
Unsupervised Data Augmentation

Overview

Unsupervised Data Augmentation or UDA is a semi-supervised learning method which achieves state-of-the-art results on a wide variety of language and vision tasks.

With only 20 labeled examples, UDA outperforms the previous state-of-the-art on IMDB trained on 25,000 labeled examples.

Model	Number of labeled examples	Error rate
Mixed VAT (Prev. SOTA)	25,000	4.32
BERT	25,000	4.51
UDA	20	4.20

It reduces more than 30% of the error rate of state-of-the-art methods on CIFAR-10 with 4,000 labeled examples and SVHN with 1,000 labeled examples:

Model	CIFAR-10	SVHN
ICT (Prev. SOTA)	7.66 \pm .17	3.53 \pm .07
UDA	4.31\pm.08	2.28\pm.10

It leads to significant improvements on ImageNet with 10% labeled data.

Model	top-1 accuracy	top-5 accuracy
ResNet-50	55.09	77.26
UDA	68.78	88.80

How it works

UDA is a method of *semi-supervised learning*, that reduces the need for labeled examples and better utilizes unlabeled ones.

What we are releasing

We are releasing the following:

- Code for text classifications based on BERT.
- Code for image classifications on CIFAR-10 and SVHN.
- Code and checkpoints for our back translation augmentation system.

All of the code in this repository works out-of-the-box with GPU and Google Cloud TPU.

Requirements

The code is tested on Python 2.7 and Tensorflow 1.13. After installing Tensorflow, run the following command to install dependencies:

```
1 pip install --user absl-py
```

Image classification

Preprocessing

We generate 100 augmented examples for every original example. To download all the augmented data, go to the *image* directory and run

```
1 AUG_COPY=100  
2 bash scripts/download_cifar10.sh ${AUG_COPY}
```

Note that you need 120G disk space for all the augmented data. To save space, you can set AUG_COPY to a smaller number such as 30.

Alternatively, you can generate the augmented examples yourself by running

```
1 AUG_COPY=100  
2 bash scripts/preprocess.sh --aug_copy=${AUG_COPY}
```

CIFAR-10 with 250, 500, 1000, 2000, 4000 examples on GPUs

GPU command:

```
1 # UDA accuracy:  
2 # 4000: 95.68 +- 0.08  
3 # 2000: 95.27 +- 0.14
```

```
4 # 1000: 95.25 +- 0.10
5 # 500: 95.20 +- 0.09
6 # 250: 94.57 +- 0.96
7 bash scripts/run_cifar10_gpu.sh --aug_copy=${AUG_COPY}
```

SVHN with 250, 500, 1000, 2000, 4000 examples on GPUs

```
1 # UDA accuracy:
2 # 4000: 97.72 +- 0.10
3 # 2000: 97.80 +- 0.06
4 # 1000: 97.77 +- 0.07
5 # 500: 97.73 +- 0.09
6 # 250: 97.28 +- 0.40
7
8 bash scripts/run_svhn_gpu.sh --aug_copy=${AUG_COPY}
```

Text classification

Run on GPUs

Memory issues The movie review texts in IMDb are longer than many classification tasks so using a longer sequence length leads to better performances. The sequence lengths are limited by the TPU/GPU memory when using BERT (See the Out-of-memory issues of BERT). As such, we provide scripts to run with shorter sequence lengths and smaller batch sizes.

Instructions If you want to run UDA with BERT base on a GPU with 11 GB memory, go to the *text* directory and run the following commands:

```
1 # Set a larger max_seq_length if your GPU has a memory larger than 11GB
2 MAX_SEQ_LENGTH=128
3
4 # Download data and pretrained BERT checkpoints
5 bash scripts/download.sh
6
7 # Preprocessing
8 bash scripts/prepro.sh --max_seq_length=${MAX_SEQ_LENGTH}
9
10 # Baseline accuracy: around 68%
11 bash scripts/run_base.sh --max_seq_length=${MAX_SEQ_LENGTH}
12
13 # UDA accuracy: around 90%
14 # Set a larger train_batch_size to achieve better performance if your
    GPU has a larger memory.
```

```
15 bash scripts/run_base_uda.sh --train_batch_size=8 --max_seq_length=${  
    MAX_SEQ_LENGTH}
```

Run on Cloud TPU v3-32 Pod to achieve SOTA performance

The best performance in the paper is achieved by using a `max_seq_length` of 512 and initializing with BERT large finetuned on in-domain unsupervised data. If you have access to Google Cloud TPU v3-32 Pod, try:

```
1 MAX_SEQ_LENGTH=512  
2  
3 # Download data and pretrained BERT checkpoints  
4 bash scripts/download.sh  
5  
6 # Preprocessing  
7 bash scripts/prepro.sh --max_seq_length=${MAX_SEQ_LENGTH}  
8  
9 # UDA accuracy: 95.3% - 95.9%  
10 bash train_large_ft_uda_tpu.sh
```

Run back translation data augmentation for your dataset

First of all, install the following dependencies:

```
1 pip install --user nltk  
2 python -c "import nltk; nltk.download('punkt')"  
3 pip install --user tensor2tensor==1.13.4
```

The following command translates the provided example file. It automatically splits paragraphs into sentences, translates English sentences to French and then translates them back into English. Finally, it composes the paraphrased sentences into paragraphs. Go to the *back_translate* directory and run:

```
1 bash download.sh  
2 bash run.sh
```

Guidelines for hyperparameters:

There is a variable *sampling_temp* in the bash file. It is used to control the diversity and quality of the paraphrases. Increasing *sampling_temp* will lead to increased diversity but worse quality. Surprisingly, diversity is more important than quality for many tasks we tried.

We suggest trying to set `sampling_temp` to 0.7, 0.8 and 0.9. If your task is very robust to noise, `sampling_temp=0.9` or 0.8 should lead to improved performance. If your task is not robust to noise, setting `sampling_temp` to 0.7 or 0.6 should be better.

If you want to do back translation to a large file, you can change the `replicas` and `worker_id` arguments in `run.sh`. For example, when `replicas=3`, we divide the data into three parts, and each `run.sh` will only process one part according to the `worker_id`.

General guidelines for setting hyperparameters:

UDA works out-of-box and does not require extensive hyperparameter tuning, but to really push the performance, here are suggestions about hyperparameters:

- It works well to set the weight on unsupervised objective '*unsup_coeff*' to 1.
- Use a lower learning rate than pure supervised learning because there are two loss terms computed on labeled data and unlabeled data respectively.
- If you have an extremely small amount of data, try to tweak '*uda_softmax_temp*' and '*uda_confidence_thresh*' a bit. For more details about these two hyperparameters, search the "Confidence-based masking" and "Softmax temperature control" in the paper.
- Effective augmentation for supervised learning usually works well for UDA.
- For some tasks, we observed that increasing the batch size for the unsupervised objective leads to better performance. For other tasks, small batch sizes also work well. For example, when we run UDA with GPU on CIFAR-10, the best batch size for the unsupervised objective is 160.

Acknowledgement

A large portion of the code is taken from BERT and RandAugment. Thanks!

Citation

Please cite this paper if you use UDA.

```
1 @article{xie2019unsupervised,  
2   title={Unsupervised Data Augmentation for Consistency Training},  
3   author={Xie, Qizhe and Dai, Zihang and Hovy, Eduard and Luong, Minh-  
4     Thang and Le, Quoc V},  
5   journal={arXiv preprint arXiv:1904.12848},  
6   year={2019}
```

Please also cite this paper if you use UDA for images.

```
1 @article{cubuk2019randaugment,  
2   title={RandAugment: Practical data augmentation with no separate  
3     search},  
4   author={Cubuk, Ekin D and Zoph, Barret and Shlens, Jonathon and Le,  
5     Quoc V},  
6   journal={arXiv preprint arXiv:1909.13719},  
7   year={2019}  
8 }
```

Disclaimer

This is not an officially supported Google product.