TextBrewer: An Open-Source Knowledge Distillation Toolkit for Natural Language Processing

Ziqing Yang, Yiming Cui, Zhipeng Chen, Wanxiang Che, Ting Liu, Shijin Wang, Guoping Hu

ACL 2020

State Key Laboratory of Cognitive Intelligence, iFLYTEK Research, Research Center for SCIR, Harbin Institute of Technology, iFLYTEK AI Research (Hebei)
Pretrained Models in NLP

- Size of large pretrained models
  - ELMo – 94M parameters
  - BERT\textsubscript{base} – 108M parameters
  - BERT\textsubscript{large} – 334M parameters
  - GPT2 – 1.5B parameters
  - T5-11B – 11B parameters
  - GPT3 – 170B parameters
Pretrained Models in NLP

Large models

• Slow inference speed
• State-of-the-art performance
• Resource-demanding

Small models

• Fast inference speed
• Inferior performance
• Resource-friendly
What is knowledge distillation (KD)?

**Knowledge distillation** is a technique of transferring knowledge from a large (teacher) model to a small (student) model, *without significant loss in performance.*
Motivation

• Various distillation methods usually share a common workflow:

1. Reduce redundant coding
2. Distillation losses/strategies as plugins
3. Achieve great flexibility in experimenting with different methods

• A reusable distillation workflow framework
TextBrewer

• A PyTorch based knowledge distillation toolkit for NLP

• Features
  • Flexibility
    • customizable configurations
  • Easy-to-use
    • re-uses the most parts of their existing training scripts
  • Wide-model-support
    • especially transformer-based models
  • Built for NLP
    • have been test on different tasks

• Available at: http://textbrewer.hfl-rc.com
TextBrewer

• A PyTorch based knowledge distillation toolkit for NLP

Tasks

<table>
<thead>
<tr>
<th></th>
<th>Models</th>
<th>Distillation modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text Classification</td>
<td>BERT/RoBERTa</td>
<td>Traditional</td>
</tr>
<tr>
<td>MRC</td>
<td>Electra</td>
<td>Multi-teacher</td>
</tr>
<tr>
<td>Sequence Labeling</td>
<td>LSTM/GRU</td>
<td>Multi-task</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>normal training</td>
</tr>
</tbody>
</table>

Available at: http://textbrewer.hfl-rc.com
Architecture

- An overview of the main functionalities of TextBrewer
Distillers

• Automatically train/distill and save models

• Five distillers:
  1. BasicDistiller
  2. GeneralDistiller
  3. MultiTeacherDistiller
  4. MultiTaskDistiller
  5. BasicTrainer
Configurations

1. TrainingConfig
   • gradient accumulation steps
   • checkpoint frequency
   • log directory
   • output directory
   • device

2. DistillationConfig
   • temperature
   • KD loss type
   • KD loss weight
   • hard-label loss weight
   • intermediate matches
   • if enable caching logits
   • ...
1. TrainingConfig

```json
{ "gradient_accumulation_steps": 1,
  "ckpt_epoch_frequency": 1,
  "ckpt_steps": "None",
  "log_dir": ".\logs",
  "output_dir": ".\saved_models",
  "device": "cuda"}
```

2. DistillationConfig

```json
{"temperature": 8,
 "temperature_scheduler": None,
 "hard_label_weight": 0,
 "hard_label_weight_scheduler": None,
 "kd_loss_type": "ce",
 "kd_loss_weight": 1,
 "kd_loss_weight_scheduler": None,
 "probability_shift": False,
 "intermediate_matches": [
  1 { "layer_T": 0, "layer_S": 0, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear", 312, 768]},
  2 { "layer_T": 3, "layer_S": 1, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear", 312, 768]},
  3 { "layer_T": 6, "layer_S": 2, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear", 312, 768]},
  4 { "layer_T": 9, "layer_S": 3, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear", 312, 768]},
  5 { "layer_T": 12, "layer_S": 4, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear", 312, 768]},
  6 { "layer_T": [0,0], "layer_S": [0,0], "feature": "hidden",
     "loss": "nst", "weight": 1},
  7 { "layer_T": [3,3], "layer_S": [1,1], "feature": "hidden",
     "loss": "nst", "weight": 1},
  8 { "layer_T": [6,6], "layer_S": [2,2], "feature": "hidden",
     "loss": "nst", "weight": 1},
  9 { "layer_T": [9,9], "layer_S": [3,3], "feature": "hidden",
     "loss": "nst", "weight": 1},
  10 { "layer_T": [12,12], "layer_S": [4,4], "feature": "hidden",
       "loss": "nst", "weight": 1}]
```
• Distillation with TextBrewer:

1. Initialize configurations and a distiller
2. Define adaptors and a callback function
3. Call the train method of the distiller
Distillation with TextBrewer

```python
train_config = TrainingConfig()
distill_config = DistillationConfig()

distiller = GeneralTrainer(train_config = train_config, distill_config = distill_config,
model_T = teacher, model_S = student,
adaptor_T = my_adaptor_T, adaptor_S = my_adaptor_S)

with distiller:
    distiller.train(optimizer, dataloader, num_epochs, callback=my_callback)
```
Adaptor

• Translates the inputs and outputs for the distiller

```python
adatpor(batch: Union[Dict, Tuple], model_outputs: Tuple) -> Dict

class Model(nn.Module):
    def forward(self, input_ids, attention_mask, labels, ...):
        ...
        return logits, hidden_states, loss

def simpleAdaptor(batch, model_outputs):
    return {'logits': (model_outputs[0],),
            'hidden': model_outputs[1],
            'input_mask': batch[1]}
```
Callback

callback(model: torch.nn.Module, step: int) -> Any

- For monitoring performance during training

```python
def predict(model, eval_dataset, step, args):
    # your evaluation code here
    ...

my_callback = partial(predict, eval_dataset=my_eval_dataset, args=args)
with distiller:
    distiller.train(..., callback=my_callback)
```
Minimal workflow

```python
from textbrewer import GeneralDistiller
from textbrewer import TrainingConfig, DistillationConfig

# We omit the initialization of models, optimizer, and dataloader.
teacher_model: torch.nn.Module = ...
student_model: torch.nn.Module = ...
dataloader: torch.utils.data.DataLoader = ...
optimizer: torch.optim.Optimizer = ...
scheduler: torch.optim.lr_scheduler = ...

def simple_adaptor(batch, model_outputs):
    # We assume that the first element of model_outputs
    # is the logits before softmax
    return {'logits': model_outputs[0]}

train_config = TrainingConfig()
distill_config = DistillationConfig()
distiller = GeneralDistiller(
    train_config=train_config, distill_config = distill_config,
    model_T = teacher_model, model_S = student_model,
    adaptor_T = simple_adaptor, adaptor_S = simple_adaptor)
distiller.train(optimizer, scheduler,
                dataloader, num_epochs, callback= None)
```

define adaptor

Initialize configurations and distiller

RUN!
Experiments

• English datasets
  • MNLI  
    sentence-pair classification
  • SQuAD  
    machine reading comprehension
  • CoNLL-2003  
    named entity recognition

• Chinese datasets
  • XNLI and LCQMC  
    sentence-pair classification
  • CMRC 2018 and DRCD  
    SQuAD-like machine reading comprehension

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Task</th>
<th>Metrics</th>
<th>#Train</th>
<th>#Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNLI</td>
<td>Classification</td>
<td>Acc</td>
<td>393K</td>
<td>20K</td>
</tr>
<tr>
<td>SQuAD</td>
<td>MRC</td>
<td>EM/F1</td>
<td>88K</td>
<td>11K</td>
</tr>
<tr>
<td>CoNLL-2003</td>
<td>NER</td>
<td>F1</td>
<td>23K</td>
<td>6K</td>
</tr>
<tr>
<td>XNLI</td>
<td>Classification</td>
<td>Acc</td>
<td>393K</td>
<td>2.5K</td>
</tr>
<tr>
<td>LCQMC</td>
<td>Classification</td>
<td>Acc</td>
<td>293K</td>
<td>8.8K</td>
</tr>
<tr>
<td>CMRC 2018</td>
<td>MRC</td>
<td>EM/F1</td>
<td>10K</td>
<td>3.4K</td>
</tr>
<tr>
<td>DRCD</td>
<td>MRC</td>
<td>EM/F1</td>
<td>27K</td>
<td>3.5K</td>
</tr>
</tbody>
</table>
**Experiments**

- **Model configurations**

<table>
<thead>
<tr>
<th>Model</th>
<th># Layers</th>
<th>Hidden size</th>
<th>Feed-forward size</th>
<th># Parameters</th>
<th>Relative size</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT_{BASE} (teacher)</td>
<td>12</td>
<td>768</td>
<td>3072</td>
<td>108M</td>
<td>100%</td>
</tr>
<tr>
<td>T6</td>
<td>6</td>
<td>768</td>
<td>3072</td>
<td>65M</td>
<td>60%</td>
</tr>
<tr>
<td>T3</td>
<td>3</td>
<td>768</td>
<td>3072</td>
<td>44M</td>
<td>41%</td>
</tr>
<tr>
<td>T3-small</td>
<td>3</td>
<td>384</td>
<td>1536</td>
<td>17M</td>
<td>16%</td>
</tr>
<tr>
<td>T4-tiny</td>
<td>4</td>
<td>312</td>
<td>1200</td>
<td>14M</td>
<td>13%</td>
</tr>
<tr>
<td>BiGRU</td>
<td>1</td>
<td>768</td>
<td>-</td>
<td>31M</td>
<td>29%</td>
</tr>
</tbody>
</table>

- **English models**: initialized with the weight released by Google
- **Chinese models**: initialized with RoBERTa-wwm-ext
## Results on English datasets

- **Single-teacher distillation**
  - All the T6 models achieve 99% performance of the teachers
  - T4-tiny outperforms TinyBERT
  - T4-tiny outperforms T3-small in most cases
  - Data augmentation (DA) is critical

<table>
<thead>
<tr>
<th>Model</th>
<th>MNLI m</th>
<th>MNLI mm</th>
<th>SQuAD EM</th>
<th>SQuAD F1</th>
<th>CoNLL-2003 F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERTBASE</td>
<td>83.7</td>
<td>84.0</td>
<td>81.5</td>
<td>88.6</td>
<td>91.1</td>
</tr>
<tr>
<td><strong>Public</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DistilBERT</td>
<td>81.6</td>
<td>81.1</td>
<td>79.1</td>
<td>86.9</td>
<td>-</td>
</tr>
<tr>
<td>TinyBERT</td>
<td>80.5</td>
<td>81.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>+DA</td>
<td>82.8</td>
<td>82.9</td>
<td>72.7</td>
<td>82.1</td>
<td>-</td>
</tr>
<tr>
<td><strong>TextBrewer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiGRU</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>85.3</td>
</tr>
<tr>
<td>T6</td>
<td>83.6</td>
<td>84.0</td>
<td>80.8</td>
<td>88.1</td>
<td>90.7</td>
</tr>
<tr>
<td>T3</td>
<td>81.6</td>
<td>82.5</td>
<td>76.3</td>
<td>84.8</td>
<td>87.5</td>
</tr>
<tr>
<td>T3-small</td>
<td>81.3</td>
<td>81.7</td>
<td>72.3</td>
<td>81.4</td>
<td>78.6</td>
</tr>
<tr>
<td>T4-tiny</td>
<td>82.0</td>
<td>82.6</td>
<td>73.7</td>
<td>82.5</td>
<td>77.5</td>
</tr>
<tr>
<td>+DA</td>
<td>-</td>
<td>-</td>
<td>75.2</td>
<td>84.0</td>
<td>89.1</td>
</tr>
</tbody>
</table>
Results on English datasets

• Multi-teacher distillation
  • All teachers are BERT$_{\text{base}}$
  • Student model is the same as the teacher
  • The student achieves the best performance, higher than the ensemble result

<table>
<thead>
<tr>
<th>Model</th>
<th>MNLI m</th>
<th>MNLI mm</th>
<th>SQuAD EM</th>
<th>SQuAD F1</th>
<th>CoNLL-2003 F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teacher 1</td>
<td>83.6</td>
<td>84.0</td>
<td>81.1</td>
<td>88.6</td>
<td>91.2</td>
</tr>
<tr>
<td>Teacher 2</td>
<td>83.6</td>
<td>84.2</td>
<td>81.2</td>
<td>88.5</td>
<td>90.8</td>
</tr>
<tr>
<td>Teacher 3</td>
<td>83.7</td>
<td>83.8</td>
<td>81.2</td>
<td>88.7</td>
<td>91.3</td>
</tr>
<tr>
<td>Ensemble</td>
<td>84.3</td>
<td>84.7</td>
<td>82.3</td>
<td>89.4</td>
<td>91.5</td>
</tr>
<tr>
<td>Student</td>
<td>84.8</td>
<td>85.3</td>
<td>83.5</td>
<td>90.0</td>
<td>91.6</td>
</tr>
</tbody>
</table>
Results on Chinese datasets

• Single-teacher distillation
  • T4-tiny still outperforms T3-small on all tasks
  • Consistent with the observations on English tasks
  • CMRC 2018 has a relatively small training set, DA has a much more significant effect

<table>
<thead>
<tr>
<th>Model</th>
<th>XNLI Acc</th>
<th>LCQMC Acc</th>
<th>CMRC 2018 EM</th>
<th>CMRC 2018 F1</th>
<th>DRCD EM</th>
<th>DRCD F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoBERTa-wwm</td>
<td>79.9</td>
<td>89.4</td>
<td>68.8</td>
<td>86.4</td>
<td>86.5</td>
<td>92.5</td>
</tr>
<tr>
<td>T3</td>
<td>78.4</td>
<td>89.0</td>
<td>63.4</td>
<td>82.4</td>
<td>76.7</td>
<td>85.2</td>
</tr>
<tr>
<td>+DA</td>
<td>-</td>
<td>-</td>
<td>66.4</td>
<td>84.2</td>
<td>78.2</td>
<td>86.4</td>
</tr>
<tr>
<td>T3-small</td>
<td>76.0</td>
<td>88.1</td>
<td>46.1</td>
<td>71.0</td>
<td>71.4</td>
<td>82.2</td>
</tr>
<tr>
<td>+DA</td>
<td>-</td>
<td>-</td>
<td>58.0</td>
<td>79.3</td>
<td>75.8</td>
<td>84.8</td>
</tr>
<tr>
<td>T4-tiny</td>
<td>76.2</td>
<td>88.4</td>
<td>54.3</td>
<td>76.8</td>
<td>75.5</td>
<td>84.9</td>
</tr>
<tr>
<td>+DA</td>
<td>-</td>
<td>-</td>
<td>61.8</td>
<td>81.8</td>
<td>77.3</td>
<td>86.1</td>
</tr>
</tbody>
</table>
Summary

• Conclusion
  • We present TextBrewer, a flexible PyTorch-based distillation toolkit for NLP research and applications.
  • TextBrewer is easy-to-use, and provides rich customization options.
  • A series of experiments shows that the distilled models can achieve state-of-the-art results with simple settings.

• Future work
  • Expand TextBrewer's scope of application
  • Automatic search for student model structures
Get TextBrewer

GitHub repo

http://textbrewer.hfl-rc.com

Install via pip

pip install textbrewer

If you like this project, you are welcome to give it a star!
Thanks for listening!

ziqingyang@gmail.com

http://textbrewer.hfl-rc.com