Improved Knowledge Distillation by Utilizing Backward Pass Knowledge in Neural Networks

Aref Jafari
David R. Cheriton School of Computer Science
University of Waterloo
aref.jafari@uwaterloo.ca

Mehdi Rezagholizadeh
Noah's Ark Lab
Huawei
mehdi.rezagholizadeh@huawei.com

Ali Ghodsi
Department of Statistics and Actuarial Science
University of Waterloo
ali.ghodsi@uwaterloo.ca

Abstract

Knowledge distillation (KD) is one of the prominent techniques for model compression. Although conventional KD is effective for matching the two networks over the given data points, there is no guarantee that these models would match in other areas for which we do not have enough training samples. In this work, we address this problem by generating new auxiliary training samples based on extracting knowledge from the backward pass and identifying the areas where the student diverges greatly from the teacher. This is done by perturbing data samples in the direction of the gradient of the difference between the student and the teacher. We studied the effect of the proposed method on various tasks in different domains, including images and NLP tasks with considerably smaller student networks. Our experiments, show the proposed method got superior results over other baselines.

1 Introduction

During the last few years, we faced the emerge of a huge number of cumbersome state-of-the-art deep neural network models in different fields of machine learning, including computer vision (Wong et al., 2019; Howard et al., 2017), natural language processing (Prato et al., 2019; Jiao et al., 2019; Lan et al., 2019; Brown et al., 2020) and speech processing (Bie et al., 2019; He et al., 2019).

Because of limited computational resources of edge devices, it is infeasible to deploying these large models on them (Sun et al., 2020). On the other hand, considering users' privacy concerns, network reliability issues, and network delays increase the demand for offline machine learning solutions on edge devices. Because of this demand we need to compress large neural networks.

Knowledge distillation (KD) (Hinton et al., 2015) is one of the most prominent compression techniques in the literature. KD tries to transfer the learned knowledge from a large teacher network to a small student. Original KD method transfers knowledge from a teacher to a student network only by matching their forward pass outputs. Newer KD methods suggest other sources of knowledge in the teacher network such as using intermediate layer feature maps (Sun et al., 2019, 2020; Jiao et al., 2019), gradients of the network outputs w.r.t the inputs (Czarnecki et al., 2017; Srinivas and Fleuret, 2018)), and matching decision boundaries for classification tasks (Heo et al., 2019). using this additional information might be useful to get the student network performance closer to that of the teacher.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

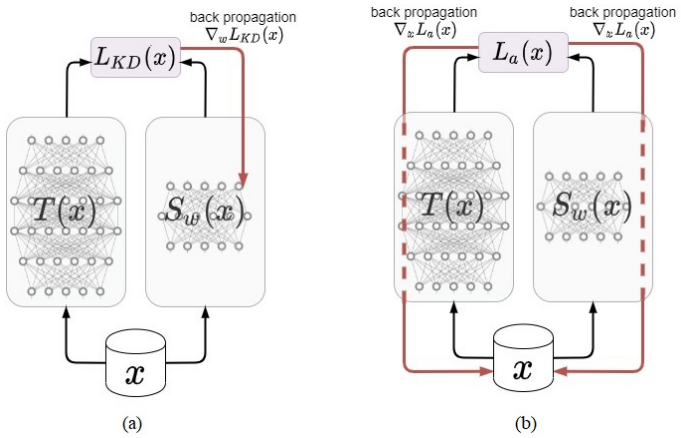


Figure 1: (a) Minimization Step: Using the teacher model knowledge for training the student in KD (utilizing forward knowledge) (b) Maximization Step: Augmenting the input dataset x with auxiliary data samples x' which is generated by the back propagation of gradient through both networks (utilizing backward knowledge)

In this work, we focus on identifying regions of the input space in which the teacher and student functions diverges from each other. Our proposed iterative backward KD approach is comprised of: first, a maximization step in which a new set of auxiliary training samples is generated by pushing training samples towards maximum divergence regions of the two functions; second, a minimization step in which the student network is trained using the regular KD approach over its training data together with the generated auxiliary samples from the first step.

We show the success of our backward KD technique in improving KD on both classification and regression tasks over the image and textual data.

2 Background: Knowledge Distillation

In the original KD, the process of transferring knowledge from a teacher to a student model is accomplished by minimizing a loss function between the logits of student and teacher networks. This loss function has been used in addition to the regular training loss function for the student network.

$$\mathcal{L}_{KD} = \alpha \mathcal{L}\Big(y, \sigma\big(S(x)\big)\Big) + (1-\alpha)\mathcal{L}\Bigg(\sigma\Big(\frac{T(x)}{\tau}\Big), \sigma\Big(\frac{S(x)}{\tau}\Big)\Bigg) \tag{1}$$

where $S(x)$ and $T(x)$ are student and teacher networks respectively, and σ is the Softmax function. τ is the temperature parameter and α is a coefficient between $[0, 1]$. This loss function is a convex combination of two loss terms. These loss functions minimize the distance between the student and both underlying and teacher functions. Since the teacher network is assumed to be a good approximation of the underlying function, it should be close enough to the underlying function of data samples. Fig. 2-(a) shows a simple example with three data points, an underlying function, a trained teacher and a potential student function that satisfies the KD loss function in Equation 1.

However, the problem is that even though the student satisfies the KD objective function and intersects the teacher function close to the training data samples, there is no guarantee that it would fit the teacher network in other regions of the input space as well. In this work, we try to address this problem by deploying the backward gradient information w.r.t the input (we refer to as backward knowledge) in the two networks.

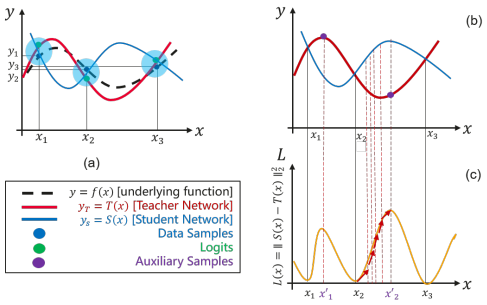


Figure 2: Visualizing the data insufficiency issue for the original KD algorithm. (a) behaviour of the teacher and the student function after training with KD loss. (b) divergence areas between the teacher and the student networks. (c) behaviour of $\mathcal{L}_{\mathcal{BKD}}$ loss function between teacher and the student and the idea of obtaining auxiliary data samples. For a better visualization, consider that $\mathcal{L}_{\mathcal{BKD}}$ is mean square error here.

3 Methodology: Improving Knowledge Distillation using Backward Pass Knowledge

3.1 Main Idea

In this section, we propose our improved KD method based on generating new out of sample points around the areas of the input domain where the student output diverges greatly from the teacher. This approach identifies the areas of the input space $\mathcal{X}$ around which the two functions have maximum distance. Then we generate out of sample points $X' \subset \mathcal{X}$ from the existing training set $X \subset \mathcal{X}$ over those regions. These new generated samples X' can be labelled by the teacher and then $X \leftarrow X \cup X'$ be deployed in the KD's training process to match the student better to the teacher over a broader range in the input space (see Fig. 2). We show that augmenting the training set by adding this auxiliary set improves the performance of KD significantly and leads to a closer match between the student and teacher. Our improved KD approach follows a procedure similar to the $minimax$ principle (Bratko and Gams, 1982) : first, in the maximization step we generate auxiliary data samples; second, in the minimization step we apply regular KD on the union of existing X and generated auxiliary data X'.

To have a better understanding of how this can be cast as an instance of minimax estimator, assume that we are given the data samples $\{x_i, T(x_i))\}_{i=1}^{N}$. The goal is to estimate $T(x)$ by $S(x)$. We may seek an estimator $S(x)$ attaining the $minimax$ principle. In minimax principle, where θ is an estimand and δ is an estimator, we evaluate all estimators according to its maximum risk $R(\theta, \delta)$. An estimator δ_0 , then, is said to be $minimax$ if:

$$\sup_{\theta} R(\theta, \delta_0) = \inf_{\delta \in C} \sup_{\theta \in \Theta} R(\theta, \delta) \tag{2}$$

That is we chose the estimator for the situation that the worst divergence between θ and δ is smallest. We follow a similar insight: i.e. the maximization step computes X', where there is the worst divergence between the teacher and the student. The minimization step finds the weights of the student network such that the difference between the student and teacher for this worst scenario is the smallest.

$$\min_{w} \max_{x} R(T_x, S_{x,w}) \tag{3}$$

In the main algorithm, we have two steps: maximization and minimization. The maximization step generates auxiliary data samples by perturbing training data samples based on the gradient of the backward KD loss $\mathcal{L}_{\mathcal{BKD}}$ which is a KL-divergence between the softmax outputs of teacher and student models. The minimization step uses the original and generated samples to minimize the $\mathcal{L}_{KD}$ loss function (Equation 1) to train the student model. The detains of these two steps can be find in appendix A.

3.2 Backward KD for NLP Applications

It is not trivial how to deploy the introduced backward KD approach (i.e. calculating $\nabla_x \mathcal{L}_{BKD}$ for discrete inputs) when data samples come from a discrete domain, such as NLP applications. To adapt the proposed method for NLP models, instead of perturbing input samples, we perturbed the embedding representations of the student and teacher models based on the gradient of $\mathcal{L}_{BKD}$ loss. Details of this method can be find in appendix B.

4 Experiments and Results

We designed five experiments to evaluate our proposed method.[1] But because of the page limitation, two of them are included here. Rest of the experiments can be found in appendix C section. Our first experiment is about the CIFAR-10 classification task. We used resNet-26 as the teacher and resNet-8 as student models. We trained the student network by using Backward KD method and compared its results with baselines: training student from scratch, the original knowledge distillation, 'FSP' Yim et al. (2017), and BSS Heo et al. (2019) (which is a adversarial attach based augmentation method). Also the performance of the combination of FSP+BSS has been reported. The results of thse experiments can be found in table 1. Since it is not clear how to use the last two baselines for text data, so we just used them for CIFAR-10 experiment. Table 1 compares the results of these experiments.

The next experiment is designed based on General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) and roBERTa family language models (Liu et al., 2019; Sanh et al., 2019). In these experiments, we fine-tuned the distilroBERTa model based on the proposed method by utilizing the fine-tuned roBERTa-large teacher for each of these tasks. Also, as our baselines, we trained the student model from scratch as well as with original KD, FreeLBZhu et al. (2019), and FreeLB+ original KD. Table 2 compares the results of these experiments. The results show that the overall score of Backward-KD outperforms other baselines.

Table 1: Results of experiment on CIFAR10 dataset

Model	method	accuracy on test set
resNet-26 (teacher)	from scratch	92.55%
resNet-8 (student)	from scratch	86.02%
resNet-8 (student)	original KD	86.66%
resNet-8 (student)	FSP+KD	87.07%
resNet-8 (student)	BSS	87.32%
resNet-8 (student)	FSP+BSS	87.52%
resNet-8 (student)	**Backward KD**	**88.4%**

Table 2: Results of experiment on GLUE tasks

Model (Network)	ColA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI	Score
roBERTa-large (Teacher)	68.1	96.4	91.9	92.3	91.5	90.2	94.6	86.3	56.3	85.28
DistilroBERTa (Student)	56.6	92.7	89.5	87.2	90.8	84.1	91.3	65.7	56.33	78.7
Student+FreeLB	58.1	93.1	90.1	88.8	90.9	84.0	91.0	67.8	56.33	80.01
Student+FreeLB+KD	58.1	**93.2**	**90.5**	88.6	91.2	83.7	90.8	68.2	56.33	80.06
Original KD	60.9	92.5	89.91	88.8	**91.6**	84.1	91.3	71.1	56.33	80.72
Our DistilroBERTa (Student)	**62.9**	93.1	90.3	**89.1**	91.5	**85.1**	**91.6**	**71.9**	56.33	**81.31**

5 Conclusion

In this paper, we have introduced the backward KD method and showed how we can use the backward knowledge of teacher model to train the student model. Based on this method, we could easily locate the diverge areas between teacher and student model in order to acquire auxiliary samples at those areas with utilizing the gradient of the networks and use these samples in the training procedure of the student model. We showed that our proposal can be efficiently applied to the KD procedure to improve its performance. Also, we introduced an efficient way to apply backward KD on discrete

[1]We used PyTorch (https://pytorch.org/) framework (Paszke et al., 2019) for implementing all experiments and Huggingface (https://huggingface.co/) framework (Wolf et al., 2019) in the implementations of NLP experiments.

domain applications such as NLP tasks. In addition to the synthetic experiment which is performed to visualize the mechanism of our method, we tested its performance on several image and NLP tasks. Also, we examined the extremely small student and the few sample scenarios in two of these experiments. We showed that the backward KD can improve the performance of the trained student network in all of these practices. We believe that all auxiliary samples do not have the same contribution to improving the performance of the student model. Also perturbing all data samples can be computationally expensive in large datasets.

References

Alex Bie, Bharat Venkitesh, Joao Monteiro, Md Haidar, Mehdi Rezagholizadeh, et al. Fully quantizing a simplified transformer for end-to-end speech recognition. *arXiv preprint arXiv:1911.03604*, 2019.

Ivan Bratko and Matjaz Gams. Error analysis of the minimax principle. In *Advances in computer chess*, pages 1–15. Elsevier, 1982.

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

Wojciech Marian Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Świrszcz, and Razvan Pascanu. Sobolev Training for Neural Networks. *arXiv e-prints*, art. arXiv:1706.04859, June 2017.

Yanzhang He, Tara N Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, et al. Streaming end-to-end speech recognition for mobile devices. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6381–6385. IEEE, 2019.

Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. Knowledge distillation with adversarial samples supporting decision boundary. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3771–3778, 2019.

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

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

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

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

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

Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *arXiv preprint arXiv:1902.03393*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. Fully quantized transformer for improved translation. *arXiv preprint arXiv:1910.10485*, 2019.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Suraj Srinivas and Francois Fleuret. Knowledge Transfer with Jacobian Matching. *arXiv e-prints*, art. arXiv:1803.00443, March 2018.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

Alexander Wong, Mahmoud Famuori, Mohammad Javad Shafiee, Francis Li, Brendan Chwyl, and Jonathan Chung. Yolo nano: a highly compact you only look once convolutional neural network for object detection. *arXiv preprint arXiv:1910.01271*, 2019.

Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4133–4141, 2017.

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

Appendix

A Details of Minimization and Maximization Steps in the Proposed Method

In this section we try to explain the details of minimization and maximization steps of backward KD method. The maximization step perturbs the input samples in a way to maximize the difference between the student and teacher models. The minimization step trains the student weights to minimize the KD loss function.

A.1 Maximization Step: Generating Auxiliary Data based on Backward-KD Loss

In the maximization step of our technique, we define a loss function (we refer to as the backward KD loss or BKD throughout this paper) to measure the distance between the output of the teacher and the student networks. This loss function can be Kullback Leibler divergence (KL-divergence) loss or mean square error loss based on the type of problem (classification or regression) and should be same as the loss in the minimization phase. Here, in this paper we use KL-divergence to demonstrate our method.

$$\mathcal{L}_{\mathcal{BKD}} = KL\Big(\sigma\Big(\frac{T(x)}{\tau}\Big), \sigma\Big(\frac{S(x)}{\tau}\Big)\Big) \tag{4}$$

where $\sigma(.)$ is the softmax function, $KL(.)$ is the KL-divergence loss, and τ is the temperature parameter. Here the main idea is that by taking the gradient of $\mathcal{L}_{\mathcal{BKD}}$ loss function in Equation 4 w.r.t the input samples, we can perturb the training samples along the directions of their gradients to increase the loss between two networks. Using this process, we can generate new auxiliary training

samples for which the student and the teacher networks are in maximum distance. To obtain these auxiliary data samples, we can consider the following optimization problem.

$$x' = \max_{x \in \mathcal{X}} \mathcal{L}_{\mathcal{BKD}} \tag{5}$$

We can solve this problem using stochastic gradient ascent method. Therefore our perturbation formula for each data sample will be:

$$x^{i+1} = x^i + \eta \, \nabla_x \, \mathcal{L}_{\mathcal{BKD}} \tag{6}$$

where in this formula η is the perturbation rate. This is an iterative algorithm and i is the iteration index. x^i is a training sample at i^{th} iteration. Each time, we perturb x^i by adding a portion of the gradient of loss to this sample. In general, if we continue the number of iterations until the convergence, there can be a risk for generating out of distribution samples. To avoid this issue and keep the distribution of generated auxiliary samples close to the distribution of original samples, in practice we do the perturbation steps for a limited number of iterations. That is because the data manifold is smooth (manifold assumption) and if we have limited number of data perturbation epochs, the auxiliary samples will stay on a locally linear patch of the manifold. See section B in the Appendix for more detail about this algorithm.

Fig. 2 demonstrates our idea using a simple example more clearly. Fig. 2-(a) shows a trained teacher and student functions given the training samples (x_1,y_1), (x_2,y_2), (x_3,y_3). Fig. 2-(c) constructs the $\mathcal{L}_{BKD}$ between these two networks. $\mathcal{L}_{BKD}$ shows where the two networks diverge in the original space. Bear in mind that $\mathcal{L}_{BKD}$ gives a scalar for each input. Hence, the gradient of $\mathcal{L}_{BKD}$ with respect to input variable x will be a vector with the same size as the variable x. Therefore, it does not need to deal with the large dimensionality issue of the Jacobian matrix as described in (Czarnecki et al., 2017). Fig. 2-(c) also illustrates an example of generating one auxiliary sample from the training sample x_2. If we apply Equation 6 to this sample, after several iterations, we will reach to a new auxiliary data point (x_2'). It is evident in Fig. 2-(a) that, as expected, there is a large divergence between the teacher and student networks in (x_2') point.

A.2 Minimization Step: Improving KD with Generated Auxiliary Data

We can apply the maximization step to the given training data to generate their corresponding auxiliary samples. Then by adding the auxiliary samples X' into the training dataset $X \leftarrow X' \cup X$, we can train the student network again based on the original KD algorithm over the updated training set in order to obtain a better output match between the student and teacher networks. Inspired by Mirzadeh et al. (2019), we have used the following KD loss function in our work:

$$\mathcal{L}_{KD} = (1-\lambda)\, H\Big(y, \sigma\big(S(x)\big)\Big) + \tau^2\, \lambda\, KL\bigg(\sigma\Big(\frac{T(x)}{\tau}\Big), \sigma\Big(\frac{S(x)}{\tau}\Big)\bigg) \tag{7}$$

where $\sigma(.)$ is the softmax function, $H(.)$ is the cross-entropy loss function, $KL(.)$ is the Kullback Leibler divergence, λ is a hyper parameter, τ is the temperature parameter, and y is the true labels.

The intuition behind expecting to get a better KD performance using the updated training data is as follows. Now given the auxiliary data samples which point toward the regions of the input space where the student and teacher have maximum divergence, these regions of input space are not dark for the original KD algorithm anymore. Therefore, it is expected from the KD algorithm to be able to match the student to the teacher network over a larger input space (see Fig. 4). Moreover, it is worth mentioning that the maximization and minimization steps can be taken multiple times. In this regard, for each maximization step, we need to construct the auxiliary set X' from scratch and we do not need the previously generated auxiliary sets. However, in our few-sample training scenarios where the number of data samples is small, we can keep the auxiliary samples. The maximization steps happen along with the regular KD training. For a better explanation, suppose regular KD needs $n = e \times (h+2)$ epochs to train the student network. First we perform the minimization step for e epochs. Then, after each minimization step, we perform the maximization step h time in order to generate the auxiliary samples, and enrich the training dataset to achieve a better match between the teacher and student models. These steps happen h times in the algorithm. Also, to pay more attention to the original data samples rather than the auxiliary data samples, at the end of the training, we fine-tune the student model with only the original data samples for e epochs.

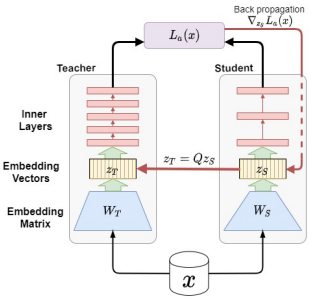


Figure 3: General procedure of utilizing auxiliary samples in NLP models. Here x is the one-hot vector of input tokens, W is the embedding matrix, and z is the embedding vector of x.

B Backward KD for NLP Applications

It is not trivial how to deploy the introduced backward KD approach (i.e. calculating $\nabla_x \mathcal{L}_{BKD}$ for discrete inputs) when data samples come from a discrete domain, such as NLP applications. Here, we propose a solution to how this technique can be adapted for the NLP domain. For neural NLP models, first, we pass the one-hot vectors of the input tokens to the so-called embedding layer of neural networks. Then, these one-hot vectors are converted into embedding vectors (see Fig. 3). The key for our solution is that embedding vectors of input tokens are not discrete and we can take the gradient of loss function w.r.t the embedding vectors z. But the problem is that, in the KD algorithm, we have two networks with different embedding sizes (see Fig. 3). To address this issue, we can take the gradient of the loss function w.r.t one of the embedding vectors (here student embedding vector z_S). However, then we need a transformation matrix like Q to be able to derive the corresponding embedding vector z_T for the teacher network form z_S.

$$z_T = Qz_S \tag{8}$$

We can show that the transform matrix Q is equal to the following equation:

$$Q = W_T W_S^T (W_S W_S^T)^{-1} \tag{9}$$

where in this equation $W_S^T (W_S W_S^T)^{-1}$ is the pseudo inverse of W_S embedding matrix. We refer you to the Appendix to see the proof of this derivation. Therefore, to obtain the auxiliary samples, we can take the gradient of the $\mathcal{L}_{BKD}$ loss function w.r.t the student embedding vector z_S. Then by using Equations 10 and 9, we can re-construct z_T during the steps of data perturbation as following.

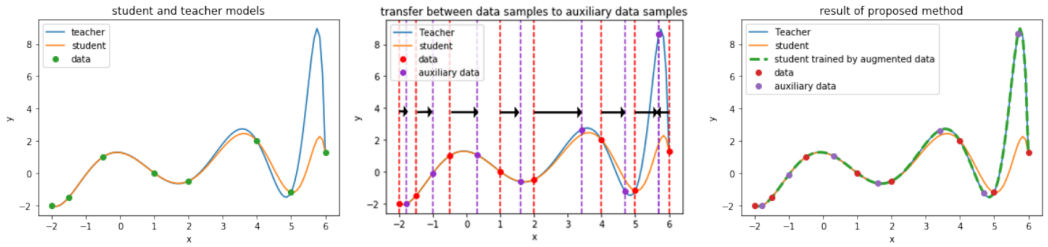
$$z_S^{i+1} = z_S^i + \eta \nabla_{z_S} \mathcal{L}_{BKD} \tag{10}$$

$$z_T^{i+1} = W_T W_S^T (W_S W_S^T)^{-1} z_S^{i+1} \tag{11}$$

C More Experiments

C.1 Synthetic Data Experiment

For visualizing our technique and showing the intuition behind it, we designed a very simple experiment to show how the proposed method works over a synthetic setting. In this experiment, we consider a polynomial function of degree 20 as the trained teacher function. Then, we considered 8 data points on its surface as our data samples to train a student network which is a polynomial function from degree 15 (see Fig. 4-(a)). As it is depicted in this figure, although the student model perfectly fits the given data points, it diverges from the teacher model in some areas between the given points. After applying the backward KD method, we can generate some auxiliary samples in the diverged areas between the teacher and student models in Fig. 4-(b). Then, we augmented the training data samples with the generated auxiliary samples and trained the student model based on this new augmented dataset. The resulting student model has achieved a much better fit on the teacher model as it is evident in Fig. 4-(c).



(a) (b) (c)

Figure 4: Visualizing the generation of auxiliary samples and their utilization in training the student model.

C.2 MNIST classification:

In this experiment, one of our goals was testing the performance of the proposed method in the scenario of extremely small student networks. Because of that, we considered two fully connected neural networks as student and teacher networks for the MNIST dataset classification task. The teacher network consists of only one hidden layer with 800 neurons which leads in 636010 trainable parameters. The student network was an extremely simplified version of the same network with only 5 neurons in the hidden layer. This network has only 3985 trainable parameters, which is 160x smaller than the teacher network. The student network is trained in three different ways: a) from scratch with only training data, b) based on the original KD approach with training data samples augmented by random noise, and c) based on the proposed method. As it is illustrated in table 1, the student network which is trained by using the proposed method achieves much better results in comparison with two other trained networks.

Table 3: Results of experiment on the MNIST dataset

Model	method	#parameters	accuracy on test set
teacher	from scratch	636010	98.14
student	from scratch	3985	87.62
student	original KD	3985	88.04
student	**proposed method**	3985	**91.45**

C.3 GLUE tasks with few sample points

In this experiment, we modified the experiment in section 6.3 of paper slightly to investigate the performance of the proposed method in the few data sample scenario. Here we randomly select a small portion of samples in each data set and fine-tuned the distilroBERTa based on these samples. For CoLA, MRPC, STS-B, QNLI, RTE, and WNLI, 10% of data samples and for SST-2, QQP, and MNLI 5% of them in the dataset are used for fine-tuning the student model.

Table 4: Results of few sample experiment on GLUE tasks

Model (Network)	ColA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI	Score
roBERTa-large (Teacher)	60.56	96.33	89.95	91.75	91.01	89.11	93.08	79.06	56.33	85.82
DistilroBERTa (Student)	43.82	91.05	76.96	81.51	84.92	75.88	83.94	52.07	56.33	71.90
Our DistilroBERTa (Student)	**44.11**	**91.74**	**77.20**	**82.82**	**85.32**	**76.75**	**84.34**	**56.31**	56.33	**72.76**

D Algorithms

This section explains the details of the proposed algorithm. First, we will see the general procedure of the algorithm. Then in section A.1, the pseudo-code of the algorithm will be explained; and finally, in section A.2, we will see the modified version of the algorithm for NLP tasks. If we consider for

training the student model, vanilla KD needs n epochs, here the idea is dividing these epochs into $h+2$ sections where each section consists of e epochs ($n=(h+2)e$) Then we have:

Pre-training Step (e epochs): We train the student network based on the original KD procedure for (e epochs). In this step, the student network will get close to the teacher network around the given training samples and will diverge from the teacher in some other areas.

Iterative Min Max Steps ($h \times e$ epochs): We do the following two steps iteratively h times:
1) Using the partially trained student network and the trained teacher network, we use the proposed maximization step in A.2 for generating an auxiliary dataset.
2) Combine the auxiliary data with the training dataset and train the student network based on the augmented dataset using the original KD procedure for e epochs.

Fine-tuning Step (e epochs): Finally, fine-tune the student network using original KD only based on the train samples for e epochs to pay more attention to the original data samples.

D.1 Algorithm 1 (general pseudo-code)

Algorithm 1 explains the details of the proposed method in previous section and section 3 of the paper.We pass the student network $S(.)$, the teacher network $T(.)$, the input dataset X, the number of training epochs e, the number of maximization steps h, and the number of sample perturbing steps l to the proposed KD function. This algorithm assumes that the teacher network $T(.)$ has been trained, and will be used to train the student network $S(.)$. Also, we assume X' is the set of the augmented data samples. We first initialize X' with data set X in line 3 of the algorithm. The basic idea is that each time we train the student network using the Vanilla-KD function for a few training epochs e in the outer loop of line 4. Then, in line 6 first, we re-initialize X' with dataset X, and in lines 7 to 11, we perturb data samples in X' using the gradient of the loss between teacher and student iteratively to generate new auxiliary samples. In line 12, we replace X with the union of X and X' sets. In the next iteration of the loop in line 4, the Vanilla-KD function will be fed with the augmented data samples X'. Note that just in the first iteration, Vanilla-KD function is fed with the original data set X which is identical to pre-training step of the previous section.

Algorithm 1

1: **function** PROPOSED-KD(S,T,X, e, h, l)
2: $X' \leftarrow X$
3: **for** $i = 1$ `to` $h+1$ **do**
4: VANILLA-KD(S,T,X',e)
5: $X' \leftarrow X$
6: **for** x' `in` X' **do**
7: **for** $j = 1$ `to` l **do**
8: $x' \leftarrow x' + \eta \nabla_x ||S(x') - T(x')||_2^2$
9: **end for**
10: **end for**
11: $X' \leftarrow X' \cup X$
12: **end for**
13: VANILLA-KD(S,T,X,e)
14: **return** S
15: **end function**

D.2 Algorithm 2 (NLP task's version)

Algorithm 2 explains how to apply the proposed method in NLP tasks. This algorithm is almost similar to algorithm 1. The only main difference is in the way we feed the networks. Here instead of considering the one-hot index vectors of tokens in the text documents, we consider the embedding vectors z_S and z_T of the input vector x (see lines 5 and 6 in the algorithm). Then we feed each of the teacher and the student networks separately using their own embedding vectors. In line 16, we use the transform method proposed in section 3.2 of the paper to transform student's perturbed embedding vectors into teacher's embedding vectors.

Algorithm 2

1: **function** PROPOSED-KD(S,T,X, e, h, l)
2: $W_T \leftarrow$ EMBEDDING-MATRIX(T)
3: $W_S \leftarrow$ EMBEDDING-MATRIX(S)
4: $Z_T \leftarrow W_T X$
5: $Z_S \leftarrow W_S X$
6: $Z'_T \leftarrow Z_T$
7: $Z'_S \leftarrow Z_S$
8: **for** $i = 1$ `to` $h + 1$ **do**
9: VANILLA-KD(S,T,Z'_T, Z'_S,e)
10: $Z'_T \leftarrow Z_T$
11: $Z'_S \leftarrow Z_S$
12: **for** (z'_S, z'_T) `in` (Z'_S, Z'_T) **do**
13: **for** $j = 1$ `to` l **do**
14: $z'_S \leftarrow z'_S + \eta \nabla_{z_S} ||S(z'_S) - T(z'_S)||_2^2$
15: $z'_T \leftarrow W_T W_S (W_S W_S^T)^{-1} z'_S$
16: **end for**
17: **end for**
18: $Z'_S \leftarrow Z'_S \cup Z_S$
19: $Z'_T \leftarrow Z'_T \cup Z_T$
20: **end for**
21: VANILLA-KD(S,T,Z_T, Z_S,e)
22: **return** S
23: **end function**