Improved Knowledge Distillation for Pre-trained Language Models via Knowledge Selection

Chenglong Wang[1], Yi Lu[1]*, Yongyu Mu[1], Yimin Hu[1], Tong Xiao[1,2]† and Jingbo Zhu[1,2]

[1]NLP Lab, School of Computer Science and Engineering,
Northeastern University, Shenyang, China
[2]NiuTrans Research, Shenyang, China
{clwang1119,yilu001102}@gmail.com
{xiaotong,zhujingbo}@mail.neu.edu.cn

Abstract

Knowledge distillation addresses the problem of transferring knowledge from a teacher model to a student model. In this process, we typically have multiple types of knowledge extracted from the teacher model. The problem is to make full use of them to train the student model. Our preliminary study shows that: (1) not all of the knowledge is necessary for learning a good student model, and (2) knowledge distillation can benefit from certain knowledge at different training steps. In response to these, we propose an actor-critic approach to selecting appropriate knowledge to transfer during the process of knowledge distillation. In addition, we offer a refinement of the training algorithm to ease the computational burden. Experimental results on the GLUE datasets show that our method outperforms several strong knowledge distillation baselines significantly.

1 Introduction

Pre-trained language models (PLMs) have significantly advanced state-of-the-art on various natural language processing tasks, such as sentiment analysis (Bataa and Wu, 2019; Baert et al., 2020), text classification (Sun et al., 2019a; Arslan et al., 2021), and question answering (Yang et al., 2019). Despite the remarkable results, PLMs have a large number of parameters which make them expensive for deployment (Yang et al., 2019).

To solve this problem, recent works resort to knowledge distillation (KD) (Hinton et al., 2015) to compress and accelerate the PLMs (Sun et al., 2019b; Sanh et al., 2019; Li et al., 2021b; Liang et al., 2021). Its key idea is to transfer the knowledge from a large PLM (*i.e.*, teacher model) into a lightweight model (*i.e.*, student model) without a significant performance loss. In this process,

we typically have multiple types of knowledge extracted from the teacher model, such as response knowledge, feature knowledge, and relation knowledge (Gou et al., 2021). Recent works mainly focus on how the student model learns the transferred knowledge more efficiently, such as designing training schemes (Sun et al., 2019b; Mirzadeh et al., 2020; Jafari et al., 2021; Li et al., 2021a) or enriching task-specific data (Jiao et al., 2020; Liang et al., 2021). However, few works consider how to make full use of the multiple types of knowledge into the training of the student model.

Our preliminary study (see Section 4.1) shows that not all of the knowledge is necessary for learning a good student model when conducting distillation with diverse knowledge. Furthermore, inspired by dynamic KD (Li et al., 2021b), we assume that learning from certain knowledge at different training steps is beneficial to KD. We conduct probing experiments to verify this assumption in Section 4.2. Specifically, we repeat KD 200 times and randomly select knowledge at each training step. We find that different distilled student models have a distinct performance. For example, the best student model is nearly 10% higher than the worst one in terms of accuracy on the RTE dataset. In addition, it notably exceeds the student model that learns from fixed knowledge. Based on our preliminary study, we have the following suggestion: the distilled student model could achieve superior performance if it learns appropriate knowledge at each training step. This suggestion motivates us to investigate how to select the appropriate knowledge for the student model during the process of KD.

Based on the above findings, we propose an actor-critic approach to selecting appropriate knowledge to transfer at different training steps (Bhatnagar et al., 2009). This approach first uses an actor-critic algorithm to implement a knowledge selection module via a long-term reward optimization. This optimization can consider the influence

---

of knowledge selection on future training steps. Furthermore, we develop a multi-phase training approach that divides the distillation process into many phases and provides a particular reward at the phase end. Compared to the usual actor-critic algorithm, it can ease the burden of computing rewards. After that, we perform KD by employing the trained knowledge selection module to select knowledge at each training step.

We test the proposed approach on six GLUE datasets (Wang et al., 2019) which involve question answering, sentiment analysis, and textual entailment. Experimental results show that our method significantly outperforms the vanilla KD method (Hinton et al., 2015) and other competitive KD methods. Notably, results also show our $BERT_6$ (6-layer BERT) student model can achieve more than 98.5% of the performance of teacher $BERT_{BASE}$ while keeping much fewer parameters ($\sim$61%). In addition, we prove that when armed with data augmentation, our approach can yield further improvements and be superior to TinyBERT (Jiao et al., 2020), leading to a strong baseline with the data augmentation.

2 Related Work

Knowledge distillation (KD) (Hinton et al., 2015) is widely used to compress and accelerate the pre-trained language models (PLMs) (Jiao et al., 2020; Sun et al., 2020; Wang et al., 2020; Li et al., 2021b). Its core idea is to transfer the knowledge from a large PLM (*i.e.*, teacher model) into a lightweight model (*i.e.*, student model). Recent works on KD could be classified into three groups. The first group focused on utilizing various knowledge from the teacher model to distill the student model. For example, multiple teacher models are leveraged to provide a more diverse and accurate knowledge for the student model (Liu et al., 2019a; Yuan et al., 2021). Sun et al. (2019b) exploited knowledge from intermediate layers of the teacher model during distillation. Moreover, Weight Distillation (Lin et al., 2021) transfers the knowledge in parameters of the teacher model to the student model. The second group tended to design effective strategies to facilitate the student model to learn the knowledge from different types of knowledge, such as enriching the task-specific data (Jiao et al., 2020; Liang et al., 2021) and the two-stage learning framework (Turc et al., 2019; Jiao et al., 2020). The third group that has attracted less attention generally explored

appropriate data and teacher models for student models in the KD. For example, Li et al. (2021b) proposed a data selection strategy that only selects some vital data for the student model according to its competency. Yuan et al. (2021) attempted to adjust the weights to the multiple teacher models during distillation.

Different from these methods, in this paper, we are concerned with how to select appropriate knowledge for student models during the process of KD. To this end, we train an effective knowledge selection model via an actor-critic algorithm and employ it to select appropriate knowledge to transfer at each training step.

3 Background

3.1 Knowledge Distillation

In KD, the student model learns knowledge from the teacher model by mimicking corresponding behaviors. In the distillation process, this mimicking can be implemented by minimizing the following loss function:

$$\mathcal{L}_{KD} = \sum_{x \in \mathcal{X}} \mathcal{L}_{diff}(f^S(x), f^T(x)) \tag{1}$$

where $\mathcal{X}$ is the training dataset, x is an input sample, $f^S(\cdot)$ and $f^T(\cdot)$ are the functions of describing the behaviors of the student model and the teacher model, respectively. $\mathcal{L}_{diff}(\cdot)$ is a loss function that evaluates the difference between their behaviors. In the process of KD, due to many behaviors (*e.g.*, extracting features) existing in the student and the teacher model, we typically have multiple types of knowledge. Previous works mainly focus on designing elaborate $f^S(\cdot)$, $f^T(\cdot)$, and $\mathcal{L}_{diff}(\cdot)$ to encourage the student model to learn certain fixed knowledge better (Sun et al., 2019b; Jiao et al., 2020; Liang et al., 2021). Unlike them, in this work, we make full use of the multiple types of knowledge and select appropriate knowledge for the student model during the process of KD.

3.2 Knowledge Types

The student model can learn from various of knowledge extracted from the teacher model in KD. According to Gou et al. (2021), the types of knowledge can often be *Response Knowledge (ResK)*, *Feature Knowledge (FeaK)*, and *Relation Knowledge (RelK)*. In addition, we consider a *Finetune Knowledge (FinK)* derived from the training dataset on compressing PLMs. The overview of these knowledge types is following:

Knowledge	RTE		SST-2	
	BERT$_6$	BERT$_3$	BERT$_6$	BERT$_3$
FinK	65.8	58.1	89.7	88.0
ResK	**67.1**	56.8	90.2	87.5
FeaK	66.1	57.8	90.4	87.1
RelK	64.6	**58.9**	88.9	88.2
FinK&ResK	65.2	55.6	90.0	**88.3**
FinK&ResK&FeaK	64.6	56.7	90.0	87.0
FinK&ResK&FeaK&RelK	64.5	57.4	**90.7**	87.7

Table 1: Accuracies (%) of distilled student models on the RTE and SST-2 development sets. The teacher model is the BERT$_{\text{BASE}}$. The student models are the 6-layer BERT$_6$ and the 3-layer BERT$_3$, respectively. Note that the knowledge is fixed during the process of KD.

- **ResK**: ResK refers to the knowledge that the student model learns by mimicking the neural response of the last layer of the teacher model. It includes more information about the predicted results (Hinton et al., 2015).
- **FeaK**: FeaK denotes the feature representation that the student model learns from the outputs of the intermediate layers of the teacher model. Different from ResK, it contains more information about the calculation process of the teacher model.
- **RelK**: RelK represents the relationships between different layers or samples (Yim et al., 2017). It focuses more on the relationship between model layers.
- **FinK**: FinK is the knowledge that the student model learns from ground-truth labels by fine-tuning on the training dataset (Hinton et al., 2015; Devlin et al., 2019). Here, we also consider it as a knowledge type in KD.

In this work, we explore how to select appropriate knowledge from these types to transfer during the process of KD. To ensure transferring the above types of knowledge to the student model, we design loss functions $\mathcal{L}_{ResK}$, $\mathcal{L}_{FeaK}$, $\mathcal{L}_{RelK}$, and $\mathcal{L}_{FinK}$, respectively. The Appendix A presents the design details of loss functions.

4 Preliminary Study

Due to its unique information and character, we assume that each type of knowledge has a different impact on KD results. Based on this, we conduct preliminary studies to probe the relationship between knowledge and KD.

4.1 Different Knowledge for KD

We conduct an experiment using different knowledge to distill the student models individually. In

Task	Method	S_{best}	S_{worst}	Δ
RTE	Random-All	68.4	58.3	10.1
	Random-One	67.2	58.9	8.7
	Fixed-All	67.1	64.5	2.6
SST-2	Random-All	91.3	86.1	5.2
	Random-One	90.7	87.9	2.8
	Fixed-All	90.7	88.9	1.8

Table 2: Accuracies (%) of S_{best} and S_{worst} on the development sets. Fixed-All denotes that we use the same knowledge to train the student model at each training step. Its results are taken from Table 1.

the experiment, we tune the weights of each type of knowledge on the dev set, similar to tuning hyper-parameters. Table 1 compares these distilled student models. We can see that the student models have noticeable gaps in terms of accuracy while using different knowledge. It indicates that *not all knowledge is necessary for learning a good student model*. Interestingly, we can also see that the appropriate knowledge changes as the student model or the dataset changes. We conjecture that factors such as student capacity and sample complexity could affect transferring knowledge, which is consistent with Stanton et al. (2021)'s findings.

4.2 Different Knowledge at Training Steps

Inspired by dynamic KD (Li et al., 2021b), we assume that the appropriate knowledge may change dynamically because the student model and the sample keep changing during training. We conduct probe experiments to verify this assumption. We randomly select knowledge for the student model at each training step. Here, we employ this random strategy for the training steps of all epochs (*Random-All*) and the training steps of the first epoch (*Random-One*), respectively. To make experiments more general, both Random-All and Random-One are repeated 200 times.

We report the best and worst performance of the distilled student models denoted by S_{best} and S_{worst} in Table 2. From the results of Random-All, we can see that there is a conspicuous gap in accuracy scores between S_{best} and S_{worst}. Notably, S_{best} performs significantly better than S_{worst} by 10% on the RTE dataset. We conjecture that S_{best} means the student model is trained with the appropriate knowledge at most of the training steps, while S_{worst} is the opposite. In addition, we notice that S_{best} achieves 68.4% accuracy, which significantly surpasses all student models with a fixed knowledge. It indicates that *KD can benefit from certain knowledge at different training steps*. This

also motivates us to investigate the method for selecting appropriate knowledge.

For the results of Random-One, there is also a remarkable performance difference between S_{best} and S_{worst}, though only the training steps of the first epoch in KD involves random selection of knowledge. It shows that knowledge selection at the current training step affects the following training steps. Therefore, we should consider the future influence while selecting appropriate knowledge.

5 Method

5.1 Overview

In this work, our goal is to select appropriate knowledge to transfer during the process of KD. Considering the current selection may affect future training steps, we propose the actor-critic approach, which addresses the knowledge selection problem via a long-term reward optimization. Figure 1 gives an overview of our method. Our method involves two stages: training a knowledge selection module (KSM) and distilling the student model with the trained KSM. In the first stage, we use an actor-critic algorithm consisting of an actor and a critic to implement KSM. The actor outputs an action that selects knowledge based on a given state. The critic network predicts the action value, *i.e.*, the sum of future rewards. In the second stage, we employ the trained KSM to select appropriate knowledge and transfer them to the student model.

5.2 Definitions

In this work, we use an actor-critic algorithm to implement the KSM. Here, we first describe some key concepts of the actor-critic algorithm, including state, action, and reward.

State. At training step t, we use s_t to denote the corresponding state. Here, s_t should comprise sufficient information for selecting appropriate knowledge. To this end, we use the informative [CLS] embeddings of the last layers in the student and the teacher models (Devlin et al., 2019) to achieve it. Specifically, for the i-th sample at training step t, we use $\boldsymbol{c}_i^S$ and $\boldsymbol{c}_i^T$ to denote the [CLS] embedding of the last layer of the student and the teacher model, respectively. Then we utilize two trainable feature networks, $\mathcal{N}_{fea}^S$ and $\mathcal{N}_{fea}^T$, to extract the useful feature vector $\boldsymbol{v}$:

$$\boldsymbol{v}_i^S = \mathcal{N}_{fea}^S(\boldsymbol{c}_i^S), \quad \boldsymbol{v}_i^T = \mathcal{N}_{fea}^T(\boldsymbol{c}_i^T) \quad (2)$$

where both $\mathcal{N}_{fea}^S$ and $\mathcal{N}_{fea}^T$ consist of a 2-layer multi-layer perceptron (MLP) network. We concatenate all extracted feature vectors in given batch as s_t.

Action. Given the state s_t, an action a_t can be created, which is used to select the appropriate knowledge to transfer at training step t. Here, we present soft and hard actions to select appropriate knowledge subtly. The soft action determines how much to learn from each knowledge type, while the hard action selects one or more knowledge types for training the student model.

Reward. Here we define the immediate reward r_t as the cross-entropy loss difference on a development set after the student model is trained with selected knowledge. See Table 6 in Appendix for a comparison of different rewards.

5.3 Knowledge Selection Module

To improve the performance of the distillation model, we need to select the appropriate knowledge for transferring at different training steps. For this purpose, we implement a knowledge selection module via the actor-critic algorithm which consists of an actor and a critic. Here, we utilize long-term rewards to optimize it to consider the influence of knowledge selection on future training steps.

5.3.1 Actor and Critic

Actor. In this work, the actor network μ_θ is composed of a three-layer MLP with four output neurons. It takes a state s_t and outputs the soft action with a Sigmoid function:

$$a_t = \text{Sigmoid}(\mu_\theta(s_t)) \quad (3)$$

where $a_t = \{a_1^t, a_2^t, a_3^t, a_4^t\}$ contains four real values belonging to $[0, 1]$. We use a_m^t in a_t to denote the percentage of the selected m-th type of knowledge[*]. At training step t, the student model's loss is calculated as follows:

$$\mathcal{L}_t^{soft} = a_1^t \mathcal{L}_{FinK} + a_2^t \mathcal{L}_{ResK} + a_3^t \mathcal{L}_{FeaK} \\ + a_4^t \mathcal{L}_{RelK} \quad (4)$$

In addition, to directly select one or more types of knowledge, we obtain the hard action via a conditional function $g(\cdot)$:

$$g(a_m^t) = \begin{cases} 1 & a_m^t \geq \lambda \\ 0 & otherwise \end{cases} \quad (5)$$

---

[*]Here, we treat *FinK*, *ResK*, *FeaK*, and *RelK* as 1-th, 2-th, 3-th, and 4-th type of knowledge.

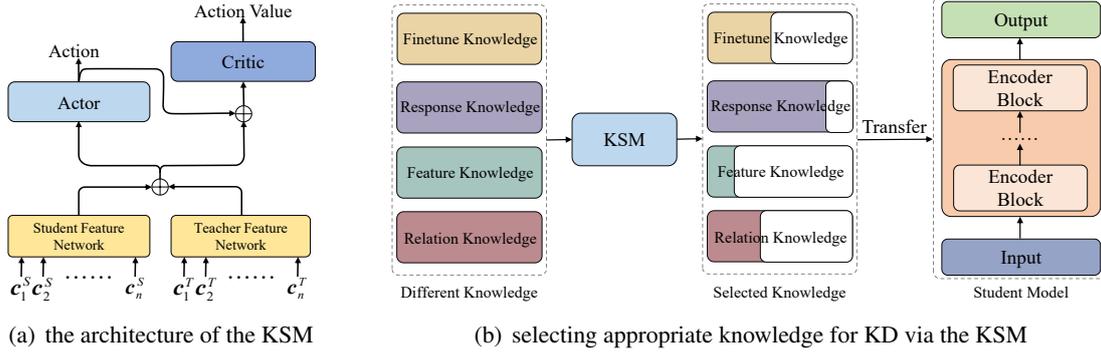(a) the architecture of the KSM   (b) selecting appropriate knowledge for KD via the KSM

Figure 1: An overview of the proposed method. We use an actor-critic approach to design a knowledge selection module (KSM), which aims to select the appropriate knowledge to transfer during KD. Besides the soft action presented in Figure 1(b), we also design the hard action for the actor module (see Section 5.3.1).

where $\lambda \in [0, 1]$ is a threshold value. The student model's loss with the hard action is as follows:

$$\mathcal{L}_t^{hard} = g(a_1^t)\mathcal{L}_{FinK} + g(a_2^t)\mathcal{L}_{ResK} + \\ g(a_3^t)\mathcal{L}_{FeaK} + g(a_4^t)\mathcal{L}_{RelK} \quad (6)$$

Here if $a_m^t \geq \lambda$, m-th type of knowledge is appropriate for the student model; otherwise, it is not.

Critic. The critic network Q_ϕ is also composed of a three-layer MLP. It computes the action value $Q_\phi(s_t, a_t)$ of a given pair (s_t, a_t), where s_t and a_t are concatenated as the input. The action value is an estimation of the sum of rewards earned after the actor takes the action a_t at the state s_t. Here, we define the action value as the long-term reward, *i.e.*, the performance gain of the student model from the training step t to the training end.

5.3.2   Optimization

Actor Optimization. Following the deep deterministic policy gradient algorithm (Silver et al., 2014), we optimize the long-term reward by updating the actor parameters via the sampled policy gradient:

$$\nabla_\theta J \approx \frac{1}{N}\sum_{t=1}^{N}\nabla_\theta Q_\phi(s_t, a_t) \quad (7)$$

Critic Optimization. At each training step, we can only gain an immediate reward r_t. However, the critic provides a long-term reward rather than an immediate reward. Therefore, we employ Temporal-Difference (Tesauro, 1991) to approximate the actual long-term reward with r_t:

$$Q_\phi^*(s_t, a_t) = \gamma^t r_t + Q_\phi(s_{t+1}, a_{t+1}) \quad (8)$$

where γ is a discount factor (Sutton and Barto, 2018). Then we can give the optimization objective of the critic network through the approximate long-term reward:

$$\mathcal{L}_Q = \frac{1}{N}\sum_{t=1}^{N}MSE(Q_\phi(s_t, a_t), Q_\phi^*(s_t, a_t)) \quad (9)$$

where $MSE(\cdot)$ is the mean squared error loss function, and N is the number of training steps in KD.

Feature Networks Optimization. During training KSM, we update the student feature network with the actor network and the teacher feature network with the critic network, respectively, in order to reduce the instability of training KSM.

5.3.3   Multi-Phase Training

Due to a large number of training steps in some datasets, computing the reward via the loss on the development set can yield considerable costs during training the KSM. For example, training with 5 epochs on the QQP dataset needs to compute the reward about 50,000 times in an episode. To ease the computational burden, we propose a multi-phase training approach to train the KSM. Specifically, we divide the complete KD process into *many phases* where each phase contains k training steps. After the student model is trained on a phase, we treat the cross-entropy loss difference on a development set as the phase reward r^p. It expresses the sum of all rewards in a phase. Here we can use a phase optimization objective $\mathcal{L}_Q^p$ to train the critic network instead of the $\mathcal{L}_Q$:

$$\mathcal{L}_Q^p = \frac{1}{N_p}\sum_{j=1}^{N_p}MSE(r_j^p, \sum_{t=b(j)}^{e(j)}\hat{r}_t) \quad (10)$$

where N_p is the number of phases in KD, $b(j)$ and $e(j)$ are the beginning and end training steps in the j-th phase. The $\hat{r}_t$ is an estimated reward computed by Eq. 8:

$$\hat{r}_t = \frac{Q_\phi(a_t, s_t) - Q_\phi(a_{t+1}, s_{t+1})}{\gamma^t} \quad (11)$$

Algorithm 1 Our Method

Input: the well-trained teacher model $\mathcal{M}_T$; the initial student model $\mathcal{M}_S$; training dataset $\mathcal{X}$
Output: the distilled $\mathcal{M}_S$;
1: *%%% the first stage*
2: Train the KSM via Algorithm 2;
3:
4: *%%% the second stage*
5: **for** $t = 1$ to N **do**
6:     compute a state s_t via Eq. 2;
7:     use an action $a_t = \mu_\theta(s_t)$ to select appropriate knowledge;
8:     train $\mathcal{M}_S$ via Eq. 4 or Eq. 6;
9: **end for**
10: return $\mathcal{M}_S$

Algorithm 2 Training of the KSM

Input: the well-trained teacher model $\mathcal{M}_T$; the initial student model $\mathcal{M}_S$; training dataset $\mathcal{X}$; the initial KSM (an actor μ_θ, a critic Q_ϕ, two feature networks $\mathcal{N}_{fea}^S$ and $\mathcal{N}_{fea}^T$)
Output: the trained KSM
1: **for** $episode = 1$ to K **do**
2:     reset $\mathcal{M}_S$;
3:     divide the whole KD process into N_p phases;
4:     **for** $j = 1$ to N_p **do**
5:         **for** $t = b(j)$ to $e(j)$ **do**
6:             compute a state s_t via Eq. 2;
7:             sample an action $a_t = \mu_\theta(s_t)$;
8:             utilize the actions in the previous phases to compute r_t^e via Eq. 12 or Eq. 13;
9:             compute an action value $Q_\phi(s, a)$;
10:            train $\mathcal{M}_S$ via Eq. 4 or Eq. 6;
11:         **end for**
12:         compute r_j^p via the loss difference;
13:         train the KSM via Eq. 9 and Eq. 10;
14:     **end for**
15: **end for**
16: return KSM

With the help of dividing phases, we can effectively relieve the computational burden. Generally, the conventional training of the actor-critic algorithm needs to compute rewards N times, while our multi-phase training only calculates rewards $\lceil N/k \rceil$ times.

5.3.4 Exploration Reward

To speed up the training of the KSM, we also design an exploration reward r_t^e at training step t, which encourages the actor to take more different actions (Tang et al., 2017). For the soft action, we consider the similarity between actions as r_t^e:

$$r_t^e = \alpha \times (1 - \frac{1}{k} \sum_{l=1}^{k} Sim(a_t, a_l)) \qquad (12)$$

where $Sim(\cdot)$ is a cosine similarity function, and a_l is the l-th action in the previous phase. In addition, we calculate exploration reward r_t^e for hard action

based on its repetition:

$$r_t^e = \alpha \times (1 - \frac{count(a_t)}{k}) \qquad (13)$$

where $count(a_t)$ is the number of times the action a_t was taken in the previous phase, and α is a scale factor. We provide the critic network with an additional optimization objective $\mathcal{L}_Q^e$ via r_t^e computed by Eq. 9.

5.4 Knowledge Distillation

Algorithm 1 presents our overall method. In the first stage, we train a KSM with an actor-critic algorithm. As described in Algorithm 2, we first divide the complete KD process into N_p phases in an episode (line 3). At each training step in a phase, we compute a state, an action, and an exploration reward (lines 6-9) and train the student model with selected knowledge (line 10). Then, we compute the phase reward when the phase ends. Subsequently, we optimize KSM via the $\mathcal{L}_Q^e$ and $\mathcal{L}_Q^p$ loss objectives computed by the Eq. 9 and Eq. 10, respectively (line 13). This process iterates on episodes until the performance of the student model converges. In the second stage, we adopt the trained KSM to select appropriate knowledge that is used to distill the student model.

6 Experiments

6.1 Datasets and Settings

Datasets. Following Sun et al. (2019b), we conduct experiments on six GLUE datasets (Wang et al., 2019): MNLI (Williams et al., 2018), QQP (Chen et al., 2018), QNLI (Rajpurkar et al., 2016), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), and RTE (Bentivogli et al., 2009).
Settings. We use BERT$_{\text{BASE}}$ as the teacher model and the BERT$_6$ and BERT$_3$ as the student models. The details of training settings are shown in Appendix B. All experiments are repeated three times, and we report the average results over three runs with different seeds.

6.2 Baselines

We compare the proposed method with **finetune** (Devlin et al., 2019), **vanilla KD** (Hinton et al., 2015), and other competitive pre-trained model KD methods such as **patient knowledge distillation (PKD)** (Sun et al., 2019b), **DistilBERT** (Sanh et al., 2019), and **Dynamic KD** (Li et al., 2021b).

In addition, we also design **Random-Soft** and **Random-Hard** baselines to evaluate the effectiveness of our method. Random-Soft and Random-

Method	Student	#Params	RTE (2.5k)	MRPC (3.7k)	MNLI-m/mm (393k)	SST-2 (67k)	QNLI (105k)	QQP (364k)	Avg.
BERT$_{BASE}$ (Teacher)	-	109.0M	69.3	87.6/83.5	84.1/83.1	94.3	90.5	71.0/89.2	84.9
PKD	BERT$_6$	67.0M	65.5	85.0/*79.9*	81.5/81.0	92.0	89.0	*70.7/88.9*	82.5
Dynamic KD	BERT$_6$	67.0M	-	86.5/-	81.8/81.0	-	-	-	-
DistilBERT	BERT$_6$	67.0M	58.4	86.9/-	82.6/81.3	*92.5*	88.9	70.1/-	-
Finetune	BERT$_6$	67.0M	65.2	85.1/79.2	81.1/79.8	91.7	87.1	69.4/88.2	81.8
Vanilla KD	BERT$_6$	67.0M	65.1	85.5/79.8	82.4/81.6	91.4	86.9	70.0/88.4	82.2
Random-Hard	BERT$_6$	67.0M	65.1	85.2/79.0	82.1/81.3	91.2	86.7	69.8/88.3	82.0
Random-Soft	BERT$_6$	67.0M	64.8	85.0/79.2	82.2/81.3	91.5	86.5	69.7/88.1	81.9
Our method (Hard Action)	BERT$_6$	67.0M	*66.6*	87.7/**82.2**	*82.6/81.8*	92.1	*89.0*	70.5/*88.9*	*83.3*
Our method (Soft Action)	BERT$_6$	67.0M	**66.8**	**87.9/82.2**	**83.1/82.1**	**92.6**	**89.3**	**71.1/89.1**	**83.6**
PKD	BERT$_3$	45.7M	58.2	80.7/*72.5*	76.7/76.3	87.5	*84.7*	*68.1/87.8*	77.7
Random-Hard	BERT$_3$	45.7M	54.3	80.7/71.8	76.8/76.1	87.4	82.9	66.4/86.4	76.5
Random-Soft	BERT$_3$	45.7M	55.7	78.9/69.7	77.1/76.3	87.7	82.3	66.4/86.5	76.5
Our method (Hard Action)	BERT$_3$	45.7M	*58.4*	*81.4/71.5*	*77.5/76.8*	**88.4**	84.6	*68.1/88.0*	*77.9*
Our method (Soft Action)	BERT$_3$	45.7M	**58.7**	**81.9/72.7**	**78.0/77.3**	*88.3*	**85.1**	**68.7/88.2**	**78.3**

Table 3: Results from the GLUE test server. The best and second-best results for each group of student models are in **bold** and in *italics*, respectively. The numbers under each dataset indicate the corresponding number of the training dataset. For MRPC and QQP, we report F1/Accuracy. We also report the average accuracy for each dataset in the "Avg." column. The results for Dynamic KD are achieved via the Uncertainty-Entropy strategy (Li et al., 2021b). The results of DistilBERT and PKD are taken from Jiao et al. (2020) and Sun et al. (2019b), respectively.

Hard denote that the KSM randomly takes a soft and hard action, respectively.

6.3 Main Results

We submit our model predictions to the official GLUE evaluation server, and the results are summarized in Table 3. First, compared with all baselines, our method can achieve optimal results while distilling the student models BERT$_6$ and BERT$_3$ on all the datasets. Second, compared with the Random-Hard (or Random-soft), we can observe that our method achieves significant improvements in the F1 and the accuracy scores. It demonstrates that the KSM system outperforms the baselines that randomly select knowledge to use. Third, the BERT$_6$ student model distilled by our method with soft action significantly outperforms the popular vanilla KD by a margin of at least 1.4%. We attribute this to the fact that KD benefits from selected knowledge at different training steps. Fourth, our method achieves the BERT$_6$ student model with ~61% parameters and achieves a similar performance compared with the teacher model BERT$_{BASE}$. For example, on the QQP dataset, the BERT$_6$ student model trained by our method with soft action has 89.1% accuracy, which only is 0.1% away from the teacher model. In addition, compared with the teacher model BERT$_{BASE}$, the BERT$_3$ student model has only 41% parameters while maintaining 92% performance.

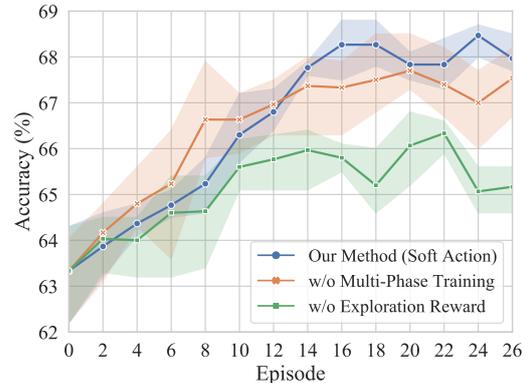Furthermore, we further investigate the performance gain on two different actions. From the

Figure 2: Ablation study. We plot the mean accuracy of student models distilled with three different seeds on the RTE development set.

results, we can find that the soft action performs better than the hard action, though both can contribute to our method to achieve an improvement over baselines on most of the datasets. One potential explanation might be that the soft action space is larger than the hard action space. Therefore, the soft action is more likely to explore a better knowledge selection than the hard action.

6.4 Ablation Study

We conduct an ablation study to explore the effects of the proposed multi-phase training and exploration reward on accuracy and efficiency. Figure 2 shows the accuracy of student models distilled with the KSM on the RTE development set after removing multi-phase training or exploration reward. We can see that using the multi-phase training can train a good KSM more likely, though the number
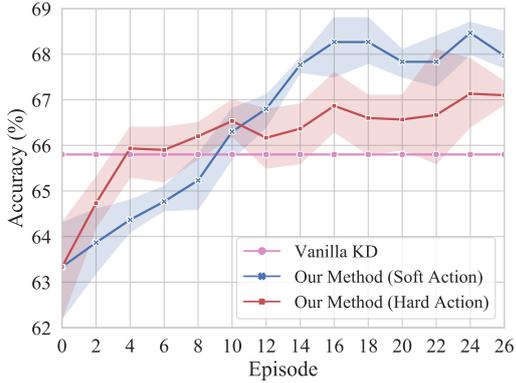
Figure 3: Comparison of the soft and hard actions. Our method can achieve faster KSM learning with the hard action and gain better results with the soft action.

Figure 4: Performance of distilled student model with various layers on the MRPC development set.

Method	RTE (52k)	MRPC (80k)	SST-2 (1,084k)	QNLI (2,237k)
Finetune	64.8	86.6/81.3	81.8	84.0
Vanilla KD	67.3	*88.0/82.8*	91.2	87.0
TinyBERT	*70.0*	87.3/82.6	*93.1*	*90.4*
Our Method	**70.4**	**88.9/83.9**	**93.8**	**90.8**

Table 4: Results of our method with the augmented datasets from the GLUE test server. The results of TinyBERT are taken from Jiao et al. (2020).

of offering a reward is reduced. We conjecture that the underlying reason is that the phase reward may be more accurate and stable than the immediate reward at each training step. In addition, we can also observe that without the exploration reward, the KSM fails to explore the better knowledge selection more quickly.

6.5 Discussion

Performance on Data Augmentation   Although the trained KSM can select the appropriate knowledge, the use of the fewer samples in the training dataset fail to provide the opportunity for the student model to learn them (Jiao et al., 2020; Liang et al., 2021). In such cases, we assume that if armed with the data augmentation, the student model could learn the knowledge assigned by KSM more sufficiently, thus achieving better performance. To this end, we augment the training datasets (*i.e.*, RTE, MRPC, SST-2, and QNLI datasets) about 20 times via the data augmentation procedure (Jiao et al., 2020). To make a fair comparison, we initialize our student model with the released $BERT_6$[†] distilled via General Distillation (Jiao et al., 2020). Table 4 compares our student model (soft action) with vanilla KD, finetune, and TinyBERT.

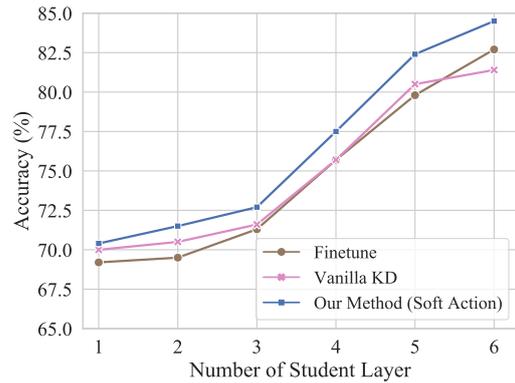[†] https://huggingface.co/huawei-noah/TinyBERT_General_6L_768D

The results show that our method is consistently better than all baselines. It demonstrates that data augmentation can significantly improve the student models distilled by our method. In addition, compared to the vanilla KD, our method can take advantage of the data more efficiently and transfer more knowledge to the student model.

This performance on data augmentation emphasizes that our method is orthogonal to the Tiny-BERT, as we address the knowledge selection problem during KD. It also provides a suggestion that our method can be complementary to other KD methods, such as the MixKD (Liang et al., 2021) and MiniLM (Wang et al., 2020).

Comparison of Soft Action and Hard Action
To intuitively present the effect of different actions on performance, we compare the soft and hard actions on the RTE dataset. Figure 3 shows the mean accuracy of the student models distilled with different action strategies on the development set. It can be found that our approach can integrate seamlessly with different strategies. Furthermore, we notice that the hard action can achieve faster KSM learning, while the soft action can achieve better results. We can draw similar observations on the results of other datasets, *e.g.*, on a relatively larger dataset QQP (see Figure 5 in Appendix).

Performance on Different Student Models   To explore the performance of different student models, we distill student models with different layers and plot the performance to compare baselines (*i.e.*, finetune and vanilla KD) in Figure 4. The results show that our method consistently outperforms the baselines while distilling the student model with various layers. It indicates that our method can be adapted to the distillation of different student models well.

A Regularization Perspective on Knowledge Selection In practice, we consider that the knowledge selection can act as a regularization which prevents the co-adaptation (Grisogono, 2006; Sabiri et al., 2022) in KD, *i.e.*, distilling a student model highly depends on a certain behavior of the teacher. If the distilled student model receives the inappropriate knowledge from the dependent behavior of the teacher, it can significantly alter the performance of the student model, which is what might happen with overfitting (Hawkins, 2004; Phaisangittisagul, 2016). However, knowledge selection allows the student model to learn from multiple different behaviors properly at each training step during KD. Thus, compared to the traditional KD methods that distill a student model via single or fixed knowledge, our method can ease the effect from the inappropriate knowledge from a behavior to prevent the co-adaptation.

See more discussion in Appendix C.

7 Conclusion

In this paper, we focus on making full use of the multiple types of knowledge into the distilling of the student model. We have proposed an actor-critic approach to selecting appropriate knowledge to transfer during the process of knowledge distillation. To ease the burden of computing rewards during training, we propose a multi-phase training approach and an exploration reward. Our experiments on GLUE datasets show that our method significantly outperforms several strong knowledge distillation baselines.

Limitations

In this section, we discuss some limitations of this work as follows:

- *We train a model to select appropriate knowledge to transfer during the process of knowledge distillation*. In this process, training the model is time-consuming and resource-intensive. For example, on the RTE dataset, if we utilize one TITAN V GPU and set the distillation epoch to 5, it will take 40 minutes to train the model.

- *It is difficult to scale the proposed approach to a large dataset*. The underlying reasons are: (*i*) More training steps in the supersize dataset will make the knowledge selection question more difficult. (*ii*) The larger dataset, the

state is more complex, which may confuse the knowledge selection module.

References

Yusuf Arslan, Kevin Allix, Lisa Veiber, Cedric Lothritz, Tegawendé F Bissyandé, Jacques Klein, and Anne Goujon. 2021. A comparison of pre-trained language models for multi-class text classification in the financial domain. In *Proceedings of WWW*.

Gaétan Baert, Souhir Gahbiche, Guillaume Gadek, and Alexandre Pauchet. 2020. Arabizi language models for sentiment analysis. In *Proceedings of COLING*.

Enkhbold Bataa and Joshua Wu. 2019. An investigation of transfer learning-based sentiment analysis in Japanese. In *Proceedings of ACL*.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *TAC*.

Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. 2009. Natural actor–critic algorithms. *Automatica*.

Zihan Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. 2018. Quora question pairs. *University of Waterloo*.

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

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of IWP*.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision*.

Anne-Marie Grisogono. 2006. Co-adaptation. In *Complex Systems*, volume 6039, pages 23–37.

Douglas M Hawkins. 2004. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *Proceedings of NeurIPS*.

Aref Jafari, Mehdi Rezagholizadeh, Pranav Sharma, and Ali Ghodsi. 2021. Annealing knowledge distillation. In *Proceedings of EACL*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Proceedings of EMNLP Findings*.

Bei Li, Ziyang Wang, Hui Liu, Quan Du, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2021a. Learning light-weight translation models from deep transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13217–13225.

Lei Li, Yankai Lin, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. 2021b. Dynamic knowledge distillation for pre-trained language models. In *Proceedings of EMNLP*.

Kevin J. Liang, Weituo Hao, Dinghan Shen, Yufan Zhou, Weizhu Chen, Changyou Chen, and Lawrence Carin. 2021. Mixkd: Towards efficient distillation of large-scale language models. In *Proceedings of ICLR*.

Ye Lin, Yanyang Li, Ziyang Wang, Bei Li, Quan Du, Tong Xiao, and Jingbo Zhu. 2021. Weight distillation: Transferring the knowledge in neural network parameters. pages 2076–2088. Proceedings of ACL.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. In *Proceedings of ACL*.

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. *arXiv preprint arXiv:1907.11692*.

Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. 2020. Improved knowledge distillation via teacher assistant. In *Proceedings of AAAI*.

Ekachai Phaisangittisagul. 2016. An analysis of the regularization between l2 and dropout in single hidden layer neural network. In *ISMS*, pages 174–179. IEEE.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*.

Bihi Sabiri, Bouchra El Asri, and Maryem Rhanoui. 2022. Mechanism of overfitting avoidance techniques for training deep neural networks. In *ICEIS (1)*, pages 418–427.

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

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. 2014. Deterministic policy gradient algorithms. In *Proceedings of ICML*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*.

Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A Alemi, and Andrew G Wilson. 2021. Does knowledge distillation really work? *Proceedings of NeurIPS*.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019a. How to fine-tune bert for text classification? In *Proceedings of CCL*.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019b. Patient knowledge distillation for BERT model compression. In *Proceedings of EMNLP*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of ACL*.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. 2017. #exploration: A study of count-based exploration for deep reinforcement learning. In *Proceedings of NeurIPS*.

Gerald Tesauro. 1991. Practical issues in temporal difference learning. *Proceedings of NeurIPS*, 4.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

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 *Proceedings of EMNLP WorkShop*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Proceedings of NeurIPS*, 33:5776–5788.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL.*

Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini. In *Proceedings of NAACL (Demonstrations)*, pages 72–77.

Junho Yim, Donggyu Joo, Ji-Hoon Bae, and Junmo Kim. 2017. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of CVPR.*

Fei Yuan, Linjun Shou, Jian Pei, Wutao Lin, Ming Gong, Yan Fu, and Daxin Jiang. 2021. Reinforced multi-teacher selection for knowledge distillation. In *Proceedings of AAAI.*

A Design Details of Learning Each Knowledge Type

A.1 Response Knowledge

For learning the response knowledge, we use the vanilla KD loss function:

$$\mathcal{L}_{ResK} = \sum_{x \in B} KL(\sigma(\frac{f^S(x)}{\tau}), \sigma(\frac{f^T(x)}{\tau})) \quad (14)$$

where $KL(\cdot)$ is the Kullback-Leibler divergence function, $\sigma(\cdot)$ is the softmax function and τ is a temperature hyper-parameter (Hinton et al., 2015). For the sample x, $f^S(x)$ and $f^T(x)$ are the final outputs of the student model and the teacher model, respectively.

A.2 Feature Knowledge

For learning the feature knowledge, we utilize the PKD-Skip method (Sun et al., 2019b), where the student model learns the outputs of the assigned teacher model's layers. The corresponding loss function can be defined by:

$$\mathcal{L}_{FeaK} = \sum_{x \in B} \sum_{i=1}^{L_S} \left\| \frac{h_{x,i}^S}{\left\| h_{x,i}^S \right\|_2} - \frac{h_{x,I(i)}^T}{\left\| h_{x,I(i)}^T \right\|_2} \right\|_2 \quad (15)$$

where $I(i)$ is the assigned indexes of the teacher layer for responding the i-th layer of the student model, L_S is the number of layers of the student model, $h_{x,i}^S$ and $h_{x,i}^T$ are the [CLS] embeddings of i-th layer of the student model and the teacher model for the sample x, respectively.

A.3 Relation Knowledge

For learning the relation knowledge, we use the flow of solution procedure (FSP) method (Yim et al., 2017). Specifically, we first define the FSP matrix by:

$$G(h_1, h_2) = \frac{h_1 \times h_2^\top}{|h_1|} \quad (16)$$

where h_1 and h_2 are the [CLS] embeddings. Let $G^S(\cdot)$ and $G^T(\cdot)$ to denote the FSP matrices of the student model and the teacher model, respectively. Based on this two FSP matrices, we can achieve the loss function for learning the relation knowledge:

$$\mathcal{L}_{RelK} = \sum_{x \in B} \sum_{i=1}^{L_S-1} MSE(G^S(h_{x,i}^S, h_{x,i+1}^S), \\ G^T(h_{x,I(i)}^T, h_{x+1,I(i+1)}^T)) \quad (17)$$

where $MSE(\cdot)$ is the mean squared error loss function. Here we use the same layer selection strategy for the teacher model as the PKD-Skip method.

A.4 Finetune Knowledge

For learning the finetune knowledge, we use the loss function:

$$\mathcal{L}_{FinK} = -\sum_{x \in B} \mathbf{1}\{y = \hat{y}\} \log p(y|x) \quad (18)$$

where $\mathbf{1}\{\cdot\}$ is the indicator function, y is the predicted label and $\hat{y}$ is the ground-truth label of the sample x.

B Training Setups

B.1 Training the KSM

We implement the KSM described in Sec. 5 based on PyTorch[‡]. Specifically, we use a 2-layer MLP to achieve the student and teacher feature networks. The input layer size and output layer size are the [CLS] embedding size and 8, respectively. Both the actor network and the critic network are composed of a 3-layer MLP. The hidden size and output layer size in the actor are 256 and 4. The hidden size and output layer size in the critic are 256 and 1. In addition, there are six hyper-parameters for training the KSM. The candidate values for these hyper-parameters are introduced in Table 5. During training the KSM, we can select the optimal hyper-parameter setups with the best development set accuracy.

Hyper-parameter	Value
Threshold value λ	{0.1, 0.2, 0.3}
Number of training steps k	{32, 64, 96, 128}
Scale factor α	{0.1, 0.2}
Discount factor γ	0.98
Learning rate of the actor ϵ_ϕ	0.0002
Learning rate of the critic ϵ_θ	0.0002

Table 5: Hyper-parameters for training the KSM.

B.2 Training the Teacher Model

In this work, we denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A for BERT. We use the pre-trained language model **BERT$_{BASE}$** ($L = 12$, $H = 768$, $A = 12$, $TotalParameters = 110M$) (Devlin et al., 2019) as the teacher model. We initialize the BERT$_{BASE}$ with the `bert-base-uncased`[§]. The teacher models are trained with random seeds. In addition, the sentence length is 128, and the learning rate is 5e-5. We set the batch size and training epoch to 32 and 5,

[‡] https://github.com/pytorch/pytorch
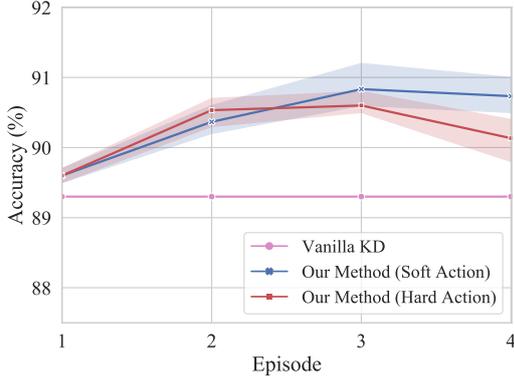[§] https://github.com/google-research/bert

Figure 5: Comparison of the soft and hard actions on the QQP development set. For the QQP dataset, we can train a well-performance KSM only across two episodes. This is because a large dataset can provide more opportunities for updating the KSM in each episode.

Reward	RTE	MRPC	QNLI	Avg.
Loss	**68.4**	**89.3/84.7**	**89.0**	**80.7**
F1	*66.9*	87.8/82.9	*88.1*	*79.3*
Accuracy	66.7	*88.1/83.2*	87.3	79.1

Table 6: Results of different rewards for training KSM.

respectively. Note that it is possible to plug-in any large pre-trained model such as BERT$_{\text{LARGE}}$ (Devlin et al., 2019) and RoBERTa (Liu et al., 2019b) in our method.

B.3 Training the Student Model

We primarily report results on two student models: **BERT**$_6$ ($L = 6$, $H = 768$, $A = 12$, $TotalParameters = 67.0M$) and **BERT**$_3$ ($L = 3$, $H = 768$, $A = 12$, $TotalParameters = 45.7M$). We initialize the BERT$_6$ and BERT$_3$ with the bottom 6 and 3 transformer layers of BERT$_{\text{BASE}}$, respectively. In training the student model, we set the batch size to 32 and the sentence length to 128. We select a well-trained student model through their scores among the learning rate set of {2e-5, 3e-5, 5e-5}.

C Discussion

Performance on Different Phase Sizes In Sec. 5.3.3, we divide the whole KD process into multiple phases. Then a question may arise about whether the phase size has an impact on the KSM performance. To probe this question, we run our method to distill the student model BERT$_6$ with different phase sizes. As shown in Figure 6, we observe that the excessive phase size can hurt the performance of the trained KSM, which we attribute to the deficiency of phase rewards. In addition, the
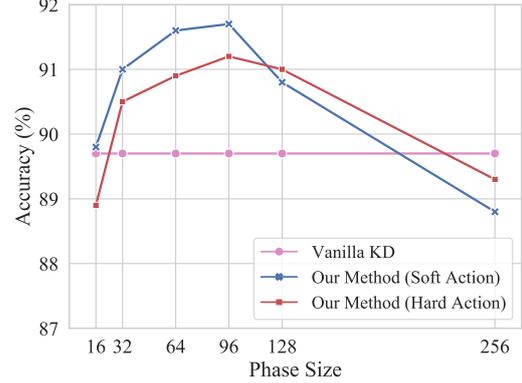

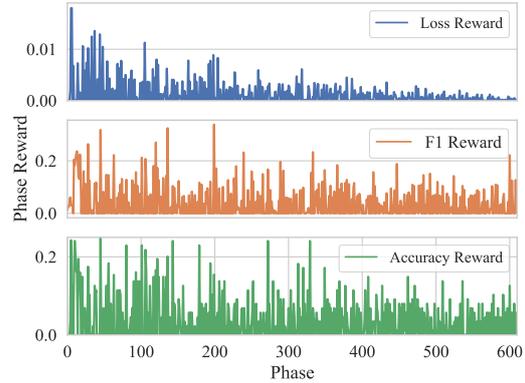Figure 6: Performance of the student models distilled by our method with different phase sizes.

Figure 7: Phase reward comparison of the different rewards in one episode. The loss reward is more stable and smooth than the F1 reward and the Accuracy reward.

too small phase size is not beneficial to our method because its phase reward is similar to an immediate reward.

Performance on Different Phase Rewards We investigate the performance of our method while taking the task metrics (*i.e.,* F1 or Accuracy) difference as the phase reward. As shown in Table 6, we observe that training KSM with a loss reward has the best performance overall. To explore the reasons for this observation, we further compare the phase rewards of different rewards in one episode, as shown in Figure 7. The results show that the loss reward is more stable and smooth than the F1 and Accuracy reward. Based on this, we conclude that the stability and smoothness give the loss reward the ability to train a better KSM.