TinyBERT:

Distilling BERT for Natural Language Understanding

TinyBERT

- 중국 기업 **Huawei** 연구 논문
- 2019년 12월 27일 기준
  GLUE Leaderboard **25th** 랭크
- BERT 모델의 경량화 위해
  Knowledge Distillation (KD) 활용
- github repo:
  https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/TinyBERT

TINYBERT: DISTILLING BERT FOR NATURAL LANGUAGE UNDERSTANDING

Xiaoqi Jiao[1*†], **Yichun Yin**[2*], **Lifeng Shang**[2], **Xin Jiang**[2]
Xiao Chen[2], **Linlin Li**[3], **Fang Wang**[1] **and Qun Liu**[2]
[1]Huazhong University of Science and Technology
[2]Huawei Noah's Ark Lab
[3]Huawei Technologies Co., Ltd.

ABSTRACT

Language model pre-training, such as BERT, has significantly improved the performances of many natural language processing tasks. However, the pre-trained language models are usually computationally expensive and memory intensive, so it is difficult to effectively execute them on resource-restricted devices. To accelerate inference and reduce model size while maintaining accuracy, we firstly propose a novel *Transformer distillation* method that is specially designed for knowledge distillation (KD) of the Transformer-based models. By leveraging this new KD method, the plenty of knowledge encoded in a large "teacher" BERT can be well transferred to a small "student" TinyBERT. Moreover, we introduce a new two-stage learning framework for TinyBERT, which performs Transformer distillation at both the pre-training and task-specific learning stages. This framework ensures that TinyBERT can capture the general-domain as well as the task-specific knowledge in BERT.

TinyBERT[1] is empirically effective and achieves more than 96% the performance of teacher $\text{BERT}_{\text{BASE}}$ on GLUE benchmark, while being **7.5x smaller** and **9.4x faster** on inference. TinyBERT is also significantly better than state-of-the-art baselines on BERT distillation, with only ~**28%** parameters and ~**31%** inference time of them.

Contents

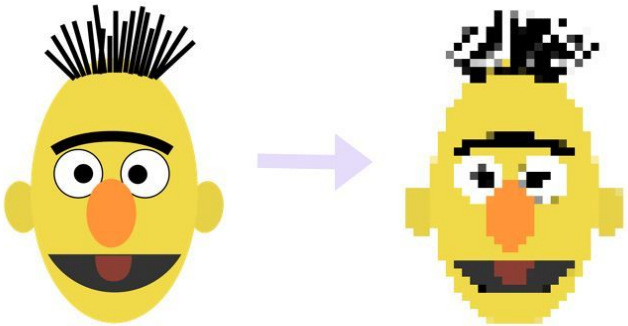
Abstract

- **Language model**을 통해 사전학습 수행하는 모델들은 자연어 처리의 큰 성능 발전 이끔 ***e.g.)*** *BERT*
- Problem: 그러나 사전학습을 거친 모델들은 대개 **많은 연산**을 필요로 하는 **큰 크기**의 모델이기에 리소스가 제한된 상황에서 해당 모델들을 사용하기란 쉽지 않음
- 따라서 우리는 모델의 정확도를 유지하며 Inference를 가속화하고, 모델 크기를 줄이기 위해 **Transformer distillation** 제안. 이는 **Transformer 기반의 모델**들을 위해 특별히 고안된 기법
- Goal: ***" 큰 크기의 Teacher BERT의 지식을 보다 작은 크기의 Student BERT에 이식 "***
- 이때, **Distillation**을 Pre-training과 Task-specific learning 스테이지 모두에 적용하자 ! (**Two-stage**)
  - TinyBERT가 **범용적 지식** 뿐만 아니라 **Task-specific한 지식**까지 얻도록 함
- Result: TinyBERT는 GLUE 벤치마크에서 BERT base 모델보다 7.5배 작은 크기, 9.4배 빠른 Inference 속도로 96%의 성능을 달성

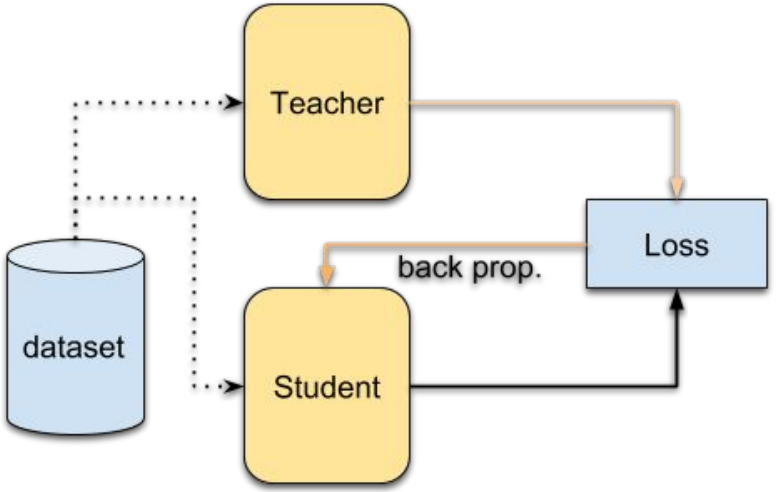
Introduction

- **LM Pre-training + downstream task Fine-tuning**은 자연어 처리의 새로운 패러다임 (이하 **PLM**)
- 그러나 **PLM** 모델들은 너무 많은 파라미터를 지니며, 긴 Inference 시간이 필요해

  모바일 폰과 같은 엣지 디바이스에서 사용하기에는 제약이 따름
- 최근 연구 결과에 따르면 PLM에 **불필요한 정보들이 과잉으로 포함**되어 있음
- 성능은 유지하되, 연산량과 모델 크기를 줄이는 것이 앞으로의 핵심 과제이다 !



Introduction



- 모델 크기를 줄이고 Inference 속도를 개선하는 **Compression** 기법에는 다양한 선택지들이 존재

  : **Quantization**, **Weights Pruning**, **Knowledge Distillation** (Teacher-Student Framework; 이하 **KD**)

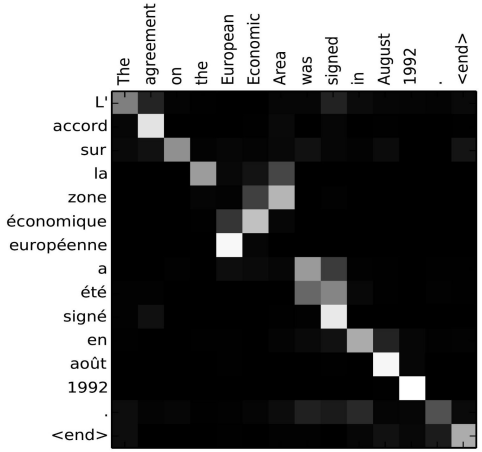
- 우리는 BERT Compression을 위해 **KD**에 집중하고자 함

- **KD**는 큰 크기의 Teacher network가 학습한 정보들을 작은 크기의 Student network로 이식하는 기법. **KD**를 활용해 Transformer 기반 모델들에 적용할 수 있는 **Transformer Distillation**을 제안할

Introduction

- Pre-training과 Fine-tuning을 거치는 BERT에 **KD**를 적용하기 위해서는 **두 번의 KD**를 수행해야 함
  - **General distillation (GD)**: Pre-training만 수행한 BERT가 Teacher network
  - **Task-specific distillation (TD)**: Task-specific하게 Fine-tuning한 BERT가 Teacher Network
- **GD**: Student TinyBERT는 **Transformer distillation**을 통해 Teacher network의 행동을 모방. 이를 통해 특정 downstream task에 Fine-tuning을 수행할 수 있는 **general TinyBERT**를 얻게 됨
- **TD**: **Task-specific distillation** 과정 중에는 Task-specific 한 데이터를 더 추가해 학습하기 위해 **Data augmentation**을 수행 -> **Augmented data**에 대하여 distillation 수행
  - Task-specific distillation 통해서는 **task-specific knowledge** 학습
- TinyBERT의 핵심 Contribution은 **Transformer Distillation**과 **Two-stage learning framework** !

Introduction



- Teacher BERT의 지식을 distillation하기 위해 다양한 Loss function 고안
  - 1) The output of **the embedding layer**
  - 2) **The hidden states** and **attention matrices** derived from the Transformer layer
  - 3) The logits output by **the prediction layer**
- Attention matrix에 Distillation을 적용하는 TinyBERT의 아이디어는

  BERT가 학습한 **Attention weight**에 언어학적 지식이 내포되어 있다는 최근 연구에서 기인

Introduction

Table 1: A summary of KD methods for BERT. Abbreviations: INIT(initializing student BERT with some layers of pre-trained teacher BERT), DA(conducting data augmentation for task-specific training data). Embd, Attn, Hidn, and Pred represent the knowledge from embedding layers, attention matrices, hidden states, and final prediction layers, respectively.

KD Methods	KD at Pre-training Stage					KD at Fine-tuning Stage				
	INIT	Embd	Attn	Hidn	Pred	Embd	Attn	Hidn	Pred	DA
Distilled $\text{BiLSTM}_{\text{SOFT}}$									✓	✓
BERT-PKD	✓							✓[3]	✓	
DistilBERT	✓				✓[4]				✓	
TinyBERT (our method)		✓	✓	✓		✓	✓	✓	✓	✓

TinyBERT와 기존 **BERT Compression 모델들**과의 비교표

Preliminaries

Transformer Layer

- 대부분의 PLM 모델들은 Transformer를 기반으로 모델링
- 최근 연구에 따르면 BERT 내 **Attention matrix**는 많은 양의 언어학적 정보들을 학습하고 있기 때문에 우리가 제안할 distillation 기법에서 핵심적으로 다룰 대상
  - Attention matrix가 학습하는 언어학적 지식으로는 **문법**, **상호참조 정보** 등이 있음
  - 그리고 두 지식은 모두 **Natural Language Understanding**에 있어 핵심적인 지식

Knowledge Distillation

- 큰 크기의 **Teacher network *T***가 학습한 정보를 작은 크기의 **Student network *S***에게 이식하는 기법
- Student network는 Loss를 통해 Teacher network의 행동을 모방하도록 학습

Preliminaries

- **Behavior function**: 네트워크에 들어온 **입력 값**을 유의미한 정보를 지니는 **Representation**으로 변환하는 함수. 즉, 입력 값을 출력 값으로 변환하는 함수
- **Transformer distillation**의 경우,
  Multi-head Attention, **Feed-forward layer**, 혹은 **Attention matrix** 등이 behavior function
- f^T, f^S가 각각 Teacher, Student network의 behavior function이라 했을 때,
  KD는 아래와 같은 손실 함수로 정의될 수 있음

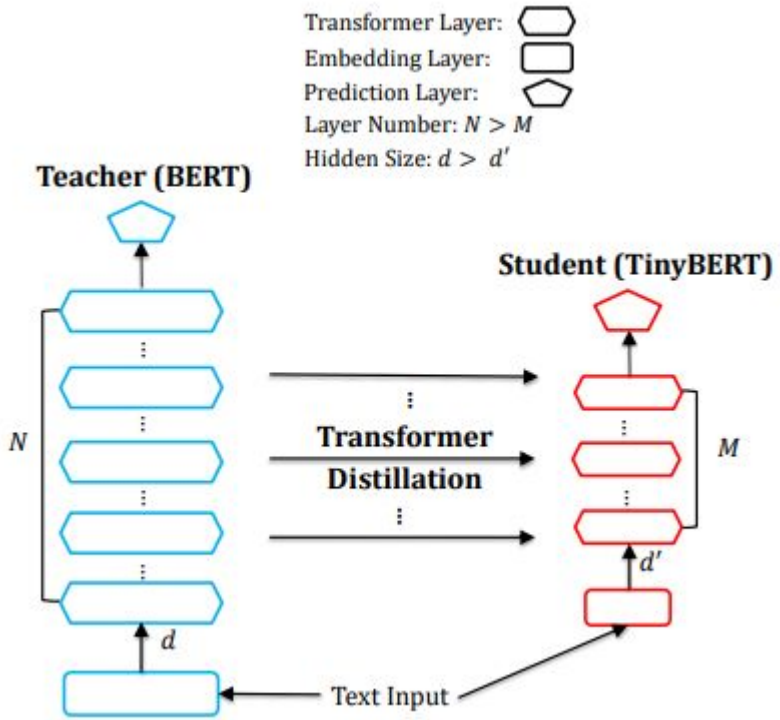
$$\mathcal{L}_{\mathrm{KD}} = \sum_{x \in \mathcal{X}} L\big(f^S(x), f^T(x)\big),$$

  - L(•)은 Teacher, Student network의 출력 값의 **차이를 비교**하는 손실 함수
  - f^T와 f^S가 각각 어떠한 behavior function으로 정의될지는 이후 섹션에서 설명

Method: Transformer Distillation

- Student가 **M개**의 Transformer layer를 Teacher가 **N개**의 layer를 지닌다 가정
- Student는 distillation를 위해 **M개의 layer를 모방**해야 함
- 이때, **n = g(m)**이라는 **Mapping function**을 활용해 학습
  - embedding layer: 0 = g(0)
  - prediction layer: N + 1 = g(M+1)

$$\mathcal{L}_{\text{model}} = \sum_{m=0}^{M+1} \lambda_m \mathcal{L}_{\text{layer}}(S_m, T_{g(m)}),$$

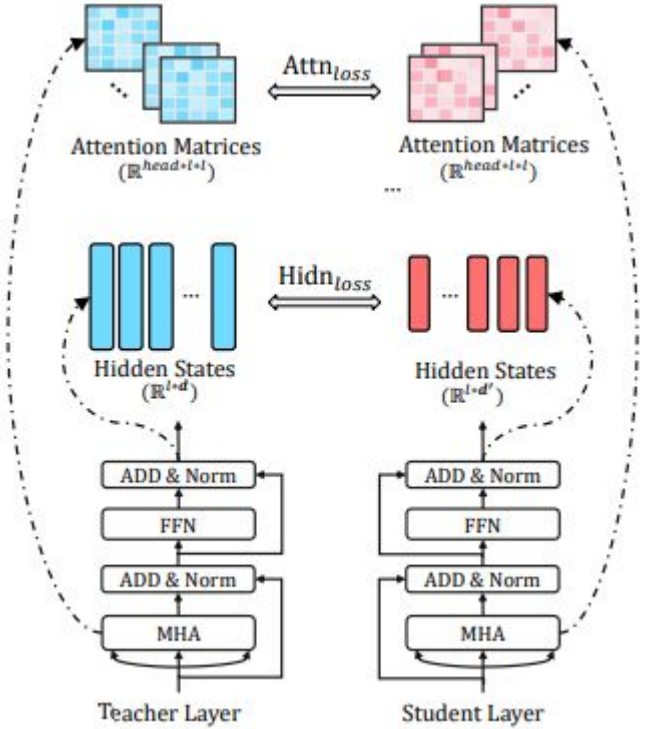


Method: Transformer Distillation (Transformer-layer)

- **Transformer-layer** distillation에는
  attention-based와 **hidden-states based**가 존재
- **Attention-based distillation**은 다음과 같이 정의:

$$\mathcal{L}_{\text{attn}} = \frac{1}{h}\sum_{i=1}^{h}\text{MSE}(\boldsymbol{A}_i^S, \boldsymbol{A}_i^T),$$

- h는 **head 개수**, A는 **Attention matrix**
- 위 수식에서 A는 Softmax가 취해진 이후 값이 아닌
  정규화 이전 값을 사용하는데,
  이는 실험 결과 Softmax를 취하지
  않은 값으로 학습할 때 모델이 보다
  빠르게 수렴하고 더 나은 성능을 보여주었기 때문



Method: Transformer Distillation (Transformer-layer)

- Transformer 레이어의 출력 값 **(hidden-states)** 에도 distillation 적용
- **Hidden-states based distillation**은 다음과 같이 정의:

$$\mathcal{L}_{\text{hidn}} = \text{MSE}(\boldsymbol{H}^S \boldsymbol{W}_h, \boldsymbol{H}^T),$$

- H^S는 (l x d') 크기의 행렬이며, H^T는 (l x d) 크기의 행렬
  - **d'**와 **d**는 각각 Student network와 Teacher network의 **hidden size**
  - **d'**는 Student network의 hidden size이기 때문에 대개 d 보다 작은 값
- W_h는 (d' x d) 크기의 행렬로 Student network의 hidden state 크기를

  Teacher network의 hidden state 크기와 맞추기 위해 존재하는 학습 가능한 **선형 변환**

Method: Transformer Distillation (Embedding-layer & Prediction-layer)

Embedding-layer distillation: $\mathcal{L}_{\text{embd}} = \text{MSE}(\boldsymbol{E}^S \boldsymbol{W}_e, \boldsymbol{E}^T),$

- **Embedding-layer distillation**도 앞선 distillation들과 유사하게 적용
- E^S와 E^T는 각각 Student와 Teacher network의 Embedding
- 본 논문에서는 embedding size와 hidden size가 같기에 마찬가지로 $\mathbf{W_e}$라는 선형변환 존재

Prediction-layer distillation: $\mathcal{L}_{\text{pred}} = -\text{softmax}(\boldsymbol{z}^T) \cdot \text{log\_softmax}(\boldsymbol{z}^S / t),$

- Teacher network의 **Prediction도 모방**하기 위해 **Prediction-layer distillation** 수행
- z^S와 z^T는 각각 Student와 Teacher의 **Logits vector**
- t는 **temperature value**로, 실험을 통해 1일 때 가장 성능이 좋음을 발견

Method: Transformer Distillation

$$\mathcal{L}_{\text{layer}}(S_m, T_{g(m)}) = \begin{cases} \mathcal{L}_{\text{embd}}(S_0, T_0), & m = 0 \\ \mathcal{L}_{\text{hidn}}(S_m, T_{g(m)}) + \mathcal{L}_{\text{attn}}(S_m, T_{g(m)}), & M \geq m > 0 \\ \mathcal{L}_{\text{pred}}(S_{M+1}, T_{N+1}), & m = M+1 \end{cases}$$

- 앞서 소개한 distillation 기법들을 종합해 **최종적으로 위와 같은 손실 함수**를 구성

Method: Code snippet

```
if args.continue_train:
    student_model = TinyBertForPreTraining.from_pretrained(args.student_model)
else:
    student_model = TinyBertForPreTraining.from_scratch(args.student_model)
teacher_model = BertModel.from_pretrained(args.teacher_model)
```

0) General distillation 중, Student 및 Teacher network 선언

Method: Code snippet

```
student_atts, student_reps = student_model(input_ids, segment_ids, input_mask)
teacher_reps, teacher_atts, _ = teacher_model(input_ids, segment_ids, input_mask)
```

1) 각 Network로부터 Attention matrix와 Hidden states 도출

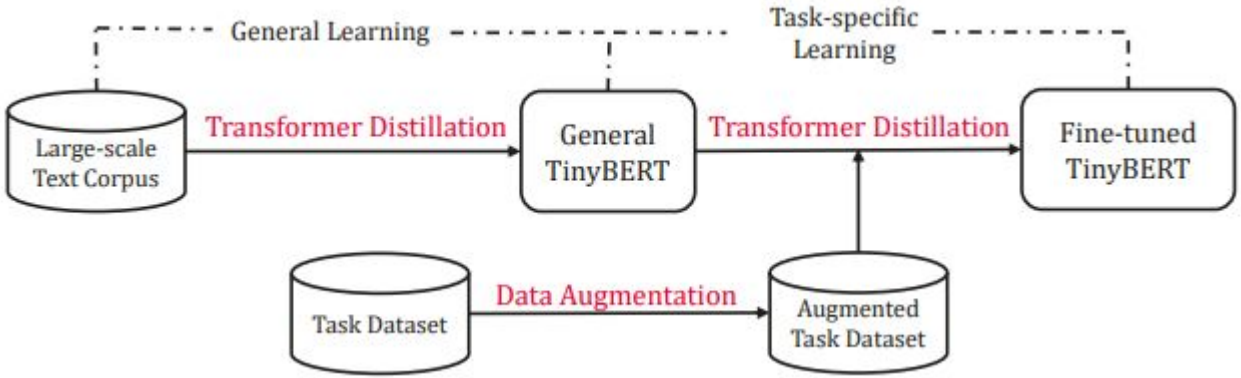
```
teacher_layer_num = len(teacher_atts)
student_layer_num = len(student_atts)
assert teacher_layer_num % student_layer_num == 0
layers_per_block = int(teacher_layer_num / student_layer_num)
new_teacher_atts = [teacher_atts[i * layers_per_block + layers_per_block - 1]
                    for i in range(student_layer_num)]
```

2) 두 Network의 num layers를 이용해 Mapping 할 layer 추출

```
for student_att, teacher_att in zip(student_atts, new_teacher_atts):
    student_att = torch.where(student_att <= -1e2, torch.zeros_like(student_att).to(device),
                              student_att)
    teacher_att = torch.where(teacher_att <= -1e2, torch.zeros_like(teacher_att).to(device),
                              teacher_att)
    att_loss += loss_mse(student_att, teacher_att)
```

3) Mapping 한 Layer의 Attention matrix 간 차이를 Loss로 사용

Method: TinyBERT Learning



- Pre-trained BERT를 Teacher network로 활용하는 **General distillation**은 TinyBERT가 **풍부한 범용 지식**을 학습하는데 큰 도움
- Task-specific distillation은 TinyBERT가 **Task-specific 지식**을 학습하는데 도움
- **두 번의 Distillation**을 통해 Teacher network와 Student network의 성능 차이를 줄일 수 있게 됨

Method: TinyBERT Learning

- General distillation은 **Pre-trained BERT**를 Teacher network로 사용해 진행되는데 hidden/embedding size, num layers의 감소로 인해 General TinyBERT의 성능은 그다지 좋지 않음
- **Task-specific distillation**에서는 Fine-tuning 된 BERT를 Teacher network로 사용
  +) Task-specific 한 훈련 데이터 양을 늘리기 위해 **Data augmentation** 적용
- Data augmentation 위해 **Pre-trained BERT**와 **GloVe** 활용
  - **Single-piece 단어**들에 한해서는 대체할 단어를 예측하기 위해 **LM**을 사용
  - **Multi-piece 단어**들에 한해서는 대체할 단어를 뽑아내기 위해 **GloVe Word embedding** 사용

```
if len(word_piece_ids) == 1:
    word_pieces[word_piece_ids[0]] = '[MASK]'
    candidate_words = self._masked_language_model(
        sentence, word_pieces, word_piece_ids[0])
elif len(word_piece_ids) > 1:
    candidate_words = self._word_distance(mask_token)
else:
    logger.info("invalid input sentence!")
```

Experiment: Model setup

- **Teacher network** (14.5M params)
  - Num layers **M = 4**
  - hidden size **d' = 312**
  - Feed-forward size = **1200**
  - head number: **12**
- **Student network** (109M params)
  - Num layers **N = 12**
  - hidden size **d = 768**
  - Feed-forward size = **3072**
  - head number = **12**
- **Mapping function**: g(m) = 3 x m
  - 이를 통해 TinyBERT는 $BERT_{base}$의 매 3번째 Transformer layer를 모방
  - **Learning weight lambda**는 1일 때, 가장 좋은 성능 보임

$$\mathcal{L}_{\text{model}} = \sum_{m=0}^{M+1} \lambda_m \mathcal{L}_{\text{layer}}(S_m, T_{g(m)}),$$

Experiment: Experimental results on GLUE

System	MNLI-m	MNLI-mm	QQP	SST-2	QNLI	MRPC	RTE	CoLA	STS-B	Average
$\text{BERT}_{\text{BASE}}$ (Google)	84.6	83.4	71.2	93.5	90.5	88.9	66.4	52.1	85.8	79.6
$\text{BERT}_{\text{BASE}}$ (Teacher)	83.9	83.4	71.1	93.4	90.9	87.5	67.0	52.8	85.2	79.5
$\text{BERT}_{\text{SMALL}}$	75.4	74.9	66.5	87.6	84.8	83.2	62.6	19.5	77.1	70.2
Distilled $\text{BiLSTM}_{\text{SOFT}}$	73.0	72.6	68.2	90.7	-	-	-	-	-	-
BERT-PKD	79.9	79.3	70.2	89.4	85.1	82.6	62.3	24.8	79.8	72.6
DistilBERT	78.9	78.0	68.5	91.4	85.2	82.4	54.1	32.8	76.1	71.9
TinyBERT	82.5	81.8	71.3	92.6	87.7	86.4	62.9	43.3	79.9	76.5

- TinyBERT가 이전까지 존재하던 Compression 모델들 보다 좋은 성능 기록
- Teacher network인 BERT base와 비교해도 괜찮은 성능 보임
- 많은 모델들이 CoLA에서 고전을 하는 모습: TinyBERT가 그 중 제일 낫다 !
  - CoLA: **The Corpus of Linguistic Acceptability**; 복잡한 언어학적 지식을 필요로 하는 태스크
  - ks08 0 * I drank some of water.
  - ks08 1 It is a golden hair.

Experiment: Effects of model size

System	MNLI-m	MNLI-mm	MRPC	CoLA	Average
$\text{BERT}_{\text{BASE}}$ (Teacher)	84.2	84.4	86.8	57.4	78.2
BERT-PKD (M=6;d'=768;d'_i=3072)	80.9	80.9	83.1	43.1	72.0
DistilBERT (M=6;d'=768;d'_i=3072)	81.6	81.1	82.4	42.5	71.9
TinyBERT (M=4;d'=312;d'_i=1200)	82.8	82.9	85.8	49.7	75.3
TinyBERT (M=4;d'=768;d'_i=3072)	83.8	84.1	85.8	50.5	76.1
TinyBERT (M=6;d'=312;d'_i=1200)	83.3	84.0	86.3	50.6	76.1
TinyBERT (M=6;d'=768;d'_i=3072)	84.5	84.5	86.3	54.0	77.3

- Default로 설정한 **TinyBERT** 보다 큰 사이즈의 TinyBERT들이 더 좋은 성능
- **CoLA**와 같이 어려운 태스크에서 성능 향상을 꾀하기 위해서는 모델을 보다 깊고 넓게 쌓아야 함
    - 단순히 Layer 수를 늘리거나, hidden size를 늘리는 것만으로는 큰 성능 향상 못 얻음

Experiment: Ablation studies

Table 5: Ablation studies of different procedures (i.e., TD, GD, and DA) of the two-stage learning framework. The variants are validated on the dev set.

System	MNLI-m	MNLI-mm	MRPC	CoLA	Average
TinyBERT	82.8	82.9	85.8	49.7	75.3
No GD	82.5	82.6	84.1	40.8	72.5
No TD	80.6	81.2	83.8	28.5	68.5
No DA	80.5	81.0	82.4	29.8	68.4

- 우리가 사용한 **TD** (Task-specific Distillation), **GD** (General Distillation) 그리고 **DA** (Data Augmentation) 는 KD에 있어 모두 중요한 기법으로 판명
- 사실 **TD + DA**의 조합만으로도 어느 정도의 성능을 달성할 수 있음
- 그러나, **No GD**와 **TinyBERT**의 점수 차이를 보게 되면 GD를 추가할 경우, **CoLA 태스크**에서 유의미한 성능 개선이 일어나는 것을 알 수 있음
  - 추측하기로는 GD를 통해 학습된 **보편적 언어학 지식**이 CoLA 태스크에 있어 중요한 역할을 담당하기 때문

Experiment: Ablation studies

Table 6: Ablation studies of different distillation objectives in the TinyBERT learning. The variants are validated on the dev set.

System	MNLI-m	MNLI-mm	MRPC	CoLA	Average
TinyBERT	82.8	82.9	85.8	49.7	75.3
No Embd	82.3	82.3	85.0	46.7	74.1
No Pred	80.5	81.0	84.3	48.2	73.5
No Trm	71.7	72.3	70.1	11.2	56.3
No Attn	79.9	80.7	82.3	41.1	71.0
No Hidn	81.7	82.1	84.1	43.7	72.9

- TinyBERT 학습에 있어 우리가 제안한 모든 **Distillation objective**는 유의미
- 특히, **Transformer-layer distillation**을 수행하지 않을 경우 가장 큰 성능 감소 포착
  - Transformer-layer distillation이 TinyBERT 학습에 있어 **Key**
- 그중에서도 **Attention-based**와 **hidden-states based distillation** 중 어느 것이 더 중요한지 실험
  - **Attention-based distillation**이 지표 상 더 중요한 것으로 나타나긴 했으나,
    두 기법은 상호보완적 관계이기에 TinyBERT가 좋은 성능을 얻기 위해서는 **두 기법이 모두 필요 !**

Experiment: Effects of mapping function

System	MNLI-m	MNLI-mm	MRPC	CoLA	Average
TinyBERT (Uniform-strategy)	82.8	82.9	85.8	49.7	75.3
TinyBERT (Top-strategy)	81.7	82.3	83.6	35.9	70.9
TinyBERT (Bottom-strategy)	80.6	81.3	84.6	38.5	71.3

- TinyBERT 학습에 사용되는 **Mapping function,** n = g(m)을 다양한 형태로 실험
- 앞서 언급한 **g(m) = 3 x m** 외 다른 두 함수를 추가적으로 실험
  - **상위 레이어**를 사용하는 **g(m) = m + N - M**
  - **하위 레이어**를 사용하는 **g(m) = m**
- 두 함수로 학습된 모델이 강점을 보이는 태스크가 서로 다름을 보여주고 있는 위 표는 각 태스크의 수행을 위해 **BERT의 서로 다른 레이어가 활용**되고 있음을 반증
- 기존 실험에 사용된 **Uniform-strategy**의 경우, 하위부터 상위까지 **BERT의 다양한 레이어를 활용**하기 때문에 다른 두 함수보다 좋은 성능을 보이게 되는 것으로 추정

Conclusion and Future work

- **Transformer-based distillation** 이라는 새로운 KD 방법론 제시
- 또한 **Two-stage framework**를 통해 KD를 두 번 수행
- 여러 실험을 통해 **TinyBERT**가 $\text{BERT}_{\text{base}}$ 에 비해 **작은 사이즈**와 **빠른 Inference 속도**를 유지하며, 비교적 **경쟁력 있는 성능**을 기록할 수 있음을 확인
- 이를 통해 여러 **엣지 디바이스**에 BERT 기반 자연어 처리 어플리케이션을 올릴 수 있는 가능성 확인
- 차후, $\text{BERT}_{\text{base}}$ 보다 큰 $\textbf{BERT}_{\textbf{large}}$, $\textbf{XLNet}_{\textbf{large}}$ 등의 모델을 TinyBERT로 이식하는 실험 수행 예정
- 또한 **Distillation**과 **Quantization**, **Pruning** 등을 혼합해 PLM을 압축하는 방법을 연구하는 것도 앞으로의 모델 압축에 있어 유망한 방향