Zero-Shot Knowledge Transfer via Adversarial Belief Matching

DD2412 Deep Learning, Advanced Course

KTH Royal Institute of Technology

Deepika Anantha Padmanaban
deap@kth.se

Polyxeni Parthena Ioannidou
ppio@kth.se

November 1, 2019

Abstract

Following the work of Micaelli and Storkley [1], we re-implemented zero-shot knowledge transfer from a large teacher's network to a student's network using adversarial belief matching. As proposed in the paper [1] we pre-trained the teacher with CIFAR-10 dataset and without using any original data, we train student's model to predict same results as teacher's model. This has been achieved by using an adversarial generator architecture, which tries generating samples from random noise for which student cannot predict same results as teacher, and then we use them to train student's network. By using this data which is very hard to predict for the student, we have an improvement in model's predictions. Furthermore, we have used a WideResNet for student and teacher architectures, whereas generator is a CNN architecture as mentioned by the author. In the paper [1] it has been proposed that the objective for generator is to maximize the KL divergence between the teacher and student models' predicted output, whereas for student the objective function is to minimize the KL Divergence between the student and the teacher model with an additional attention term based on a subset of layer for each student-teacher pair. We should mention that due to the fact that teacher is a deeper network, in the attention term we use the activation from a subset of layers of teacher and student models,i.e. only those at the end of each network block.

1 Introduction

Training on an extremely large dataset is a complex task which requires a large amount computational power and time. This means that large neural networks add some limitations to users, who do not have the appropriate hardware and resources, as training takes a lot of time. In addition to this, as the number of hyper-parameters is quite big, searching for best model's structure is a real challenge. Knowledge transfer is a technique that can handle these difficulties as has been mentioned in the previous works such as Knowledge Distillation and Attention Transfer.

There are a couple of reasons why knowledge transfer has become an important subject of study within the field of deep learning. First of all, knowledge transfer enables us to train a new network even though the original data and labels might be unavailable, due to privacy concerns, proprietary issues, sheer size of the dataset or other factors. Knowledge transfer also facilitates knowledge distillation, that is, by training a smaller network than that of the original network it is possible to produce a leaner and faster model of comparable performance. This is advantageous as it allows for the implementation of highly efficient and accurate deep learning models on platforms where computational resources might be severely limited, such as on mobile cellphone devices.

Even though knowledge transfer is possible through the use of available training data, an arguably more efficient way is through the use of adversarial pseudo-data. In this type of learning scheme a generator is trained to generate data which maximizes the difference in output between the student and teacher, allowing for faster training than when using regular dataset in addition to relieving the concerns of needing the original data. This is the essence of zero-shot knowledge transfer, where none of the original data is used in teaching the student model, but the pre-trained teacher network's output is used as the true label while training. The training is called adversarial belief matching because, the generator model is trained to produce samples that have high chances of being classified different by the student and the teacher and hence has the objective to maximize the KL divergence between the student and the teacher model, while the student model that is trained to extract the knowledge of the teacher model has the objective of minimizing the same KL divergence.

2 The Model

The model presented in Micaelli and Storkley's paper [1], henceforth referred to as "the paper", consists of three principal components: A pre-trained teacher network T, a generator network G and the student network S. The paper presents a number of experiments, each experiment consisting of a teacher-student pair where the larger teacher network is paired with a smaller student network, which has the effect that knowledge is distilled when the student network is trained. The experiment was performed once for training a 40-2 WRN teacher and transferring knowledge to a 16-2 WRN student on the CIFAR-10 dataset.

2.1 The Teacher Network

The teacher network T acts as a pre-trained network that has been trained on the entire dataset in order to match an input image $\boldsymbol{x}$ to a probability vector $\boldsymbol{t}$. The structure of T is a WideResNet with a depth of 16 or 40 layers and a widening factor of 1 and 2, the resulting number of parameters varying from 0.6 million to 2.2 million parameters. Two datasets are used to train T: CIFAR-10, consisting of 50 000 spear across 10 different classes.

Teacher Networks (# parameters)	$\rightarrow$	Student Networks (# parameters)
WRN-40-2 (2.2M)	$\rightarrow$	WRN-16-1 (0.2M)
WRN-40-2 (2.2M)	$\rightarrow$	WRN-16-2 (0.7M)

Table 1: The sample structure of the teacher-student pairs used in the paper, with the accompanying number of parameters.

2.2 The Generator Network

The generator network G is a basic convolutional neural network (CNN) that uses batch normalization as well as Leaky ReLU for activation. This network is trained to generate pseudo-examples $\boldsymbol{x_p}$ from a zero-mean Gaussian noise vector of variance $\boldsymbol{I}$ which are passed to student network for training. G is trained in order to maximize the Kullback-Liebler (KL) divergence between the output probability vectors of T and S. There is no attention term to the generator Loss function, a we don't want to make it extremely hard for the student to learn, thus avoiding the possibility of model divergence.

$$\mathcal{L}_g = -D_{KL}(T(\boldsymbol{x_p})||S(\boldsymbol{x_p})) \tag{1}$$

This has the effect that the generator is trained to generate adversarial pseudo-examples for the student, i.e. the generated examples will tend not to be easily classified, potentially improving the learning and not allowing the network's learning to get stuck in any local minima that correctly classifies simple inputs but performs poorly on difficult inputs.

2.3 The Student Network

The student network S structurally consists of WideResNets with a depth of 16 and a widening factor or 1 and 2 in most of the experiments presented in the paper. The number of parameters varies from 0.2 million to 0.7 million, significantly smaller than the teacher network. The student network takes the pseudo-examples generated by the generator and matches it to an output probability vector $\boldsymbol{s}$. The aim of the student is to learn the output probability matchings of the teacher network, which is achieved by using a loss function that minimizes the KL-divergence between itself and the teacher:

$$\mathcal{L}_s = D_{KL}(T(\boldsymbol{x_p})||S(\boldsymbol{x_p})) + \beta \sum_{l}^{N_l} \left\| \frac{\boldsymbol{f}(A_t^{(t)})}{\left\|\boldsymbol{f}(A_t^{(t)})\right\|_2} - \frac{\boldsymbol{f}(A_t^{(s)})}{\left\|\boldsymbol{f}(A_t^{(s)})\right\|_2} \right\|_2 \tag{2}$$

where $A_t^{(t)}$ and $A_t^{(s)}$ are the activations in the layers of the teacher and student networks. The right side of the loss function $\mathcal{L}_s$ is the *attention term.* This term has the effect that student trains to minimize the difference in attention between itself and the teacher, i.e. to focus on the same features as the teacher network when learning.

The model and learning scheme presented here depends on a balance between the ability of the generator to produce training examples for which the teacher's and student's predictions significantly differs, and the student's ability to learn and adapt to the adversarial examples passed from the generator. To this end, the most important parameter which can be tuned in order to achieve this balance is the number of gradient steps taken by the generator and student during each epoch. The paper proposes that an effective balance occurs when the generator is allowed one gradient step in each epoch and the student is allowed to take 10 gradient steps. This balance is also the reason for not using an attention term when training the generator, as the generated pseudo-examples are too good at fooling the student, resulting in poor student performance.

3 Reimplementation and Technical Details

3.1 Technical description of the architecture

First, the complete models including the teacher, student and generator networks were reimplemented in TensorFlow 2.0.0b using python 3. More specifically, for teacher's and student's network we used a WideResNet architecture, that is a residual network with convolutional 2d layers, a widening factor k and a deepening factor l. The WRN-40-2 model was used to train the teacher, whereas the WRN-16-2 was used for student. We also used *he_norm* as kernel initializer, uniform as gamma initializer, zero for dropout and 0.1 for momentum. According to the activations, we use the ReLU function, as well as the number of channels in the input is taking different values from 16 in the first block, 32 in the second and 64 in the third one. Moreover, in Wide-ResNet all layers are connected with each other in a way that the feature maps of all previous layers is being passed into the current layer and then the output is being passed to the subsequent layers. For this reimplementation in tensorflow we also checked the number of parameters in WideResNet, which was around ~ 2.2 millions. In this point we should mention that for teacher's network all these parameters were set to non-trainable, and we loaded a pre-trained model that we had trained on cifar-10 for our implementation.

Generator is a Convolutional Neural Network of three layers as is mentioned in the paper. We also use batch normalization to ensure that the input is normalized over the three channels, LeakyReLU is being added after convolutional layers and ReLU is being used in the fully connected layer as activation. The generator model is essentially used for creating inputs for training the student model. The input is generated from a random noise of dimension 100 and then is reshaped into $(32, 32, 3)$, the shape of the input as expected by the wideresnet model. The generator is a trainable model in the network. It is trained with an objective of maximizing the KL Divergence between the teacher and student outputs, thus producing samples that are not-so-easy for the student model to classify, promoting learning and thus, generating samples that tend to represent the original dataset.

As regards teacher's training, we trained the WideResNet model and we saved the weights so as to use them for student's learning. Concerning now the zero-shot knowledge transfer, we implemented the algorithm proposed in the paper, where for N total iterations we were taking fixed steps in both Generator and Student so as to minimize their losses by updating their gradients. More specifically, student takes 10 steps, whereas generator takes 1 step. It should be mentioned also, that student takes the same total iterations as the teacher, which means that if teacher is taking 80000 total iterations, the student should take only 8000 iterations with 10 steps in each.

3.2 Training Details

Since, the network was adversarial, we had to custom train both the generator and the student network to compute and apply gradients on the trainable variables. We used the KLD inbuilt loss function and Gradient Tape for gradient calculation over the loss and apply_gradients() for taking a step over the optimizer.

3.3 Parameters specification

This section is devoted to the detail of the implementation that was presented in the paper. We tried following the same parameterization as presented in the paper and produce an account of the same in this section.

1. For the dataset we used, CIFAR10, the official split of 5000 images per class for training and 1000 images per class for testing was followed.
2. All the models in the network are trained on a batch size of 128.
3. The teacher model was trained using an SGD optimizer with a maximum learning rate of 0.1. The learning rate was reduced by a factor of 5 after 30%, 60% and 80% of the training. The model was trained over 200 epochs.
4. The learning rate of the student and generator models were set to be $2e^{-3}$ with cosine annealing for adjusting the learning rates. The optimizer used in this case was Adam.
5. The total steps taken by the student was set to match the teacher iterations. Since, the teacher had 200 epoch over 390 batches of training data, the student was set to take 80000 iterations in total.
6. Since, we want the algorithm to converge, we ensure that the student is given more time to learn, thus, for every step of a generator, the student is allowed to take 10 steps.i.e. $n_s = 10$ and $n_g = 1$.
7. The generator is input with a random noise of dimension 100.

4 Experimentation and Results

Since, the original code was in PyTorch, we developed our code in the TensorFlow framework.

4.1 Teacher Training:

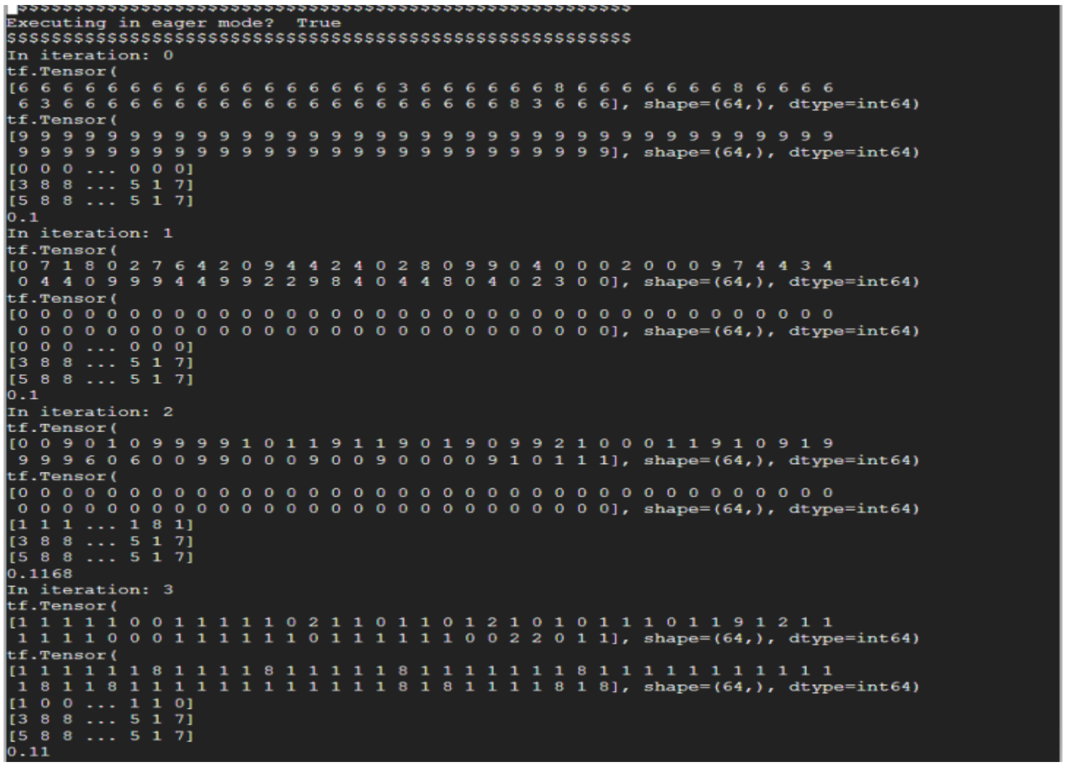
Since, the experiments in the paper started with pre-trained teacher model, we developed a code to train a WideResNet on the CIFAR-10 dataset from the scratch.We had initially planned experiments on various combinations of the teacher network and thus tabulate the results of the various batchsizes used while training the teacher, table 2. We could not get the state-of-art resuts for the teacher as we did not try any hyper-parameter tuning of the model.

Teacher Networks	Batch Size	Test Accuracy
WRN-40-2 (2.2M)	128	0.84
WRN-40-2 (2.2M)	64	0.86
WRN-16-2 (0.7M)	128	0.80
WRN-16-2 (0.7M)	64	0.84

Table 2: The structure of the teacher-student pairs used in the paper, with the accompanying number of parameters.

4.2 Sample Input Generation from Generator:

As said previously, we use the generator to generate samples from some random noise. The CNN model converts this sample to a input of batch normalized input of shape$(32, 32, 3)$.
The fig.[1] shows that the samples that are generated are not the same and are different from each other from the beginning of training. With further training, we see these to be diverging into various classes.



```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Executing in eager mode?  True
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
In iteration: 0
tf.Tensor(
[6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 3 6 6 6 6 6 6 8 6 6 6 6 6 6 6 8 6 6 6 6
 6 3 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 8 3 6 6 6], shape=(64,), dtype=int64)
tf.Tensor(
[9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9], shape=(64,), dtype=int64)
[0 0 0 ... 0 0 0]
[3 8 8 ... 5 1 7]
[5 8 8 ... 5 1 7]
0.1
In iteration: 1
tf.Tensor(
[0 7 1 8 0 2 7 6 4 2 0 9 4 4 2 4 0 2 8 0 9 9 0 4 0 0 0 2 0 0 0 9 7 4 4 3 4
 0 4 4 0 9 9 9 4 4 9 9 2 2 9 8 4 0 4 4 8 0 4 0 2 3 0 0], shape=(64,), dtype=int64)
tf.Tensor(
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0], shape=(64,), dtype=int64)
[0 0 0 ... 0 0 0]
[3 8 8 ... 5 1 7]
[5 8 8 ... 5 1 7]
0.1
In iteration: 2
tf.Tensor(
[0 0 9 0 1 0 9 9 9 9 1 0 1 1 9 1 1 9 0 1 9 0 9 9 2 1 0 0 0 1 1 9 1 0 9 1 9
 9 9 9 6 0 6 0 0 9 9 0 0 0 9 0 0 9 0 0 0 0 9 1 0 1 1 1], shape=(64,), dtype=int64)
tf.Tensor(
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0], shape=(64,), dtype=int64)
[1 1 1 ... 1 8 1]
[3 8 8 ... 5 1 7]
[5 8 8 ... 5 1 7]
0.1168
In iteration: 3
tf.Tensor(
[1 1 1 1 1 0 0 1 1 1 1 1 0 2 1 1 0 1 1 0 1 2 1 0 1 0 1 1 1 0 1 1 9 1 2 1 1
 1 1 1 1 0 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 2 2 0 1 1], shape=(64,), dtype=int64)
tf.Tensor(
[1 1 1 1 1 1 8 1 1 1 1 8 1 1 1 1 1 8 1 1 1 1 1 1 1 8 1 1 1 1 1 1 1 1 1 1 1
 1 8 1 1 8 1 1 1 1 1 1 1 1 1 1 1 1 8 1 8 1 1 1 1 1 8 1 8], shape=(64,), dtype=int64)
[1 0 0 ... 1 1 0]
[3 8 8 ... 5 1 7]
[5 8 8 ... 5 1 7]
0.11
```

Figure 1: The debug output for the first few iterations. For each iteration, the first tensor is the teacher predicted output for a random noise, the second tensor is the student predicted output for a random noise, the following lists are student predicted output, true label, teacher predicted output on cifar-10 test data respectively, followed by a float value of the model test accuracy.

4.3 Student Training:

The student network was trained using the output of the generator as the input and the output of the teacher for the same input was used as the label while training. The student was trained for over 8,000 iterations as in each iteration the student was taking 10 steps summing upto 80k steps as taken in the teacher.

Unfortunately, despite training with the parameters as mentioned in the paper, we could not have an improvement in our test accuracy over the iterations as the accuracy remained clamped around 0.1, which was the accuracy when the model was initialized, meaning the model could not learn any significant information from the teacher network. In the following section, we describe the debug steps we undertook to give a clarity on the problem and the working of each block in the entire model.

4.4 Debug Steps Undertaken

We had issues in training the student model with all the parameters set to the prescribed values. The following are the steps that we took for making sure that all the individual module in the code work fine, after having a discussion with one of the TAs on the issue we had;

1. **Same Student Output for the all the test/train inputs:** One of the main problem with the student model is that it seems to produce the same output for the most of the test/train inputs provided. We suspected that this could be because the sample that is generated from the generator is not stochastic. So, we tried predicting the outputs of the pre-trained teacher model on the same input and it seemed to assign different classes to the same batch of train samples generated, thus, eliminating that possibility.
   The other possibility was the bias overpowering the weights of the model. We tried to make sure the weight initialization of the model are good. We tried multiple initialization and they did not seem to improve the situation. We referred to the code of the author and provided the same set of initialization for the parameter of our model, namely the 'he_normal' initialization for weights and all ones initialization for the bias, which is the standard practice in Deep Learning.

2. **Ensuring Non-Zero Gradients:** Since, the test accuracy did not improve over the iterations, we wanted to make sure that the gradients that were generated were not infinitesimally small. We printed the gradients of both generator and student models and ensured that they were significant values.

3. **Effect of Learning Rate:** The next thing we experimented with was the learning rate of the models. Since, we had the same learning rate for both the models, we decreased the learning rate of the generator, just to ensure that the generator was not making it too hard for the student to learn.

4. **Ensuring Loss Updation:** According to the adversarial nature of the network, we tried to ensure that the KL divergence loss between the student and teacher was updating as expected. We expected the loss to reduce after each student gradient update step and the loss to increase after each generator update step. The updation seemed to work fine as the values agreed.

5. **Movement of Student towards Teacher:** With each update of the student we wanted the output of the student to move closer to the output of the teacher for the random input. We tried printing the predicted result after each update step. Since, the random sample generated was different each time and the generator was updated only once for 10 steps over the student, the student was trying to copy the outputs of the teacher, but the issue was it had almost the same output for all the images in the train batch.fig.1

6. **Proper loading of Pretrained weights:** We initially had this issue of pretrained model being trainable despite having frozen the model, due to default eager execution in TF2.0, but we fixed it by loading only the weights of the model and freezing the entire model.

4.5 Possible Issues

From the above mentioned debugging scenarios,after confirming that each block performs its expected operation, we concluded on the following plausible problem with the network. Going by the flow of the model, the generated sample that is fed to the WideResNet model is essentially random. The WideResNet architecture was similar to the framework that was used to train the teacher, which worked perfectly fine.For every iteration- one step of the generator and 10 steps of the student, the resulting predictions of the model was behaving as expected, as the student was trying to learn from the teacher.But since most of the outputs of the student went into the same class, it forced the generator gradually to take steps, that started pulling the predictions of the teacher towards the student, as the student takes multiple steps in an iteration. This further leads

to the teacher and student becoming more similar, but without the student learning much from the teacher. The output of the student changes over iterations depending on the teacher outputs, giving cues that it is trying to learn according to the teacher, but without any improvement in the accuracy, fig.1.
The main reason we think this happens is due to poor hyper-parameter setting, which is making it really easy for the student to get stuck in the local minima at various regions, rather than exploring the parameter space. This can be attributed to the fact that there is an imbalance between improving the complexity of the sample data generated and the student learning over this data. In other words, the generated samples are too simple that the student is not getting an opportunity to learn. This can be fixed by finding the correct number of steps taken by the student and generator in each iteration and setting a suitable learning rate. The current test accuracy using the parameter settings mentioned in the paper is **0.1**, which is just random.
We couldn't get enough time to search through the hyper-parameters and are still trying to find a better hyper-parameter setting/solution that can get the student to train better. We will update the report if we get to see any improvement in the results. The link to our GitHub has been provided in a separate section below.

5 Group Issues

In this section, we wanted to bring to your notice that we started as a group of 3 members but we had some issues with the third member of the group (Jonatan Lindgren, jonatlin@kth.se), as he did not contribute to our project at all, as you can see in our commits from our Github repository. We do not know if he has dropped the course but we should mention that we had only two meetings with him and after that he did not contribute to the project. We have also dropped an email to the Professor describing the situation.

6 Github repository

The code has been uploaded in the following link:
`https://github.com/deepikaaap/Zero-Shot-Knowledge-Transfer-via-Adversarial-Belief-Matching`

References

[1] Micaelli, Paul and Storkey, Amos, "Zero-shot Knowledge Transfer via Adversarial Belief Matching",https://arxiv.org/pdf/1905.09768.pdf

[2] Larochelle, H., Erhan, D., and Bengio, Y. (2008). Zero-data learning of new tasks. In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008, pages 646–651.

[3] Zagoruyko, S. and Komodakis, N. (2016b). Wide residual networks. CoRR, abs/1605.07146

[4] D Wang, Y Li, Y Lin, Y Zhuang, "Relational Knowledge Transfer for Zero-Shot Learning", https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11802/11854

[5] W Wang, Y Pu, V K Verma, K Fan, Y Zhang,C Chen, P Rai, L Carin, "Zero-Shot Learning via Class-Conditioned Deep Generative Models", https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16087/16709