transformers[6]. For Chinese tasks, teacher is initialized with the pre-trained RoBERTa-wwm-ext [7] (Cui et al., 2019a). We test the performance of the following student models:

- T6 and T3 are BERT$_{\texttt{BASE}}$ with fewer layers of transformers. Especially, T6 has the same structure as DistilBERT (Sanh et al., 2019).
- T3-small is a 3-layer BERT with half BERT-base's hidden size and feed-forward size.
- T4-tiny is the same as TinyBERT, a 4-layer model with an even smaller hidden size and feed-forward size.
- BiGRU is a single-layer bidirectional GRU. Its word embeddings are taken from BERT$_{\texttt{BASE}}$.

T3-small and T4-tiny are initialized randomly. The model structures of the teacher and students are summarized in Table 3.

Training settings. To keep experiments simple, we directly distill the teacher model that has been trained on the task, while we do not perform task-irrelevant language modeling distillation in advance. The number of epochs ranges from 30 to 60, and the learning rate of students is 1e-4 for all distillation experiments.

[5]https://github.com/google-research/bert
[6]https://github.com/huggingface/transformers
[7]https://github.com/ymcui/Chinese-BERT-wwm

Model	MNLI		SQuAD		CoNLL-2003
	m	mm	EM	F1	F1
BERT$_{\texttt{BASE}}$	83.7	84.0	81.5	88.6	91.1
Public					
DistilBERT	81.6	81.1	79.1	86.9	-
TinyBERT	80.5	81.0	-	-	-
+DA	82.8	82.9	72.7	82.1	-
TextBrewer					
BiGRU	-	-	-	-	85.3
T6	83.6	84.0	80.8	88.1	90.7
T3	81.6	82.5	76.3	84.8	87.5
T3-small	81.3	81.7	72.3	81.4	78.6
T4-tiny	82.0	82.6	73.7	82.5	77.5
+DA	-	-	75.2	84.0	89.1

Table 2: Performance of BERT$_{\texttt{BASE}}$ (teacher) and various students on the development sets of MNLI and SQuAD, and the test set of CoNLL-2003. *m* and *mm* under MNLI denote the accuracies on matched and mismatched sections respectively.

Distillation settings. Temperature is set to 8 for all experiments. We add intermediate losses uniformly distributed among all the layers between teacher and student (except BiGRU). The loss functions we choose are `hidden_mse` loss which computes the mean square loss between two hidden states, and `NST` loss which is an effective method in CV. In Figure 3 we show an example of distillation configuration for distilling BERT$_{\texttt{BASE}}$ to a T4-tiny. Since their hidden sizes are different, we use `proj` option to add linear layers to match the dimensions. The linear layers will be trained together with the student automatically. We experiment with two kinds of distillers: `GeneralDistiller` and `MultiTeacherDistiller`.

4.2 Results on English Datasets

We list the public results (DistilBERT and TinyBERT) and our distillation results obtained by `GeneralDistiller` in Table 2. We have the following observations.

First, teachers can be distilled to T6 models with minor losses in performance. All the T6 models achieve 99% performance of the teachers, higher than the DistilBERT.

Second, T4-tiny outperforms TinyBERT though they share the same structure. This is attributed to the NST losses in the distillation configuration. This result proves the effectiveness of applying KD method developed in CV on NLP tasks.

Third, although T4-tiny has less parameters than T3-small, T4-tiny outperforms T3-small in most

Model	# Layers	Hidden size	Feed-forward size	# Parameters	Relative size
$\text{BERT}_{\texttt{BASE}}$ (teacher)	12	768	3072	108M	100%
T6	6	768	3072	65M	60%
T3	3	768	3072	44M	41%
T3-small	3	384	1536	17M	16%
T4-tiny	4	312	1200	14M	13%
BiGRU	1	768	-	31M	29%

Table 3: Model sizes of teacher and students. The number of parameters includes embeddings but does not include output layers.

Model	MNLI		SQuAD		CoNLL-2003
	m	mm	EM	F1	F1
Teacher 1	83.6	84.0	81.1	88.6	91.2
Teacher 2	83.6	84.2	81.2	88.5	90.8
Teacher 3	83.7	83.8	81.2	88.7	91.3
Ensemble	84.3	84.7	82.3	89.4	91.5
Student	**84.8**	**85.3**	**83.5**	**90.0**	**91.6**

Table 4: Results of multi-teacher distillation. All the models are $\text{BERT}_{\texttt{BASE}}$. Different teachers are trained with different random seeds. For each task, the ensemble is the average of three teachers' results.

Model	XNLI	LCQMC	CMRC 2018		DRCD	
	Acc	Acc	EM	F1	EM	F1
RoBERTa-wwm	79.9	89.4	68.8	86.4	86.5	92.5
T3	78.4	89.0	63.4	82.4	76.7	85.2
+DA	-	-	66.4	84.2	78.2	86.4
T3-small	76.0	88.1	46.1	71.0	71.4	82.2
+DA	-	-	58.0	79.3	75.8	84.8
T4-tiny	76.2	88.4	54.3	76.8	75.5	84.9
+DA	-	-	61.8	81.8	77.3	86.1

Table 5: Development set results for the teacher and various students on Chinese tasks.

cases. It may be a hint that narrow-and-deep models are better than wide-and-shallow models.

Finally, data augmentation (DA) is critical. For the experiments in the last line in Table 2, we use additional datasets during distillation: a subset of NewsQA (Trischler et al., 2017) training set is used in SQuAD; passages from the HotpotQA (Yang et al., 2018) training set is used in CoNLL-2003. The augmentation datasets significantly improve the performance, especially when the size of the training set is small, like CoNLL-2003.

We next show the effectiveness of `MultiTeacherDistiller`, which distills an ensemble of teachers to a single student model. For each task, we train three $\text{BERT}_{\texttt{BASE}}$ teacher models with different seeds. The student is also a $\text{BERT}_{\texttt{BASE}}$ model. The temperature is set to 8, and intermediate losses are not used. As Table 4 shows, for each task, the student achieves the best performance, even higher than the ensemble result.

5 Results on Chinese Datasets

The results on Chinese datasets are presented in Table 5. We notice that T4-tiny still outperforms T3-small on all tasks, which is consistent with their performance on English tasks. In the experiments with DA, CMRC 2018 and DRCD take each other's dataset as data augmentation. We observe that since CMRC 2018 has a relatively small training set, DA has a much more significant effect.

6 Conclusion and Future Work

In this paper, we present TextBrewer, a flexible PyTorch-based distillation toolkit for NLP research and applications. TextBrewer provides rich customization options for users to compare different distillation methods and build their strategies. We have conducted a series of experiments. The results show that the distilled models can achieve state-of-the-art results with simple settings.

TextBrewer also has its limitations. For example, its usability in generation tasks such as machine translation has not been tested. We will keep adding more examples and tests to expand TextBrewer's scope of application.

Apart from the distillation strategies, the model structure also affects the performance. In the future, we aim to integrate neural architecture search into the toolkit to automate the searching for model structures.

Acknowledgments

We would like to thank all anonymous reviewers for their valuable comments on our work. This work was supported by the National Natural Science Foundation of China (NSFC) via grant 61976072, 61632011, and 61772153.

References

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D. Manning, and Quoc V. Le. 2019. BAM! born-again multi-task networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5931–5937, Florence, Italy. Association for Computational Linguistics.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. 2018. XNLI: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2475–2485, Brussels, Belgium. Association for Computational Linguistics.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Ziqing Yang, Shijin Wang, and Guoping Hu. 2019a. Pre-training with whole word masking for chinese BERT. *CoRR*, abs/1906.08101.

Yiming Cui, Ting Liu, Wanxiang Che, Li Xiao, Zhipeng Chen, Wentao Ma, Shijin Wang, and Guoping Hu. 2019b. A span-extraction dataset for Chinese machine reading comprehension. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5886–5891, Hong Kong, China. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Zehao Huang and Naiyan Wang. 2017. Like what you like: Knowledge distill via neuron selectivity transfer. *CoRR*, abs/1707.01219.

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.

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

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. *CoRR*, abs/1904.09482.

Xin Liu, Qingcai Chen, Chong Deng, Huajun Zeng, Jing Chen, Dongfang Li, and Buzhou Tang. 2018. LCQMC:a large-scale Chinese question matching corpus. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1952–1962, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

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

Alec Radford. 2018. Improving language understanding by generative pre-training.

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

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Chih-Chieh Shao, Trois Liu, Yuting Lai, Yiying Tseng, and Sam Tsai. 2018. DRCD: a chinese machine reading comprehension dataset. *CoRR*, abs/1806.00920.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019a. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Yu Sun, Shuohuan Wang, Yu-Kun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. 2019b. ERNIE: enhanced representation through knowledge integration. *CoRR*, abs/1904.09223.

Yu Sun, Shuohuan Wang, Yu-Kun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2019c. ERNIE 2.0: A continual pre-training framework for language understanding. *CoRR*, abs/1907.12412.

Xu Tan, Yi Ren, Di He, Tao Qin, Zhou Zhao, and Tie-Yan Liu. 2019. Multilingual neural machine translation with knowledge distillation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Raphael Tang, Yao Lu, and Jimmy Lin. 2019. Natural language generation for effective knowledge distillation. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 202–208, Hong Kong, China. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A machine comprehension dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 191–200, Vancouver, Canada. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Tiancheng Wen, Shenqi Lai, and Xueming Qian. 2019. Preparing lessons: Improve knowledge distillation with better supervision. *CoRR*, abs/1911.07471.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Junho Yim, Donggyu Joo, Ji-Hoon Bae, and Junmo Kim. 2017. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 7130–7138.

Sanqiang Zhao, Raghav Gupta, Yang Song, and Denny Zhou. 2019. Extreme language model compression with optimal subwords and shared projections. *CoRR*, abs/1909.11687.

Neta Zmora, Guy Jacob, Lev Zlotnik, Bar Elharar, and Gal Novik. 2018. Neural network distiller.