Improved Knowledge Distillation by Utilizing Backward Pass Knowledge in Neural Networks

by

Aref Jafari

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Waterloo, Ontario, Canada, 2020

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Knowledge distillation (KD) is one of the prominent techniques for model compression. In this method, the knowledge of a large network (teacher) is distilled into a model (student) with usually significantly fewer parameters. KD tries to better-match the output of the student model to that of the teacher model based on the knowledge extracts from the forward pass of the teacher network. 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 that problem by generating new auxiliary training samples based on extracting knowledge from the backward pass of the teacher in the areas where the student diverges greatly from the teacher. We compute the difference between the teacher and the student and generate new data samples that maximize the divergence. This is done by perturbing data samples in the direction of the gradient of the difference between the student and the teacher. Augmenting the training set by adding this auxiliary improves the performance of KD significantly and leads to a closer match between the student and the teacher. Using this approach, when data samples come from a discrete domain, such as applications of natural language processing (NLP) and language understanding, is not trivial. However, we show how this technique can be used successfully in such applications. We studied the effect of the proposed method on various tasks in different domains, including images and NLP tasks with considerably smaller student networks. The results of our experiments, when compared with the original KD, show 4% improvement on MNIST with a student network that is 160 times smaller, 1% improvement on a CIFAR-10 dataset with a student that is 9 times smaller, and an average 1.5% improvement on the GLUE benchmark with a distilroBERTa-base student.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Ali Ghodsi for his continuous support of my Master's research, for his patience, motivation, enthusiasm, and immense knowledge.

I would like to acknowledge Dr. Mehdi Rezagholizadeh and NLP team of Huawei technologies Canada Co. ltd. for their generous supports and helps.

I would also like to thank Prof. Hans De Sterck, Dr. Jeff Orchard, Chérisse Mike, and the rest of the Computational Mathematics faculty, staff, and students to create a great learning environment and a pleasant community to pursue my Master's Degree.

Dedication

To all who dedicated their lives to shed light on our unknowns.

Table of Contents

List of Tables

List of Figures

Chapter 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 [49, 17], natural language processing [31, 19, 21, 4] and speech processing [2, 14]. We need powerful servers to be able to deploy such large models. Running these models on edge devices would be infeasible due to the limited memory and computational power of edge devices [41]. 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. The field of neural model compression focuses on providing compression solutions such as quantization [18], pruning [45], Low rank tensor factorization [43] and knowledge distillation (KD) [16] for large neural networks.

Knowledge distillation (KD) is one of the most prominent compression techniques in the literature. As its name implies, KD tries to transfer the learned knowledge from a large teacher network to a small student. The idea of KD was proposed by [5] for the first time and later this idea generalized by [16] for deep neural nets. The original KD method concerns transferring knowledge from a teacher to a student network only by matching their forward pass outputs. Later on, several works in the literature suggested other sources of knowledge in the teacher network besides the logit outputs of the last layer. This includes using intermediate layer feature maps [40, 41, 19], gradients of the network outputs w.r.t the inputs [10, 38]), and matching decision boundaries for classification tasks [15]. Using this additional information might be useful to get the student network performance closer to that of the teacher.

In this work, we focus on identifying regions of the input space of the teacher and student networks in which the two functions diverge the most from each other. Moreover, we

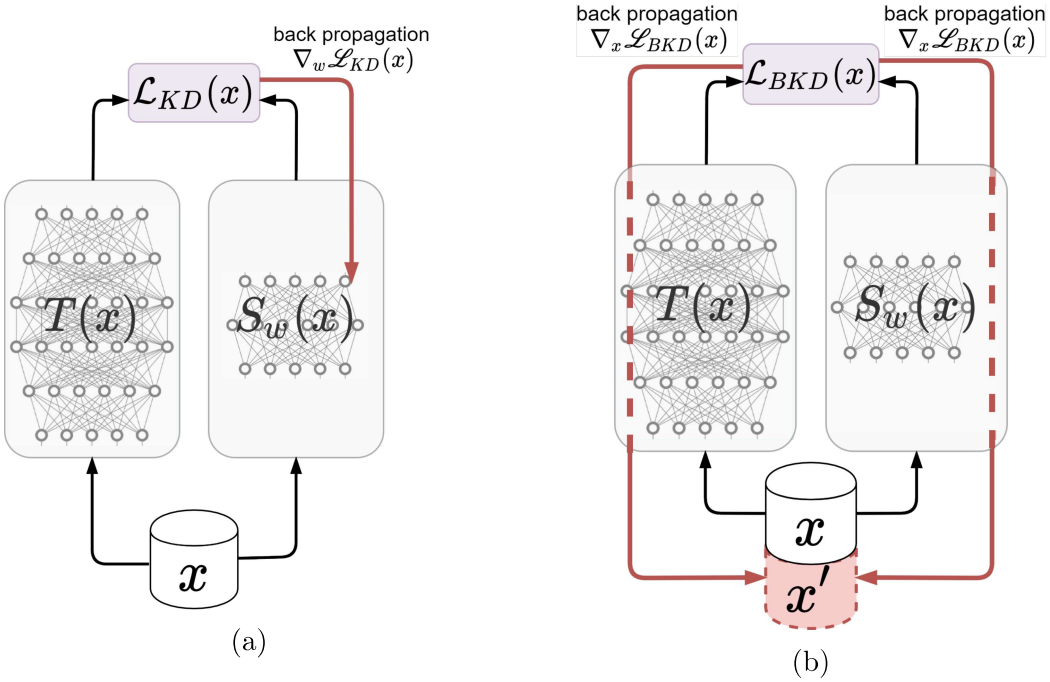


Figure 1.1: (a) Minimization Step: Using the teacher model knowledge for training the student in KD (utilizing forward knowledge) (b) Maximization Step: Augmenting the input dataset x with auxiliary data samples x' which is generated by the back propagation of gradient through both networks (utilizing backward knowledge)

highlight the importance of incorporating backward knowledge of the teacher and student networks in the knowledge distillation process. 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 and also in the few-sample KD scenario. We summarize the main contributions of this research in the following:

- Our technique extracts knowledge from both the forward and backward passes of the teacher and student networks in order to identify the maximum divergence regions between the two functions and generate auxiliary data samples around those regions.

- We provide a solution on how to address the non-differentiability of discrete tokens in NLP applications.
- Our approach is generic and is applicable to any improved KD approach.
- The results of our experiments, show 4% improvement on MNIST with a student network that is 160 times smaller, 1% improvement on the CIFAR-10 dataset with a student that is 9 times smaller, and an average 1.5% improvement on the GLUE benchmark with a distilroBERTa-base student.

Chapter 2

Background and Problem Definition

Model compression is one of the current challenging hot research topics among the machine learning community. In the last couple of years, deep learning-based models have been applied successfully to solve various real-world problems. Most of the successful deep models are large in terms of the number of parameters. They consist of millions and sometimes billions of parameters that require powerful computers with a considerable amount of storage capacity to train and use them. On the other hand, the number of devices with limited computational power like mobile phones and other edge-devices increases every day. Because of the resource constraints on these devices, the applications that use deep learning-based inferences provide their services online and on remote servers most of the time. Moreover, because of the real-time inference and accessibility demand to these services, we need to run them offline on the edge devices. The problem of model compression is related to finding a way to reduce the size of these models and run them on edge devices with minimum loss of accuracy.

It has been shown in the literature that the neural networks are over parameterized [23]. It means, for a particular trained neural network, we must be able to find another network with a smaller number of parameters which has almost similar behaviour and accuracy. This problem has been attacked from different approaches, including low-rank factorization, quantization, pruning and knowledge distillation. Each of these approaches has its pros and cons. The presented work in this research paper is an attempt to improve the current knowledge distillation methods. However, before investigating this method, it would be useful to review some of the other model compression approaches briefly.

2.1 Pruning

One way to reduce the size of a neural network is pruning. The idea behind this approach is removing the redundant and less important weights [13, 23]. Pruning can be done either during or after training a neural network. This approach's methods help to reduce the number of parameters in a neural network, increase the inference speed, prevent overfitting, and improve the generalization of a neural network. Dropout technique [39] is one of the famous pruning techniques used to prevent overfitting. The pruning techniques can be categorized as follows [9]:

- Weight pruning: In these techniques, the less important weights are removed from the network [13].
- Neuron pruning: neurons with redundant outputs remove from the network as well as all of its incoming weights [37].
- Filter pruning. These techniques have been developed to compress convolutional neural networks (CNNs). Filter pruning techniques rank the importance of filters in a CNN model and remove the least important filters [25].
- Layer pruning. These techniques try to remove one or more entire layer from a deep neural network [7].

2.2 Low rank factorization

As we discussed in previous sections, deep models have redundancy in their parameters and feature maps. Generally, this redundancy is reflected in weight matrices of these models and makes them to have a lower rank. By exploiting this feature of weight matrices, the low-rank factorization methods decompose the weight matrices of a deep model and replace them with smaller matrices without any serious effect on models' performance. The most famous and general method used in this area is singular value decomposition (SVD) [9]. This method, decomposes a matrix $A \in \mathbb{R}^{m \times n}$ into three factors $A = U\Sigma V^T$, where $U \in \mathbb{R}^{m \times r}$, $\Sigma \in \mathbb{R}^{r \times r}$, and $V^T \in \mathbb{R}^{r \times n}$. Here r is the rank of the matrix A and U, Σ, and V^T are smaller matrices than A. Low-rank factorization techniques can be applied either on dense layers of feed-forward networks or on filters of CNNs [33]. Previous studies have shown that these techniques can reduce the size of networks and improve the inference time of them up to 30-50% in comparison to original networks [9].

2.3 Knowledge distillation

Knowledge distillation (KD) [16] is another approach to reduce the size of a neural network. In this method, we have a large and a small neural network in terms of the number of parameters called the teacher and student networks. The teacher network has been trained on a specific dataset and usually has a good performance. On the other hand, we want to train the student network, which is usually similar to the teacher network, in order to mimic the teacher's behaviour. More precisely, the student is trained based on both the target labels and the teacher's outputs. The primary advantage of KD is leveraging the acquired dark knowledge of the teacher network to train the student network better.

Recently it has been shown in the literature; the neural network models extract some knowledge about the dataset after training, which does not exist in the original dataset. This knowledge is sometimes called dark knowledge and can guide the smaller models during training to achieve higher performance [16]. The original knowledge distillation method [16] uses the following loss function to train the student network. (Here we consider our dataset is $\{(x_i, y_i)\}_{i=1}^{n}$)

$$\mathcal{L}_{KD} = \alpha \mathcal{L}_1\Big(\sigma\big(S(x)\big), y\Big) + (1-\alpha)\mathcal{L}_2\left(\sigma\left(\frac{S(x)}{\tau}\right), \sigma\left(\frac{T(x)}{\tau}\right)\right) \tag{2.1}$$

Where $S(x)$ and $T(x)$ are student and teacher networks respectively. $\sigma(.)$ is the softmax function. τ is the temperature parameter and α is a coefficient between $[0, 1]$. This loss function is a linear combination of two loss functions. The first loss function $\mathcal{L}_1$ minimizes the difference between the output of the student model and the given true labels. The second loss function $\mathcal{L}_2$ minimizes the difference between the outputs of the student model and the teacher model. Therefore the whole loss function minimizes 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 data samples' underlying function. Figure 2.1 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 eq. 2.1.

2.4 Problem

In the original KD, 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 (see Figure 2.1). In machine learning problems, we always have access to a limited number of data samples from the underlying function in a given dataset. The original KD method uses the teacher function's responses to these data samples as well as the target responses to training the student network. Therefore, we only have information from the areas indicated by light blue circles in Figure 2.1, and we do not consider the behaviour of the underlying and the teacher functions in other areas. A high capacity model like a student network can easily match to the teacher and underlying functions in the areas we have data samples but diverge from them in other areas (Figure 2.1). Since we only have access to a limited number of samples, we can not observe the underlying function's behaviour in other areas. However, the teacher network is a continuous function, and its behaviour is observable everywhere on its domain. If we assume the teacher network as a good approximation of the underlying function, we can leverage its continuity and train the student network better. 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 and generate auxiliary data points in the areas that we have maximum divergences between the student and teacher. In the next sections, we will see the details of this method.

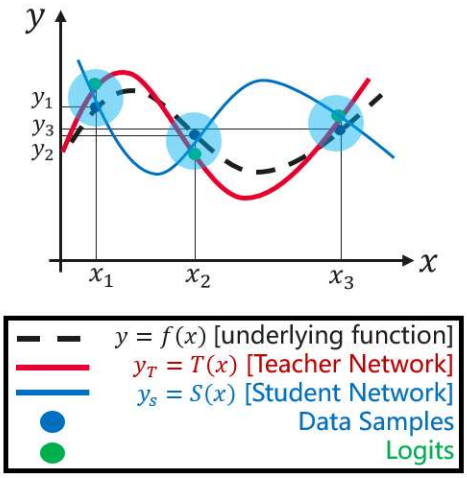


Figure 2.1: Visualizing the data insufficiency issue for the original KD algorithm. It demonstrates the behaviour of the teacher and the student function after training with KD loss.

Chapter 3

Related Works

3.1 Sobolev training for KD

As we mentioned in section 2.4, the KD loss cannot guarantee the student and teacher functions to match over the entire input space. The reason is training two networks based on the original KD loss function would only match their output values on the training samples and not their gradients. There are some works in the literature to address this issue by matching the gradients of the two networks at given training samples during training [10, 38]. However, since we usually deal with networks with multidimensional inputs and outputs, the gradients of output vectors w.r.t input vectors give rise to large Jacobin matrices. Matching these Jacobian matrices is not computationally efficient and is not practical in real-world problems.

Sobolev training [10] proposes a solution to avoid large Jacobian matrices: instead of directly matching the gradients of the two networks, one can match the projection of the gradients onto a random vector v which is sampled uniformly from the unit sphere. Although this approach can reduce the computational complexity of matching gradients during the training, still computing Jacobian matrices before this projection can be very computationally expensive (especially for NLP applications that deal with large vocabulary sizes). To tackle this problem in our work, we define a new scalar loss function based on an l_2 norm to measure the distance between the teacher and student networks (see Figure 4.1-(b)). Gradients of this scalar loss function is a vector with the same size as the input vector x and can be used as a proxy for the network gradients introduced in [10, 38].

3.2 Knowledge distillation with adversarial samples

A good practice for knowledge distillation is transferring the teacher network's decision boundary to the student network in classification problems. One way to do this is by using adversarial samples. Adversarial attack is called to a set of techniques that change a data sample from a particular class into an adversarial sample of another class to confuse a given classifier. Authors of knowledge distillation with adversarial samples method [15] have been inspired by this idea and proposed a KD method to transfer the teacher's decision boundary to the student network. They proposed to perturb the data samples and generate adversarial samples near the decision boundary of the teacher network. These samples are called boundary supporting sample (BSS), and they can be used in the training phase of the student network to transfer the teacher boundary (Figure 3.1).

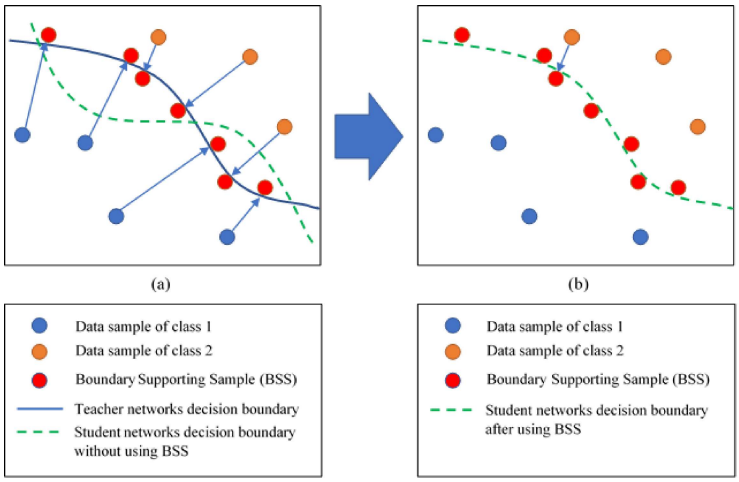


Figure 3.1: Knowledge distillation with adversarial samples [15]. (a) shows the dataset with two classes (blue and orange points). The red points are boundary supported samples (BSS) generated based on the original samples. The blue line is the teacher decision boundary ad the dashed green line is the student decision boundary without using BSS. (b) illustrates the student decision boundary after using BSS.

Although utilizing BSS is a great idea for knowledge distillation, it is only applicable to classification problems. The other problem is that the adversarial attack to a sample point can fail, and a good BSS may find after several failure cases, which is not efficient. In the next section, we propose a novel idea to address these shortcomings.

Chapter 4

Methodology

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 Figure 4.1). 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 [3] : 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:

$$\sup_{\theta} R(\theta, \delta_0) = \inf_{\delta \in C} \sup_{\theta \in \Theta} R(\theta, \delta) \tag{4.1}$$

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.

$$\min_{w} \max_{x} R(T_x, S_{x,w}) \tag{4.2}$$

4.1 Maximization step: Generating auxilary data based on backward-KD loss

In the maximization step of our technique, we define a new 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:

$$\mathcal{L}_{\mathcal{BKD}} = ||S(x) - T(x)||_2^2 \tag{4.3}$$

Here the main idea is that by taking the gradient of $\mathcal{L}_{\mathcal{BKD}}$ loss function in eq. 4.3 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.

$$x' = \max_{x \in \mathcal{X}} ||S(x) - T(x)||_2^2 \tag{4.4}$$

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

$$x^{i+1} = x^i + \eta \, \nabla_x \, ||S(x) - T(x)||_2^2 \tag{4.5}$$

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, in practice we do the perturbation steps for a limited number of iterations to keep the generated auxiliary samples in data distributions. 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.

Figure 2.1 demonstrates our idea using a simple example more clearly. Figure 2.1 shows a trained teacher and student functions given the training samples (x_1,y_1), (x_2,y_2), (x_3,y_3). Figure 4.1-(b) 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 [10]. Figure 4.1-(b) also illustrates an example of generating one auxiliary sample from the training sample x_2. If we apply eq. 4.5 to this sample, after several iterations, we will reach to a new auxiliary data point (x'_2). It is evident in Figure 2.1-(a) that, as expected, there is a large divergence between the teacher and student networks in (x'_2) point.

4.2 Minimization step: Improving KD with generated auxiliary data

We can apply the maximization step to all 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 [28], we have used the following KD loss function in our work:

$$\mathcal{L}_{KD} = (1-\alpha)\ H\Big(\sigma\big(S(x)\big), y\Big) + \tau^2\ \alpha\ KL\bigg(\sigma\Big(\frac{S(x)}{\tau}\Big), \sigma\Big(\frac{T(x)}{\tau}\Big)\bigg) \tag{4.6}$$

where $\sigma(.)$ is the softmax function, $H(.)$ is the cross-entropy loss function, $KL(.)$ is the Kullback Leibler divergence, α 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 Figure 5.1). 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 for h times 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. The next section describes this procedure in more detail.

4.3 Backward KD algorithm

Algorithm 1 explains the details of the proposed method. Our proposed KD function's input variables are the student network $S(.)$, the teacher network $T(.)$, the input dataset X, the number of training epochs e, the number of hyper epochs h, and the number of samples perturbing steps l. This algorithm assumes that the teacher network $T(.)$ has trained and the student network $S(.)$ has not trained yet. Also, we assume X' is the set of 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 (original KD method) 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. In lines 7 to 11, we perturb data samples in X' using the gradient of the loss between teacher and student iteratively in order to generate new auxiliary samples. Then in line 12, we replace X with the union of X and X' sets. In the next iteration of the loop in line 4, 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. Since we perturb the data samples based on the gradient of $\mathcal{L}_{BKD}$ loss function, the input dataset needs to have a continuous domain. However, in cases like natural language processing (NLP) where the domain is discrete, It is impossible to apply this method directly. In the next section, we will see how we can adopt backward KD for such applications.

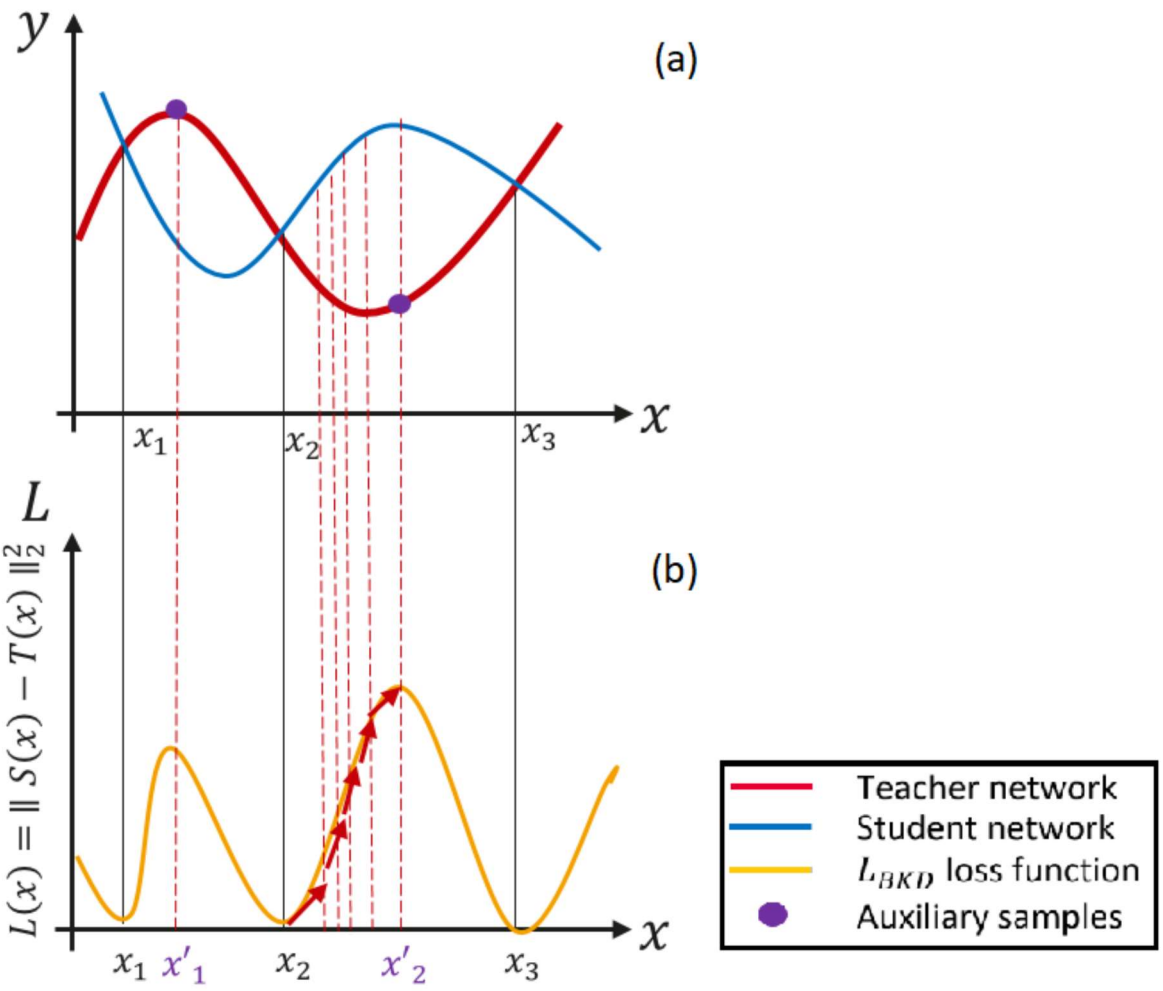


Figure 4.1: (a) divergence areas between the teacher and the student networks. (b) behaviour of l_2-norm loss function between teacher and the student and the idea of obtaining auxiliary data samples.

Algorithm 1

function PROPOSED-KD(S,T,X, e, h, l)
  $X' \leftarrow X$
  for $i = 1$ `to` $h + 1$ **do**
    VANILLA-KD(S,T,X',e)
    $X' \leftarrow X$
    for x' `in` X' **do**
      for $j = 1$ `to` l **do**
        $x' \leftarrow x' + \eta \nabla_x ||S(x') - T(x')||_2^2$
      end for
    end for
    $X' \leftarrow X' \cup X$
  end for
  VANILLA-KD(S,T,X,e)
  return S
end function

4.4 Backward KD for NLP applications

In this section, 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 Figure 4.2). 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 Figure 4.2). 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.

Theorem 4.4.1. *If $W_S \in \mathbb{R}^{d_1 \times |V|}$ be the embedding matrix of the student network and $W_T \in \mathbb{R}^{d_2 \times |V|}$ be the embedding matrix of the teacher network, where $|V|$ is the vocabulary size, d_1 and d_2 are the embedding vector size of the student and the teacher networks respectively. If $x \in \{0,1\}^{|v|}$ be the one-hot vector of a token in a text document and if $z_S = W_S x$ and $z_T = W_T x$ be the student and teacher embedding vectors of x, then there*

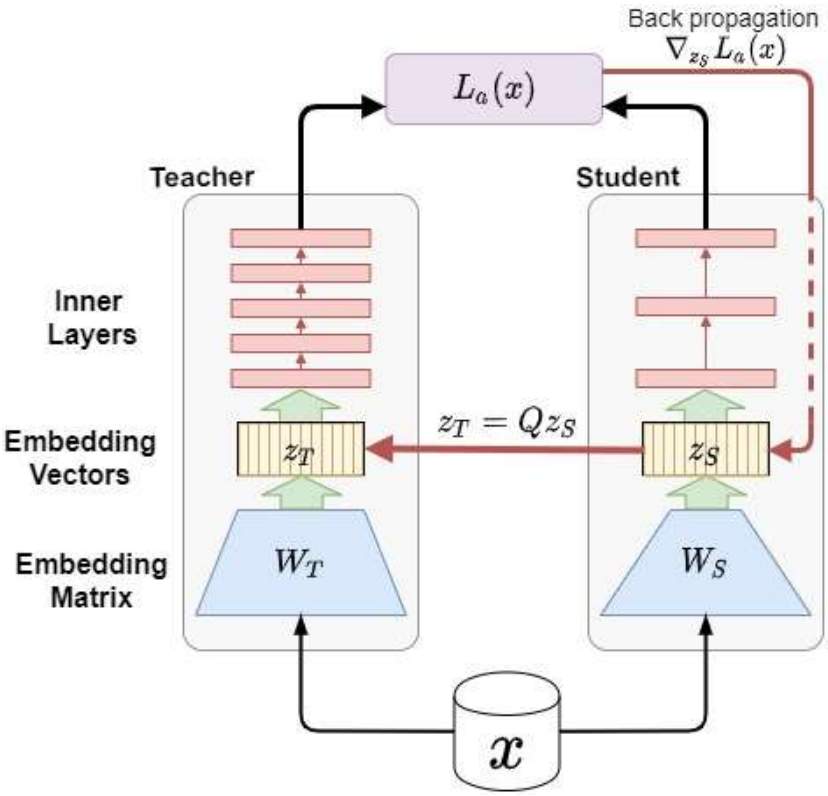


Figure 4.2: General procedure of utilizing auxiliary samples in NLP models. Here x is the one-hot vector of input tokens, W is the embedding matrix, and z is the embedding vector of x.

exists a transform matrix $Q \in \mathbb{R}^{d_2 \times d_1}$ *such that:*

$$z_T = Qz_S \tag{4.7}$$

Proof.

$$z_T = W_T x \tag{4.8}$$

$$z_S = W_S x \tag{4.9}$$

We want to find a transform matrix Q such that:

$$W_T = QW_S \tag{4.10}$$

For this purpose we can solve the following optimization problem by using list square method:

$$\min_{Q} ||W_T - QW_S||^2 \tag{4.11}$$

By solving the above optimization problem using the least squares method, we achieves the following solution for Q:

$$Q = W_T W_s^T (W_s W_s^T)^{-1} \tag{4.12}$$

Now, from Eq. 4.10 we have:

$$W_T = QW_s \tag{4.13}$$

$$W_T x = QW_s x \tag{4.14}$$

$$z_T = Qz_s \tag{4.15}$$

□

In equation 4.12, term $W_S^T(W_S W_S^T)^{-1}$ is the pseudo inverse of W_S embedding matrix. 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 4.16 and 4.12, we can re-construct z_T during the steps of data perturbation as following.

$$\begin{aligned} z_S^{i+1} &= z_S^i + \eta \nabla_{zS} \mathcal{L}_{BKD} \\ z_T^{i+1} &= W_T W_S^T (W_S W_S^T)^{-1} z_S^{i+1} \end{aligned} \tag{4.16}$$

We will see the details of the modified algorithm of backwad KD for NLP tasks in next section.

4.5 Algorithm of backward KD for NLP tasks

Algorithm 2 explains the details of applying backward KD in NLP tasks. This algorithm is almost similar to algorithm 1. However, 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). In line 15, we use the transform method proposed in section 4.4 to transform student's perturbed embedding vectors into teacher's embedding vectors.

Algorithm 2

function PROPOSED-KD(S,T,X, e, h, l)
  $W_T \leftarrow$ EMBEDDING-MATRIX(T)
  $W_S \leftarrow$ EMBEDDING-MATRIX(S)
  $Z_T \leftarrow W_T X$
  $Z_S \leftarrow W_S X$
  $Z'_T \leftarrow Z_T$
  $Z'_S \leftarrow Z_S$
  for $i = 1$ `to` $h + 1$ **do**
    VANILLA-KD(S,T,Z'_T, Z'_S,e)
    $Z'_T \leftarrow Z_T$
    $Z'_S \leftarrow Z_S$
    for (z'_S, z'_T) `in` (Z'_S, Z'_T) **do**
      for $j = 1$ `to` l **do**
        $z'_S \leftarrow z'_S + \eta \nabla_{z_S} ||S(z'_S) - T(z'_S)||_2^2$
        $z'_T \leftarrow W_T W_S (W_S W_S^T)^{-1} z'_S$
      end for
    end for
    $Z'_S \leftarrow Z'_S \cup Z_S$
    $Z'_T \leftarrow Z'_T \cup Z_T$
  end for
  VANILLA-KD(S,T,Z_T, Z_S,e)
  return S
end function

Chapter 5

Experiments and Results

Five experiments were designed to evaluate the proposed method. [1] The first experiment is designed based on synthetic data to visualize the idea behind this technique. The second and third ones are on the image classification tasks, and the last two are in NLP. We followed the general procedure explained in the algorithms of sections 4.3 and 4.5 for all of these experiments. Also, the method explained in section 4.4 was applied for NLP experiments. The next sections explain the details of these experiments.

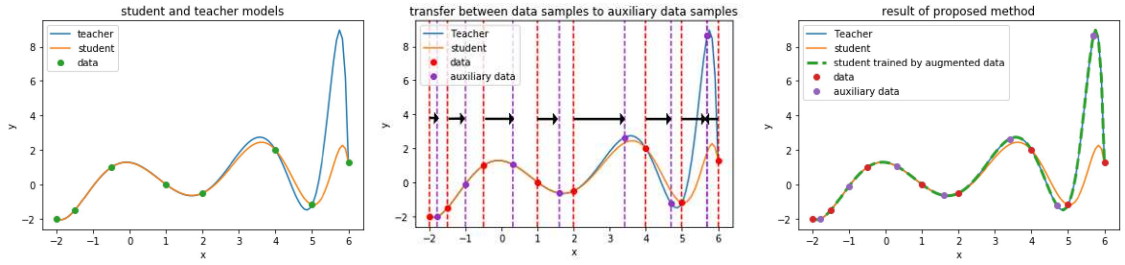
5.1 Datasets

For image classification, we assess backward KD on MNIST [22] and CIFAR-10[20] datasets. MNIST consists of 28×28 monochrome handwritten digits images with 10 classes. CIFAR-10 contains 32×32 color images of 10 different classes. For the natural language inference task, we employ the General Language Understanding Evaluation (GLUE) benchmark [44], which is a collection of nine different tasks for training, evaluating, and analyzing natural language understanding models. GLUE consists of Multi-Genre Natural Language Inference (MNLI) [47], Quora Question Pairs (QQP) [8], Question Natural Language Inference (QNLI) [32], Stanford Sentiment Treebank (SST-2) [36], Corpus of Linguistic Acceptability (COLA) [46], Semantic Textual Similarity Benchmark (STS-B) [6], Microsoft Research Paraphrase Corpus (MRPC) [12], Recognizing Textual Entailment (RTE) [1], Winograd NLI (WNLI) [24].

[1] We used PyTorch (https://pytorch.org/) framework [30] for implementing all experiments and Huggingface (https://huggingface.co/) framework [48] in the implementations of NLP experiments.

5.2 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 Figure 5.1-(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 Figure 5.1-(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 Figure 5.1-(c).



(a) (b) (c)

Figure 5.1: Visualizing the generation of auxiliary samples and their utilization in training the student model.

5.3 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 feed forward 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 5.1, the student network which is trained by using the proposed method achieves much better results in comparison with two other trained networks.

Table 5.1: 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**

5.4 CIFAR-10 classification

The second experiment is conducted on the CIFAR-10 dataset with two popular network structures as the teacher and the student networks. In this experiment, we used the inception v3 [42] network as the teacher and mobileNet v2 [34] as the student. The teacher is approximately 9 times bigger than the student. We repeated the previous experiment on CIFAR-10 by using these two networks. Table 5.2 shows the results of this experiment.

Table 5.2: Results of experiment on CIFAR-10 dataset

Model	method	#parameters	accuracy on test set
inception v3 (teacher)	from scratch	21638954	95.41%
mobilenet (student)	from scratch	2236682	91.17%
mobilenet (student)	original KD	2236682	91.74%
mobilenet (student)	**proposed method**	2236682	**92.60%**

5.5 GLUE Tasks

The third experiment is designed based on General Language Understanding Evaluation (GLUE) benchmark [44] and roBERTa family language models [26, 35]. roBERTa models

(roBERTa-large, roBERTa-base, and distilroBERTa) are BERT [11] based language understanding pre-trained models where roBERTa-large and roBERTa-base are the cumbersome versions which are proposed in [26] and have 24 and 12 transformer layers respectively. distilroBERTa is the compressed version of these models with 6 transformer layers and has been trained based on KD procedure proposed in [35] with utilizing the roBERTa-base as the teacher. The general procedure in GLUE tasks is fine-tuning the pre-trained models for its down-stream tasks and the average performance score. We fine-tuned the distilroBERTa model based on the proposed method by utilizing the fine-tuned roBERTa-large teacher for each of these tasks. As it is shown in table 5.3, the proposed method could improve the distilroBERTa performance on most of these tasks.

Table 5.3: Results of 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)	56.61	**92.77**	84.06	87.28	90.8	84.14	**91.36**	65.70	56.33	78.78
Our DistilroBERTa (Student)	**60.49**	92.51	**87.25**	**87.56**	**91.21**	**85.1**	91.19	**71.11**	56.33	**80.30**

5.6 GLUE tasks with few sample points

In this experiment, we modified the previous experiment 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 are used for fine-tuning the student model. Table 5.4 shows the results of this experiment.

Table 5.4: 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**

Chapter 6

Conclusion

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 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

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

[2] 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.

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

[4] 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.

[5] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[6] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

[7] Shi Chen and Qi Zhao. Shallowing deep networks: Layer-wise pruning based on feature representations. *IEEE transactions on pattern analysis and machine intelligence*, 41(12):3048–3056, 2018.

[8] Zihan Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. Quora question pairs, 2018.

[9] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, pages 1–43, 2020.

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

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[13] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[14] 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.

[15] 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.

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

[17] 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.

[18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[19] 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.

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] 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.

[22] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[23] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2:598–605, 1989.

[24] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. Citeseer, 2012.

[25] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[26] 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.

[27] Paul Micaelli and Amos J Storkey. Zero-shot knowledge transfer via adversarial belief matching. In *Advances in Neural Information Processing Systems*, pages 9547–9557, 2019.

[28] 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.

[29] Gaurav Kumar Nayak, Konda Reddy Mopuri, Vaisakh Shaj, R Venkatesh Babu, and Anirban Chakraborty. Zero-shot knowledge distillation in deep networks. *arXiv preprint arXiv:1905.08114*, 2019.

[30] 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.

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

[32] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[33] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[35] 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.

[36] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[37] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

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

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

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

[41] 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.

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

[43] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Tensor decomposition for compressing recurrent neural network. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[44] 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.

[45] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. Pruning from scratch. *arXiv preprint arXiv:1909.12579*, 2019.

[46] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[47] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[48] 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.

[49] 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.