트랜스포머를 활용한 자연어 처리

Chapter 8. 효율적인 트랜스포머 구축

24.08.07 박현빈

목차

- 의도탐지
- 벤치마크 정의
- 지식 정제 knowledge distillation
- 양자화 quantization
- ONNX(Open Neural Network Exchange)
- 가지치기 pruning

의도 탐지

- 주어진 텍스트가 어떤 의도를 가지는지 분류
- 데이터셋
  - CLINC150
    - 텍스트 – 의도 쌍

```
{'intent': 133, 'text': 'transfer $100 from my checking to saving account'}
```

벤치마크 클래스 정의

- 모델 성능 – 정확도
- 레이턴시 – input을 받고 prediction 생성까지 걸리는 시간
- 메모리 – 모델 크기

- BERT의 벤치마크(CLINC150으로 fine-tunning된 모델)

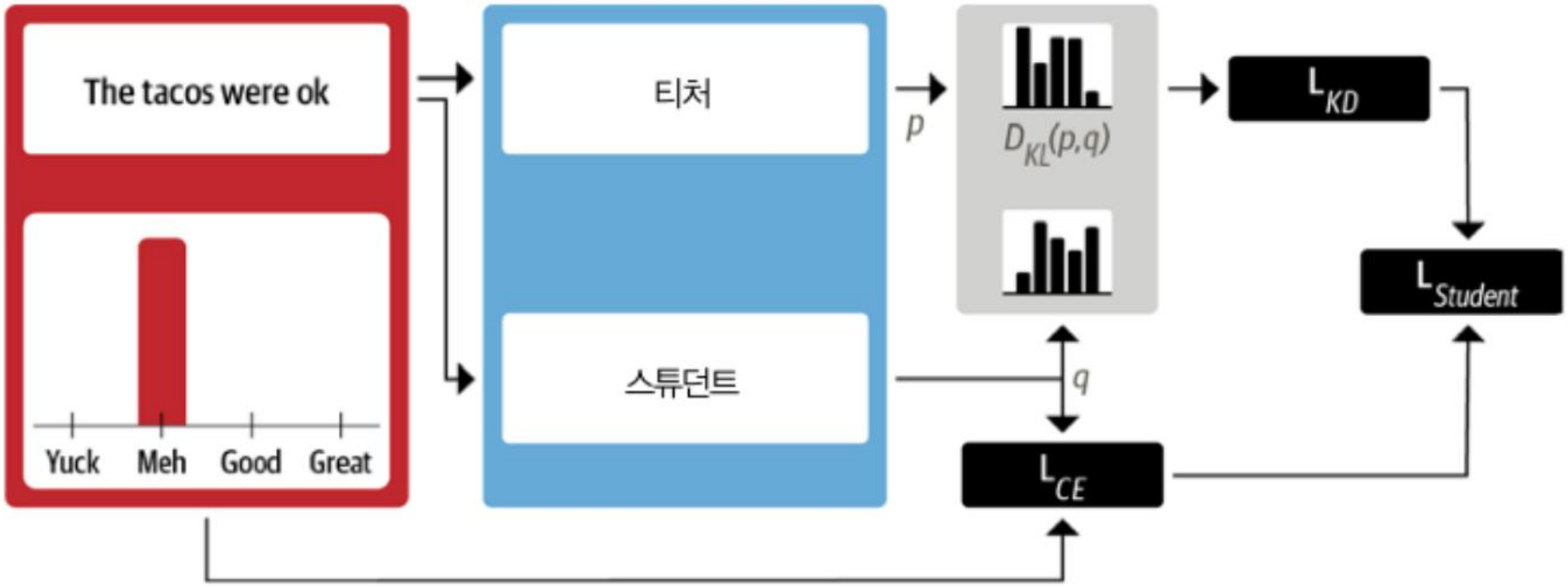
```
모델 크기 (MB) - 418.16
평균 레이턴시 (ms) - 54.20 +\- 1.91
테스트 세트 정확도 - 0.867
```

Distillation

- Teacher
  - 성능은 좋지만 크고 느린 모델
  - CLINC150으로 fine-tuning 된 BERT
- Student
  - 작고 빠르지만 성능이 낮은 모델
  - 기본 DistilBERT

- Student 모델이 Teacher 모델과 비슷한 예측을 하도록 훈련

Student 모델의 Loss function



Student 모델의 Loss function

$$L_{student} = \alpha L_{CE} + (1 - \alpha) L_{KD}$$

크로스 엔트로피
Student가 정답 레이블을 잘 분류했는가

KL 발산
Student가 Teacher의 확률 분포를 잘 따라했는가

$$D_{KL}(p,\ q) = \sum_i p_i(x) \log \frac{p_i(x)}{q_i(x)} \qquad p_i(x) = \frac{\exp(z_i(x)/T)}{\sum_j \exp(z_i(x)/T)}$$

Distillation 코드

- Trainer

```python
class DistillationTrainer(Trainer):
    def __init__(self, *args, teacher_model=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.teacher_model = teacher_model

    def compute_loss(self, model, inputs, return_outputs=False):
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        inputs = inputs.to(device)
        outputs_stu = model(**inputs)
        # 스튜던트의 크로스 엔트로피 손실과 로짓을 추출합니다
        loss_ce = outputs_stu.loss
        logits_stu = outputs_stu.logits
        # 티처의 로짓을 추출합니다
        with torch.no_grad():
            outputs_tea = self.teacher_model(**inputs)
            logits_tea = outputs_tea.logits
        # 확률을 부드럽게하고 정제 손실을 계산합니다
        loss_fct = nn.KLDivLoss(reduction="batchmean")
        loss_kd = self.args.temperature ** 2 * loss_fct(
            F.log_softmax(logits_stu / self.args.temperature, dim=-1),
            F.softmax(logits_tea / self.args.temperature, dim=-1))
        # 가중 평균된 스튜던트 손실을 반환합니다
        loss = self.args.alpha * loss_ce + (1. - self.args.alpha) * loss_kd
        return (loss, outputs_stu) if return_outputs else loss
```

DistilBERT 벤치마크

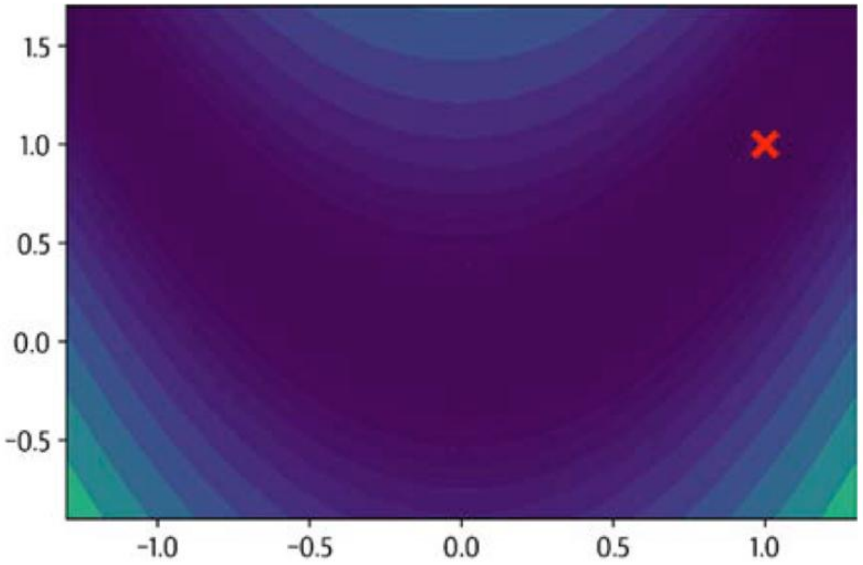
- DistilBERT의 벤치마크 (T=2, alpha=1)
  - Teacher로부터 어떤 신호도 받지 않은 순수 DistilBERT

```
모델 크기 (MB) - 255.89
평균 레이턴시 (ms) - 27.53 +\- 0.60
테스트 세트 정확도 - 0.858
```

- 성능을 높이기 위해 적절한 T, alpha 탐색
  - 그리디 서치
  - Optuna 프레임워크

Optuna

$$f(x, y) = (1 - x)^2 + 100\,(y - x^2)^2$$



(x, y) = (1, 1)일 때 최솟값 0을 가짐

```python
import optuna

study = optuna.create_study()
study.optimize(objective, n_trials=1000)
```

```python
study.best_params
```

```
{'x': 1.0884186262922515, 'y': 1.1894112108107937}
```

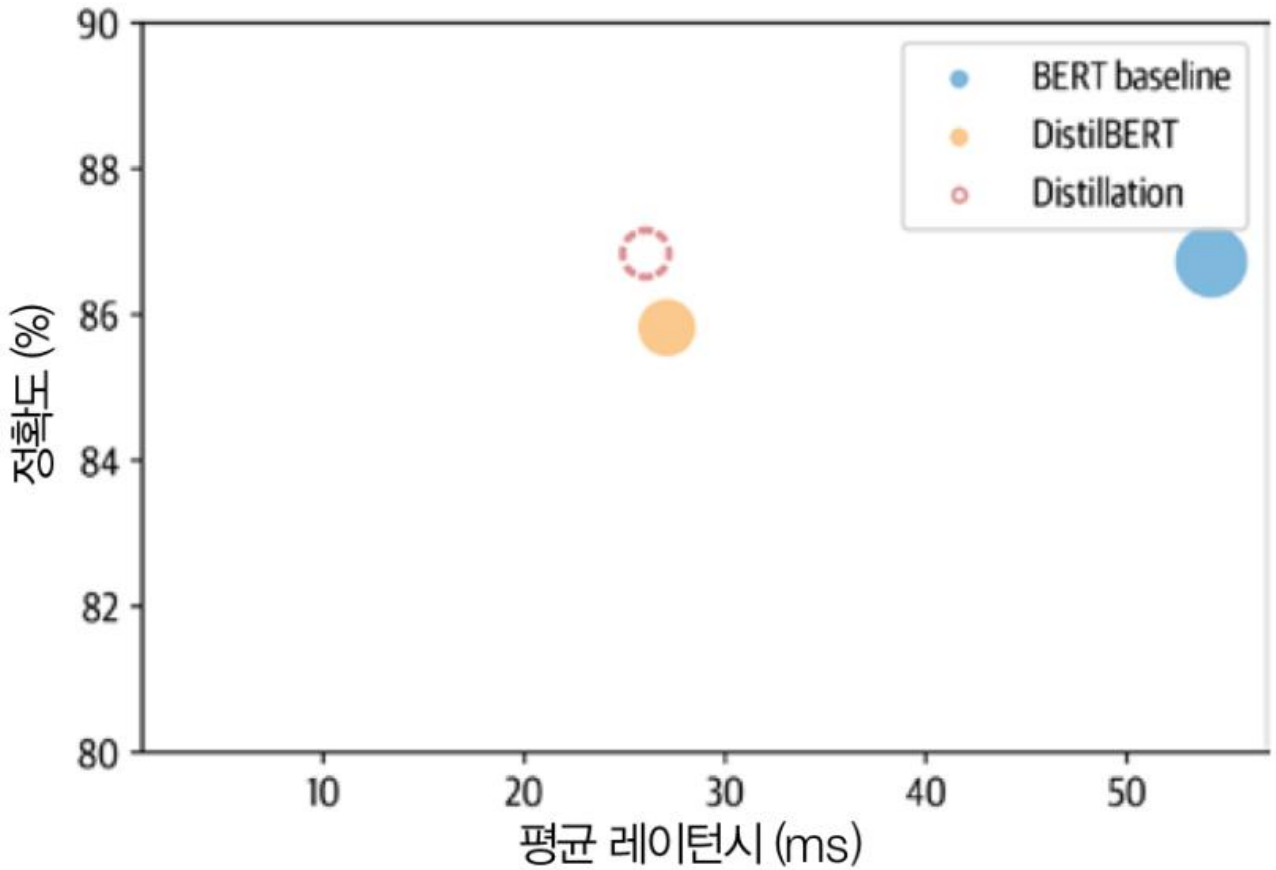
최적의 DistilBERT 벤치마크

- Optuna로 찾은 최적의 T=7, alpha=0.12
  - Teacher의 확률분포를 모방하는 것에 비중을 많이 둠

```
BestRun(run_id='1', objective=0.927741935483871,
hyperparameters={'num_train_epochs': 10, 'alpha': 0.12468168730193585,
'temperature': 7})
```

- DistilBERT의 벤치마크 (T=7, alpha=0.12)

```
모델 크기 (MB) - 255.89
평균 레이턴시 (ms) - 25.96 +\- 1.63
테스트 세트 정확도 - 0.868
```


90
88
86
84
82
80
정확도 (%)
10
20
30
40
50
평균 레이턴시 (ms)
BERT baseline
DistilBERT
Distillation

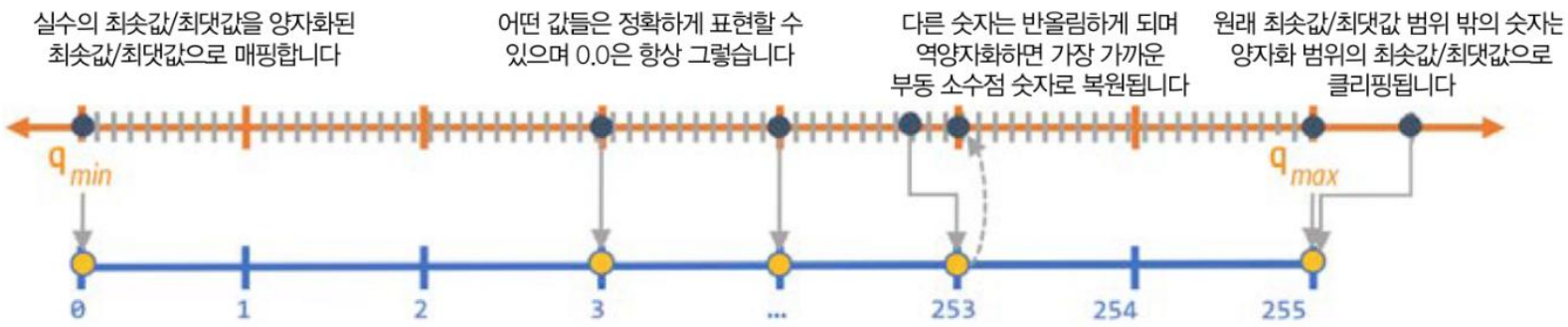
양자화

- 부동소수점(FP32)을 더 적은 비트 수의 정수(INT8)로 변환
- 연산 효율성 상승, 메모리 사용량 감소

```
tensor([[ -5,  -8,   0,  ...,  -6,  -4,   8],
        [  8,   3,   1,  ...,  -4,   7,   0],
        [ -9,  -6,   5,  ...,   1,   5,  -3],
        ...,
        [  6,   0,  12,  ...,   0,   6,  -1],
        [  0,  -2, -12,  ...,  12,  -7, -13],
        [-13,  -1, -10,  ...,   8,   2,  -2]], dtype=torch.int8)
```

양자화

- 부동소수점(FP32)을 더 적은 비트 수의 정수(INT8)로 변환
- 연산 효율성 상승, 메모리 사용량 감소



$$f = \left(\frac{f_{max} - f_{min}}{q_{max} - q_{min}} \right) (q - Z) = S(q - Z)$$

동적 양자화 vs 정적 양자화

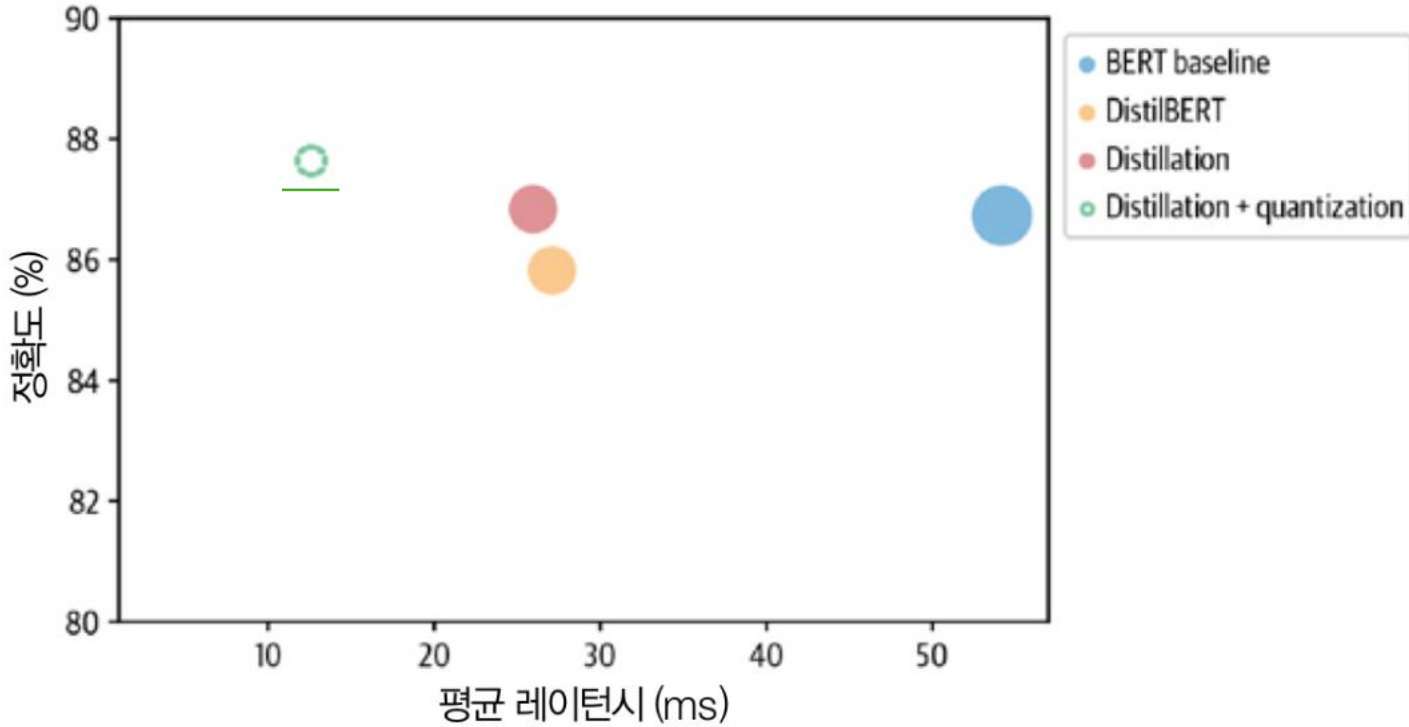
- 입력 data에 따라 부동소수점 범위가 달라지므로 Scale이 달라짐
- 매 Layer마다 부동소수점 범위 변동 □ 스케일 값 달라짐

	동적 양자화	정적 양자화
추론 전	모델 가중치, input 데이터 양자화	모델 가중치, input 데이터 양자화 매 layer마다 양자화 Scale 값 정의
추론	매 layer마다 Scale 계산하고 양자화 진행	미리 정의된 Scale값으로 양자화 진행
특징	느리지만 성능 유지	빠르지만 성능 감소

양자화 code

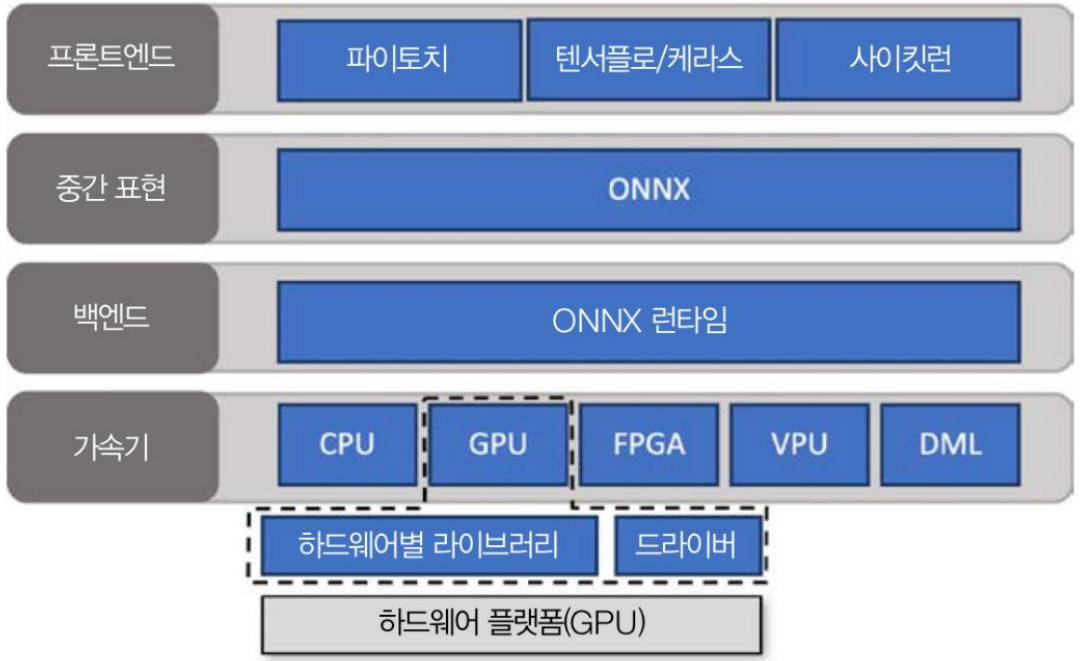
- Quntize_dynamic()

```
from torch.quantization import quantize_dynamic

# `haesun`를 자신의 허브 사용자 이름으로 바꾸세요.
model_ckpt = "haesun/distilbert-base-uncased-distilled-clinc"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
model = (AutoModelForSequenceClassification
         .from_pretrained(model_ckpt).to("cpu"))

model_quantized = quantize_dynamic(model, {nn.Linear}, dtype=torch.qint8)
```


90
88
86
84
82
80
정확도 (%)
10
20
30
40
50
평균 레이턴시 (ms)
BERT baseline
DistilBERT
Distillation
Distillation + quantization

ONNX, ORT(ONNX RunTime)

- 모델을 표현하기 위한 Open Format(양식)
- 프레임워크 간 모델 전환 용이
  - Pytorch로 만든 모델 □ ONNX format □ Tensorflow에서 import
- ORT(ONNX RunTime)
  - ONNX 포멧 전용 가속기
  - 속도 향상



ONNX 코드

- 환경 변수 설정

```
import os
from psutil import cpu_count

os.environ["OMP_NUM_THREADS"] = f"{cpu_count()}"
os.environ["OMP_WAIT_POLICY"] = "ACTIVE"
```

- convert()로 모델을 ONNX 포멧으로 변환

```
from transformers.convert_graph_to_onnx import convert

# `haesun`를 자신의 허브 사용자 이름으로 바꾸세요.
model_ckpt = "haesun/distilbert-base-uncased-distilled-clinc"
onnx_model_path = Path("onnx/model.onnx")
convert(framework="pt", model=model_ckpt, tokenizer=tokenizer,
        output=onnx_model_path, opset=12, pipeline_name="text-classification")
```

ONNX 코드

- ORT를 위한 InferenceSession 객체 생성

```
from onnxruntime import (GraphOptimizationLevel, InferenceSession,
                         SessionOptions)

def create_model_for_provider(model_path, provider="CPUExecutionProvider"):
    options = SessionOptions()
    options.intra_op_num_threads = 1
    options.graph_optimization_level = GraphOptimizationLevel.ORT_ENABLE_ALL
    session = InferenceSession(str(model_path), options, providers=[provider])
    session.disable_fallback()
    return session
onnx_model = create_model_for_provider(onnx_model_path)
```

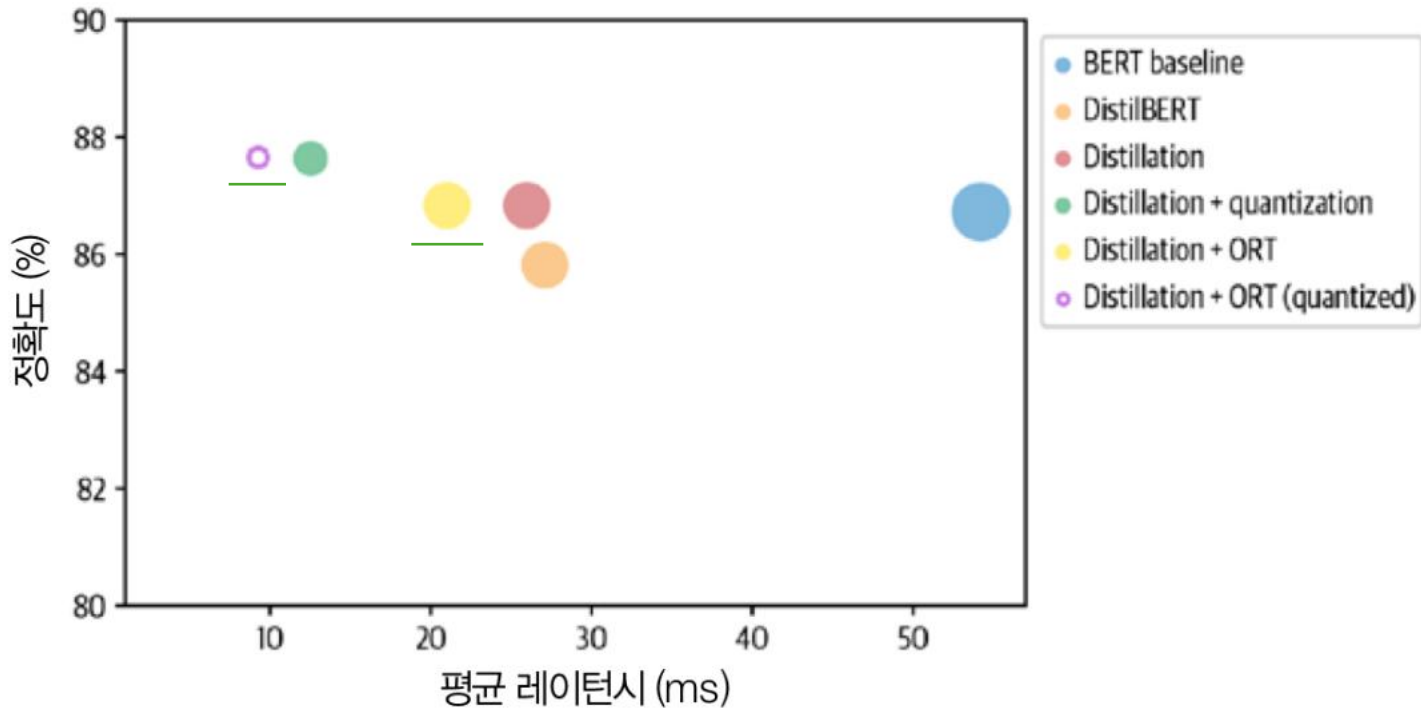
- run()으로 실행

```
inputs = clinc_enc["test"][:1]
del inputs["labels"]
logits_onnx = onnx_model.run(None, inputs)[0]
logits_onnx.shape
```

```
(1, 151)
```

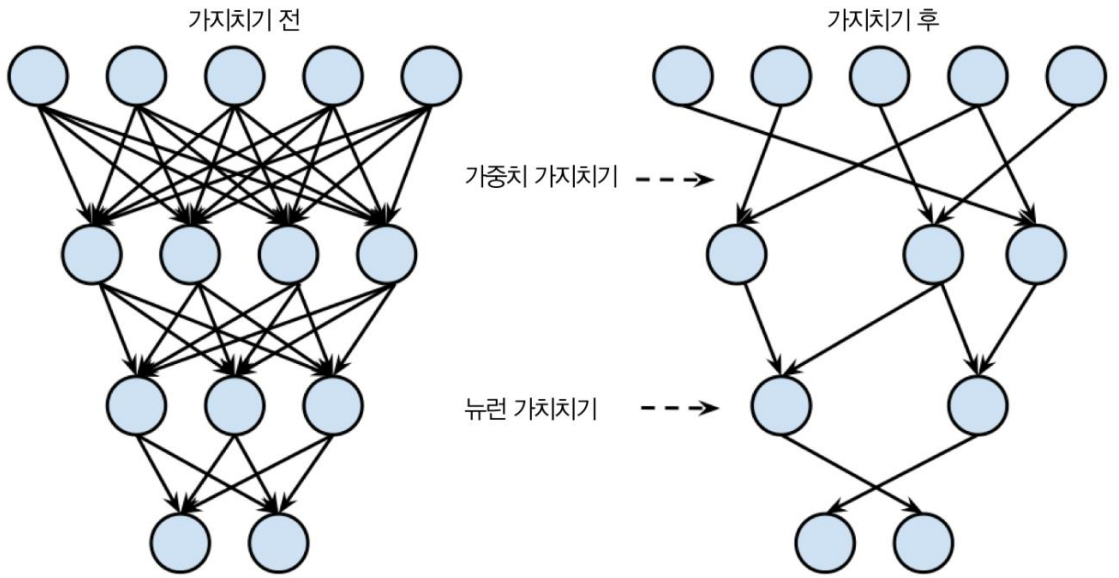
```
np.argmax(logits_onnx)
```

```
61
```


90
88
86
84
82
80
정확도 (%)
10
20
30
40
50
평균 레이턴시 (ms)
BERT baseline
DistilBERT
Distillation
Distillation + quantization
Distillation + ORT
Distillation + ORT (quantized)

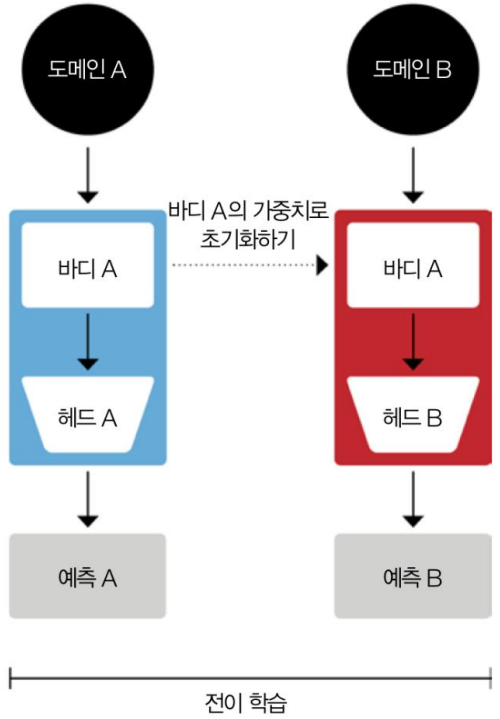
가지치기

- 신경망에서 중요하지 않은 가중치를 제거하여 모델 크기 감소
  - 메모리 사용량이 강력히 제한될 때 사용
- 0인 값을 가지는 파라미터가 더 많아 희소 행렬 포멧으로 저장
- 중요도 점수 S가 낮은 가중치 제거
- 제거 후 모델 재학습



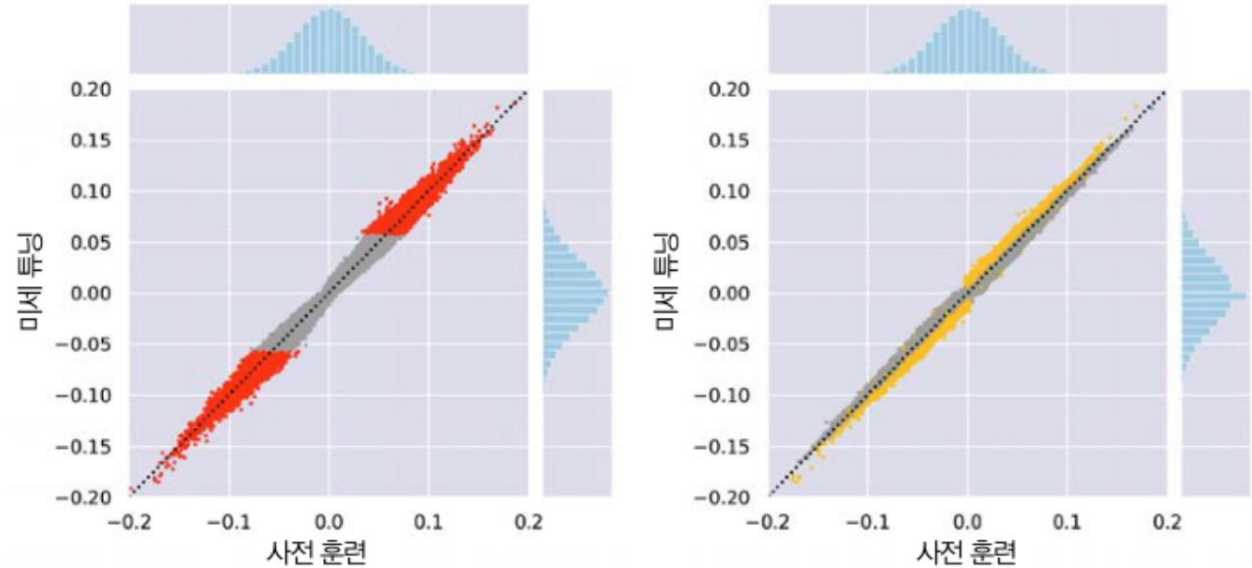
절댓값 가지치기

- 중요도 점수 S를 가중치의 절댓값으로 계산
- 가중치의 절댓값이 낮으면 삭제(마스킹)
- 전이학습과는 맞지 않음



이동 가지치기

- 전이 학습에서 사용 가능
- 중요도 점수 S를 사전학습 된 모델의 가중치와 fine-tuning 후의 가중치를 비교했을 때 원점으로부터 이동한 거리로 계산
  - 0.1 □ 0.12, S=0.02 −0.08 □ −0.1, S=0.02
  - 0.1 □ 0.08, S=−0.02 −0.08 □ −0.05, S=−0.03



절댓값
가지치기

이동
가지치기

결론

- 효율적 트랜스포머
  - Distillation, 양자화, ONNX, 가지치기
  - 모델의 크기와 레이턴시는 줄이면서 성능 유지

- 효율이 좋으므로 다양한 어플리케이션에서 활용 권장

Q&A