6주

Author	**Neil Houlsby , Andrei Giurgiu , Stanisław Jastrze, bski ,Bruna Morrone , Quentin de Laroussilhe**
Published Date	@2019년 6월 13일
keyword	
status	Reading
주차	6주차

(a) **Transformer 레이어 사이에 Adapter 삽입**
(b) **소수의 파라미터만 학습**
(c) **파라미터 효율성**
3. **논문과 코드의 핵심 연결점 요약**
코드에서 소수의 파라미터 학습을 확인하는 방법
코드의 학습 파라미터 제어를 확인할 수 있는 부분들:
03 | 실습 : 코드 분석
Adapter-BERT 프로젝트 분석
1. 사전 학습 (Unsupervised Pre-Training)
2. 과제별 미세 조정 (Fine-Tuning)
3. 최적화 전략 (Optimization Strategies)
4. 모델 구조 (Model Architecture)
5. 손실 함수 및 보조 목적 (Loss Functions and Auxiliary Objectives)
6. 평가 및 로깅 (Evaluation and Logging)
결론
코드 파일 순서
`tokenization.py` (텍스트 전처리 및 토크나이징)
1. **라이브러리 임포트**
2. **WordPieceTokenizer 클래스**
3. **BasicTokenizer 클래스**
4. **FullTokenizer 클래스**
5. **유틸리티 함수**
6. **코드의 핵심 기능 요약**
1. `from __future__ import absolute_import`
2. `from __future__ import division`
3. `from __future__ import print_function`
`tokenization_test.py` (토크나이저 테스트 파일)
1. **라이브러리 임포트**
2. **TokenizationTest 클래스**
3. `test_full_tokenizer` **메서드**
4. `test_basic_tokenizer_lower` **메서드**
5. `test_wordpiece_tokenizer` **메서드**
6. **기타 테스트 메서드**
7. **결론**
`modeling.py` (BERT 모델 정의 및 관련 함수)
1. **파일 개요**
2. `BertConfig` **클래스**
3. `BertModel` **클래스**
4. **Transformer 모델**
5. **기타 주요 함수**
6. **파일의 역할**
1. `BertConfig` **클래스**
주요 함수들:
2. `BertModel` **클래스**

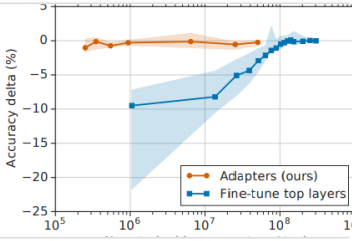
Parameter-Efficient Transfer Learning for NLP

00 | 참고자료

[논문 리뷰] Parameter-Efficient Transfer Learning for NLP (Adapter)

0. AbstractGPT, Bert와 같은 Large Language Model(LLM)을 Fine-tuning하는 것은 굉장히 효과적인 transfer mechanism이지만, fine-tuning은 많은 downstream task들에 대해 파라미터적으로 비효율적이다. (모든 테스크에 대해 항

https://ngp9440.tistory.com/143



01 | 논문정리

0. Abstract

GPT와 BERT와 같은 **대형 언어 모델(LLM)**을 **fine-tuning**하는 것은 매우 효과적인 전이 메커니즘(transfer mechanism)이다. 그러나, Fine-tuning은 많은 downstream task에서 **파라미터 비효율성** 문제를 야기함.

> why?
>
> 각 작업마다 **전체 모델을 다시 학습**해야 하며, 모든 **파라미터를 새로 학습**시키기 때문이다.
>
> → **즉, 모든 파라미터를 다시 학습시킨다!!!**

→ **어댑터 모듈(adapter modules)**을 통한 Transfer leaerning을 제시!!

🍭 **Adaptor Module**

: **압축적(compact)**이면서도 **확장 가능(extensible)**한 모델을 만들 수 있다.

- **압축적 모델(Compact Model)**: 작업마다 **소수의 파라미터**만 추가하여 문제를 해결하는 모델.
- **확장 가능 모델(Extensible Model)**: 이전 작업을 잊지 않고, **새로운 작업을 점진적으로 추가**하여 훈련할 수 있는 모델.

BERT Transformer 모델을 활용한 **전이 학습(transfer learning)** 실험을 통해 다음과 같은 결과를 도출

- **전체 미세 조정 방식과 비교**했을 때, **0.4% 이내의 성능 차이**를 보였다.
- 어댑터 모듈을 사용한 경우, **각 작업마다 3.6%의 파라미터**만 추가되었다.
- 반면, **미세 조정 방식**은 **각 작업마다 100%의 파라미터**를 학습해야 한다.

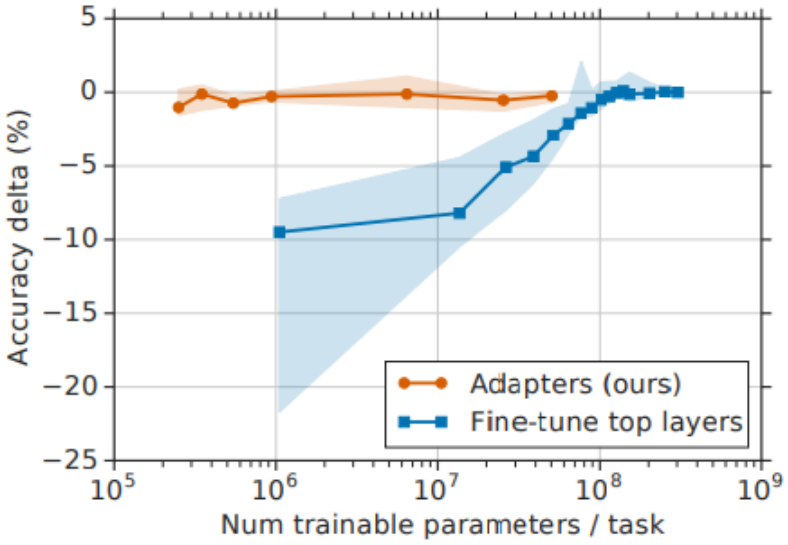
→ 어댑터 모듈은 더 **효율적**이면서도 **성능 저하 없이** 전이 학습을 수행할 수 있음을 보여준다.

1. Introduction

사전 학습된 모델을 활용한 **전이 학습(transfer learning)**은 NLP 작업에서 강력한 성능을 보여준다. 이전 연구들은 이를 통해 **텍스트 분류**나 **질문 응답** 같은 다양한 작업에서 매우 좋은 성능을 달성했음을 입증했다 (Dai & Le, 2015; Howard & Ruder, 2018; Radford et al., 2018).

예를 **들어, BERT는 대규모 텍스트 코퍼스를 대상으로 비지도 학습(unsupervised loss)**을 적용하여 학습된 **Transformer 네트워크**로, **텍스트 분류**와 **질문 응답** 작업에서 **최첨단 성능**을 달성하였다 (Devlin et al., 2018).

🍭 그래프 분석



- `y축` : 정확도
- `x축` : 학습가능한 파라미터의 숫자를 의미

→ adapter based tuning을 통해 적은 파라미터를 학습함에도 불구하고, full fine-tuning과 비슷한 성능을 달성할 수 있음을 의미한다.

🍭 adapter 모듈을 통해 달성할 Goal은??

- The goal is to **build a system that performs well on all of them**, but **without training an entire new model** for every new task.

즉, 새로운 테스크를 위해 항상 **전체 모델을 학습시키지 않으며** 이런 **새로운 테스크들에 대해 잘 작동하는 시스템을 개발**하는 것이 목표라고 한다.

테스크들 간에 높은 정도의 sharing은 (많은 양파라미터를 공유하는 것은) 클라우드 서비스와 같이 클라이언트로부터 다양한 요청(테스크)가 들어오는 application들에게 유용하다. 이를 위해 저자들은 compact하고 extensible한 downstream model을 생성할 수 있는 transfer learning 전략(=adapter tuning)을 제안한다.

1.1 기존 전이 학습 기법

NLP에서 일반적으로 사용되는 두 가지 **전이 학습 기법**

1. `특징 기반 전이(Feature-based Transfer)`와 2. `미세 조정(Fine-tuning)`이다.

→ 본 논문에서는 대신 **어댑터 모듈(Adapter Modules)** 활용한 **대안적 전이 방법**을 제시

1. **특징 기반 전이(Feature-based Transfer)**:
   - 이 기법은 사전 학습된 모델의 **실수 값 임베딩 벡터(embedding vectors)**를 사용한다.
     - 벡터 : **단어(Mikolov et al., 2013)**, **문장(Cer et al., 2019)**, 또는 **문단(Le & Mikolov, 2014)** 단위로 학습
   - 사전 학습된 임베딩 벡터는 다운스트림 작업에 맞게 **맞춤형 모델**에 입력된다.

   → 즉, pre-trained 임베딩 벡터들을 포함하도록 하는 방식
2. **미세 조정(Fine-tuning)**:
   - 미세 조정은 사전 학습된 네트워크의 가중치(weights)를 복사하여 다운스트림 작업에서 해당 가중치를 **조정**하는 방법이다.
   - 최근 연구에 따르면, 미세 조정이 특징 기반 전이보다 **더 나은 성능**을 보일 때가 많다 (Howard & Ruder, 2018).

하지만, **두 가지 기법 모두** 각 작업마다 **새로운 가중치**가 필요하다. 특히 **fine-tuning**의 경우, **네트워크의 하위 레이어**를 여러 작업에서 **공유**할 수 있지만, 여전히 작업마다 상당한 **파라미터**가 필요하다.

1.2 어댑터 기반 조정(Adapter Tuning)의 효율성

본 논문에서는 `어댑터 모듈(Adapter Modules)`을 기반으로 한 **조정 방법(tuning method)**을 제안한다.

> **Adapter?**
>
> **: 사전 학습된 네트워크의 레이어 사이**에 **새로운 모듈**을 추가하여 각 작업에 필요한 **소수의 파라미터**만 학습할 수 있다.

🍭 이전 방법들과의 차이는?

- *특징 기반 전이(Feature-based Transfer)**는 사전 학습된 **기본 함수(신경망)** φw(x)를 **새로운 작업에 맞게 추가 함수 χv(φw(x))**로 구성하고, **작업에 특화된 새로운 파라미터 v**만 학습한다.
- *미세 조정(Fine-tuning)**은 **기존 파라미터 w**를 **각 작업마다 조정**한다. 이는 여러 작업을 처리할 때 **압축성(compactness)**을 떨어뜨리는 요인이 된다.

BUT,

어댑터 조정(Adapter Tuning)은 사전 학습된 파라미터 `w`를 복사해 사용하고, **새로운 파라미터** `v`를 학습하면서도 `w`**는 고정된 상태**로 유지한다.

→ **파라미터 효율성**을 극대화하면서도 **기존 작업 성능을 유지**할 수 있다.

→ $|v| \ll |w|$일 경우, 작업별로 추가되는 파라미터 수는 매우 작아지며, 새로운 작업이 추가되더라도 **기존 작업에 영향을 미치지 않는다**.

1.3 어댑터 기반 조정과 다중 작업 학습 및 연속 학습

🍭 다중 작업 학습(Multi-task Learning) & 연속 학습(Continual Learning

- **다중 작업 학습(Multi-task Learning)**: 여러 작업을 **동시에 학습**해야 하며, 모든 작업에 대한 **동시적 접근**이 필요하다.

  → 반면, 어댑터 기반 조정은 각 작업에 **동시적 접근 없이도** 가능!.
- **연속 학습(Continual Learning)**: 이 방식은 **끝없는 작업 스트림**에서 학습하는 방법으로, **망각 문제**가 발생할 수 있다.

  → 어댑터 기반 조정은 작업 간 **파라미터가 공유**되지 않으며, **공유된 파라미터는 고정**되어 있기 때문에 **이전 작업을 잊지 않는다**.

→→ 어댑터 기반 조정은 **작업 간 독립성**을 보장하며, 각 작업에 **소수의 특화된 파라미터**만을 학습하여 **효율적이고 확장 가능한 학습**이 가능하다.

Multi-task learning은 compact model을 만들 수 있지만, 모든 테스크들에 대해 동시적 접근(simultaneous access)를 요구한다. (반면 adapter-based tuning은 그렇지 않다.)

Continual learning system은 extensible model을 만들 수 있지만 다른 테스크들을 계속 학습하면 이전 테스크를 잊어먹는 문제가 있다. (adapter는 공유된 파라미터들을 frozen하기 때문에 문제가 없다.)

1.4 어댑터 모듈의 실험 결과

🚨 **주요 혁신점!!**

1. 어댑터 모듈의 설계
2. **기본 모델과의 통합**

→ 특히, 효과적인 **`병목 구조(bottleneck architecture`를 제안한다.

- **GLUE 벤치마크**에서, 어댑터 모듈은 **전체 미세 조정된 BERT와 거의 동일한 성능**을 기록하였으며, **작업당 3%의 파라미터**만을 사용했다. 반면, 미세 조정은 **작업당 100%의 파라미터**를 사용했다.
- 추가로, **17개의 공개 텍스트 데이터셋**과 **SQuAD 질문 응답 작업**에서도 유사한 성능을 보였다.

결론적으로, **어댑터 기반 조정**은 **단일 확장 가능한 모델**을 생성하며, 이는 다양한 텍스트 분류 작업에서 **최신 성능에 근접**하는 성능을 달성했다.

2. Adapter Tuning for NLP

: **대규모 텍스트 모델**을 여러 다운스트림 작업에 맞게 조정할 수 있는 전략을 제시한다.

🍭 **전략 특징 3가지**

1. **좋은 성능**을 달성한다.
2. **작업을 순차적으로 학습**할 수 있으며, 모든 데이터셋에 **동시적 접근이 필요 없다**.
3. 각 작업에 대해 **소수의 추가 파라미터**만 필요하다.

→ 특히 클라우드 서비스와 같은 환경에서 매우 유용하다.

why? 클라우드 서비스에서는 **다수의 모델**을 여러 다운스트림 작업에 맞게 **순차적으로 학습**해야 하며, **파라미터 공유**가 많이 이루어져야 하기 때문이다.

2.1 bottleneck adapter module

🚨 **Bottleneck adapter module**

: **모델에 소수의 새로운 파라미터**를 추가한 후, 이를 다운스트림 작업에 맞게 학습하는 방식이다

→ 전통적인 미세 조정(vanilla fine-tuning)에서는 네트워크의 최상위 레이어(top layer)를 수정하여 **업스트림과 다운스트림 작업 간의 레이블 공간과 손실 함수**가 다르다는 점을 반영해야 한다.

> **But, 어댑터 모듈**은 사전 학습된 네트워크를 **다운스트림 작업에 맞게 재구성**하기 위해 더 일반적인 **아키텍처 수정**을 수행한다.

`어댑터 조정 전략(adapter tuning strategy)`은 **기존 네트워크에 새로운 레이어를 주입하는 방식**

- **기존 네트워크의 가중치는 변경되지 않음**
- 새로운 **어댑터 레이어**는 **무작위로 초기화**

→ 표준 미세 조정(standard fine-tuning)에서는 **새로운 최상위 레이어**와 **기존 가중치**가 함께 학습

> **But, 어댑터 조정**에서는 **기존 네트워크의 파라미터는 고정(frozen)**되며, 따라서 **여러 작업에 걸쳐 공유**될 수 있다.

2.2 어댑터 모듈의 주요 특징

1. **소수의 파라미터**:
   - **기존 네트워크 레이어에 비해 매우 적은 수의 파라미터**를 포함한다.

   → **새로운 작업이 추가**될 때 모델의 **전체 크기**는 상대적으로 **천천히 증가**한다.
2. **거의 동일한 초기화(near-identity initialization)**:
   - 어댑터 모듈의 **안정적인 학습**을 위해 **거의 동일한 초기화**가 필요(학습 초기 단계에서 **기존 네트워크의 가중치에 영향을 주지 않기** 위해)
   - 어댑터 모듈은 **거의 동일한 함수(near-identity function)**로 초기화되어, 학습이 시작될 때 **기존 네트워크가 영향을 받지 않도록** 한다. 이후 학습 과정에서 어댑터는 **활성화**되어 네트워크의 **활성화 분포(activation distribution)**를 변경할 수 있다.

어댑터 모듈은 **필요하지 않은 경우 무시될 수도 있으며**, 이는 실험에서 관찰된다. 일부 어댑터는 네트워크에 **큰 영향을 미치지 않는 반면**, 다른 어댑터는 중요한 역할을 할 수 있다. 또한, 어댑터 모듈이 **초기화에서 identity 함수로부터 너무 많이 벗어나면**, 모델이 학습에 실패할 수 있다는 점도 관찰되었다.

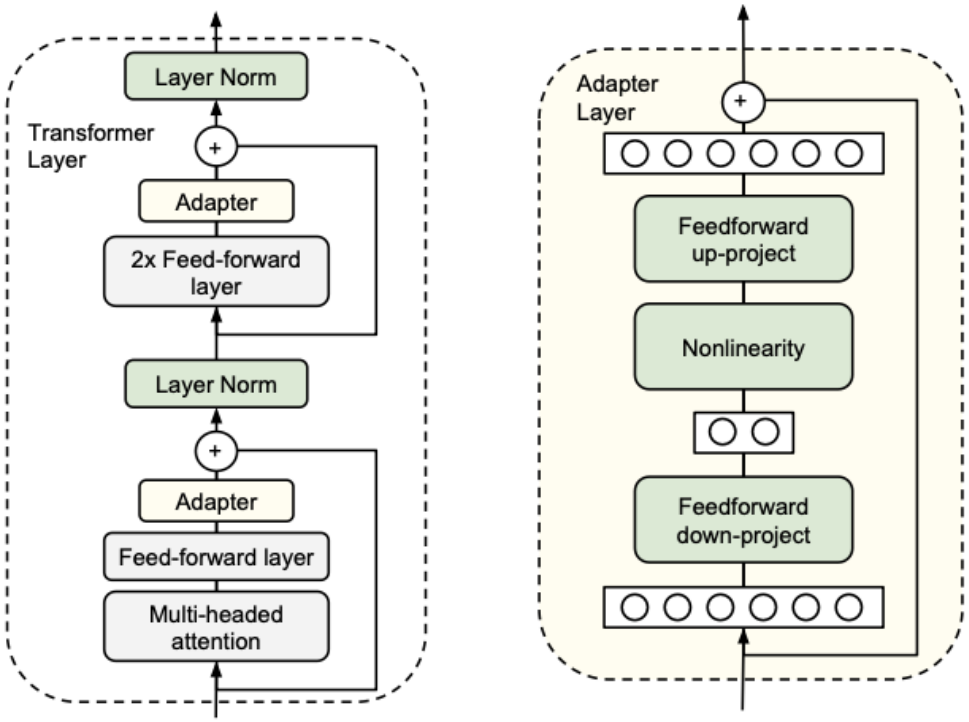
→ 어댑터 모듈을 사용하면 적은 수의 추가 파라미터로 성능을 유지하면서도, 기존 네트워크를 건드리지 않고 새로운 작업을 효율적으로 학습할 수 있다.

2.3 Transformer 네트워크에 대한 어댑터 기반 조정의 구현

어댑터 모듈을 구현하는 데는 여러 가지 아키텍처적 선택이 가능.

- 우리는 **간단한 설계**를 통해 좋은 성능을 달성.(더 복잡한 설계도 실험했지만(자세한 내용은 3.6장 참조), 우리가 제시한 전략은 여러 데이터셋에서 테스트한 결과 **동일한 성능**을 제공)

🚨 **Figure 2 : Adapter Archtiecture & How to adopt to Transformer**



🚧 **Transforemr의 레이어 분석 서브레이어**

: 두가지 주요 주요 ** `서브 레이어(sub-layers)` **로 구성

1. 어텐션 레이어(attention layer)
2. **피드포워드 레이어(feedforward layer)**

- 각 서브 레이어 뒤에는 `projection` 이 존재

  : 피처(feature) 크기를 다시 서브 레이어 입력 크기로 매핑

- `스킵 연결(skip-connection)` 은 각 서브 레이어에 적용
- 서브 레이어의 출력은 **레이어 정규화(layer normalization)**에 입력.

→ 각 서브 레이어 후에* `두 개의 직렬 어댑터(serial adapters)` 를 삽입하였다.

어댑터는 항상 서브 레이어 출력에 **직접 적용**되며, **투영(projection) 후**에, 그리고 **스킵 연결**이 적용되기 전에 배치된다. 어댑터 출력은 **레이어 정규화**로 바로 전달된다.

2.4 병목 아키텍처(bottleneck architecture) 설계

파라미터 수를 제한하기 위해 **병목 아키텍처**를 제안한다. 어댑터는 먼저 **d-차원 피처**를 **더 작은 차원 m**으로 투영한 뒤, 비선형성을 적용하고 다시 **d-차원**으로 투영한다.

- **추가되는 파라미터 수**는 **2md + d + m**이며, 여기서 **m ≪ d**로 설정해 각 작업에 추가되는 파라미터 수를 최소화한다.
- 실험적으로, 우리는 원래 모델의 **0.5%에서 8%** 정도의 파라미터만 추가하여 사용하였다.

병목 차원 m은 **성능**과 **파라미터 효율성** 사이에서 **절충점**을 제공하는 중요한 요소이다. 어댑터 모듈에는 **스킵 연결(skip-connection)**이 포함되어 있는데, 이는 투영 레이어가 **거의 0**에 가깝게 초기화될 경우, 모듈이 **거의 동일한 함수(approximate identity function)**로 작동하게 해준다.

어댑터 모듈과 함께 각 작업마다 **새로운 레이어 정규화 파라미터**도 학습된다. 이 기법은 **조건부 배치 정규화(conditional batch normalization)**, **FiLM**, **자기 조절(self-modulation)** 기법들과 유사하며, **각 레이어당 2d**의 파라미터만 추가된다. 하지만 **레이어 정규화 파라미터**만 학습하는 것은 충분한 성능을 보장하지 못하며, 이 부분은 3.4장에서 자세히 다룬다.

3. Experiments

: 어댑터 기반 조정(adapter-based tuning)이 **텍스트 작업**에서 **파라미터 효율적인 전이 학습**을 달성함을 입증

- `GLUE 벤치마크`
  - 어댑터 조정은 **BERT의 전체 미세 조정(full fine-tuning)**에 비해 **0.4% 이내의 성능 차이**
  - 미세 조정의 파라미터 수의 ****3%****만 추가

  → 이 결과는 추가적인 **17개의 공개 분류 작업**과 **SQuAD 질문 응답 작업**에서도 확인되었다.

  → **어댑터 기반 조정**은 자동으로 네트워크의 **상위 레이어**에 집중한다는 것을 보여준다.

3.1 실험 설정

🚧 사전 학습된 BERT Transformer 네트워크를 기본 모델로 사용

BERT로 분류 작업을 수행하기 위해, Devlin et al. (2018)에서 제안된 방식

각 시퀀스의 첫 번째 토큰은 `특수 "분류 토큰"`이며, 이 임베딩에 **선형 레이어**를 연결하여 **클래스 레이블**을 예측

훈련 절차 또한 Devlin et al. (2018)의 방법을 따름

- **Adam 옵티마이저**(Kingma & Ba, 2014)를 사용
- 학습 속도는 처음 10%의 스텝 동안 선형적으로 증가하고, 이후 선형적으로 **제로로 감소**.
- **Google Cloud TPU 4대**에서 **배치 크기 32**로 학습.
- 각 데이터셋과 알고리즘에 대해 **하이퍼파라미터 탐색(hyperparameter sweep)**을 진행하고, 검증 세트에서 정확도가 가장 높은 모델을 선택하였다.
- GLUE 작업에 대해서는 **테스트 메트릭**을 제출 웹사이트에서 제공받고, 나머지 분류 작업에 대해서는 **테스트 세트 정확도**를 보고

→ **fine-tuning**(전체 미세 조정)과 어댑터 조정을 비교하였다.

N개의 작업에 대해, 전체 미세 조정은 **사전 학습된 모델 파라미터의 N배**를 요구

목표 : **더 적은 파라미터**로 **fine-tuning과 동일한 성능**을 달성하는 것이며, **이상적으로는 1배에 가깝게** 만들고자 한다.

3.2 GLUE 벤치마크 평가

🚧 **사전 학습된 BERTLARGE 모델**을 사용

- 24개의 레이어
- 총 **3억 3천만 개의 파라미터**
- 소규모 하이퍼파라미터 탐색
- 학습률: $3 \cdot 10^{-5}, 3 \cdot 10^{-4}, 3 \cdot 10^{-3}$에서 탐색
- 학습 에포크 수 : {3, 20}에서 선택
- **고정된 어댑터 크기**(병목 차원의 수)를 사용하거나, {8, 64, 256} 중에서 작업별로 가장 적합한 크기를 선택하였다.

- 어댑터 크기는 **어댑터 특화 하이퍼파라미터** 중 유일하게 조정된 값이었다.
  - 학습 불안정성으로 인해, 우리는 **다른 무작위 시드(random seeds)**로 5번 반복 실행하였으며, 검증 세트에서 가장 성능이 좋은 모델을 선택

🍭 **Table 1 분석**

: 어댑터 조정은 평균 **80.0**의 GLUE 점수를 달성했으며, 이는 전체 미세 조정의 **80.4**에 매우 근접한 성능이다.

- 최적의 어댑터 크기는 데이터셋별로 다르게 나타냄

  EX) MNLI*에서는 256이 선택된 반면,* ***RTE****와 같은 가장 작은 데이터셋에서는 8이 선택*

- 어댑터 크기를 항상 64로 고정하면 평균 정확도는 약간 감소하여 **79.6**을 기록

→ **Table 1**의 모든 데이터셋을 해결하기 위해, 전체 미세 조정은 **BERT 파라미터의 9배**가 필요하다. 반면, 어댑터 조정은 **1.3배**의 파라미터만 필요하다.

3.3 추가 분류 작업

어댑터가 **컴팩트하고 성능이 좋은 모델**을 제공한다는 것을 추가적으로 검

→ **공개 텍스트 분류 작업**. (이 작업 세트는 매우 다양한 작업을 포함)

- **학습 예시 수:** 900에서 33만 개
- **클래스 수**는 2에서 157
- **텍스트 길이**는 평균 57자에서 1,900자

🔍 실험방식

- **배치 크기 : 32**
- **학습률 :** $1 \cdot 10^{-5}, 3 \cdot 10^{-5}, 1 \cdot 10^{-4}, 3 \cdot 10^{-3}$ 에서 넓게 탐색
- **학습 에포크 수:** {20, 50, 100}에서 수동으로 선택(데이터셋 수가 많음) ← 검증 세트 학습 곡선을 보고 결정

→ **fine-tuning**과 **어댑터** 모두에 대해 최적의 값을 선택

어댑터 크기를 {2, 4, 8, 16, 32, 64}에서 테스트하였다.

일부 작은 데이터셋의 경우 **전체 네트워크를 미세 조정하는 것이 최적이 아닐 수** 있으므로, 우리는 **가변 미세 조정(variable fine-tuning)**을 추가적인 기준선으로 설정하였다.

이를 위해, **최상위 n개의 레이어**만 미세 조정하고 나머지는 **고정**하였다. 우리는 n ∈ {1, 2, 3, 5, 7, 9, 11, 12}에서 탐색하였다.

이 실험에서는 **BERTBASE 모델**(12개의 레이어)을 사용하였기 때문에, n = 12일 경우 **전체 미세 조정**과 동일하게 된다.

GLUE 작업과 달리, 이 작업 세트에 대한 **최신 성능(state-of-the-art)**을 모두 포함한 종합적인 세트는 존재하지 않는다. 따라서, 우리는 **BERT 기반 모델이 경쟁력 있음을 확인**하기 위해 **우리만의 벤치마크 성능**을 수집하였다. 이를 위해 **대규모 하이퍼파라미터 탐색**을 실행하였고, **AutoML 알고리즘**(Zoph & Le, 2017; Wong et al., 2018)과 유사한 방식을 사용하였다. 이 알고리즘은 **사전 학습된 텍스트 임베딩 모듈**(TensorFlow Hub에서 제공)을 기반으로 **피드포워드 및 컨볼루션 네트워크** 공간을 탐색한다. 각 작업에 대해 **검증 세트 정확도**를 기준으로 **최종 모델**을 선택하였다.

🔍 **Table 2**

: AutoML 기준선, **fine-tuning**, **variable fine-tuning**, **어댑터 조정**의 결과를 보고한다.

1. **AutoML 기준선**은 BERT 모델이 **경쟁력 있음을** 입증한다.
2. **fine-tuning**은 **17×**의 BERTBASE 파라미터를 요구하였으며, **variable fine-tuning**은 평균적으로 **52%의 네트워크**를 학습하여 `9.9배의 파라미터`를 사용하였다.
3. 어댑터는 훨씬 더 **컴팩트한 모델**을 제공하며, 작업당 **1.14%의 새로운 파라미터**만 추가되어, **모든 17개의 작업**에서 `1.19배의 파라미터`만 사용하였다.

→ **어댑터 조정**이 **파라미터 효율적인 방식**으로 여러 작업에서 **경쟁력 있는 성능**을 발휘함을 실험적으로 입증하였다.

3.4 Parameter/Performance Trade-off

어댑터 크기는 **파라미터 효율성**을 결정하며, **어댑터 크기가 작을수록 더 적은 파라미터가 도입되지만 성능 저하의 가능성**이 있다.

→ `절충점(trade-off)`을 탐구하기 위해, 다양한 어댑터 크기를 실험하였으며, 두 가지 기준선과 비교

1. **BERTBASE의 상위 k개의 레이어만 미세 조정**하는 방식.
2. **레이어 정규화(layer normalization) 파라미터만 조정**하는 방식.

🍭 **GLUE 벤치마크에 대해 어댑터 조정의 성능과 파라미터 수를 비교한 결과**

- **BERTLARGE 전체 미세 조정(fine-tuning)**: 100%의 파라미터를 사용하며 GLUE 작업에서 최상의 성능(예: 80.4)을 달성한다.
- **어댑터 크기 8-256 범위**: 평균 1.3배의 파라미터만 추가되며, 전체 미세 조정에 근접한 80.0 점수를 기록한다.
- **고정된 어댑터 크기 64**: 파라미터는 1.2배만 추가되었으며, 평균 점수는 79.6으로 약간의 성능 저하를 보인다.

3.5 추가 분류 작업에 대한 실험

추가 텍스트 분류 작업에서도 어댑터가 **컴팩트하면서도 뛰어난 성능**을 발휘하는지 검증

- **AutoML 기준선**과 비교하여 BERT 모델이 **경쟁력 있는 성능**을 보였다.
- **전체 미세 조정**은 BERTBASE 파라미터의 **17배**를 필요로 했으며, **가변 미세 조정(variable fine-tuning)**은 평균적으로 네트워크의 **52%**만 학습하여 총 **9.9배**의 파라미터를 사용했다.
- **어댑터**는 작업당 **1.14%**의 새로운 파라미터만 추가하여, **1.19배**의 파라미터로 모든 작업을 해결하였다.

3.6 SQuAD 질문 응답 작업에서의 실험

어**분류 작업 외**의 다른 작업에서도 효과적인지 검증→**SQuAD v1.1**(Rajpurkar et al., 2018)

: 주어진 질문에 대해 Wikipedia 단락에서 `정답 범위(answer span`를 선택.

<결과>

- **어댑터 크기 64**(전체 파라미터의 2%)로 **최고 F1 점수 90.4%**를 기록했으며, 전체 미세 조정은 **90.7%**를 달성했다.
- **어댑터 크기 2**(전체 파라미터의 0.1%)로도 **F1 점수 89.9%**를 달성했다.

→ 어댑터는 **적은 수의 파라미터로도 경쟁력 있는 성능을 유**지할 수 있음을 확인할 수 있다.

3.7 분석 및 토론

우리는 어댑터의 영향을 분석하기 위해 **훈련된 어댑터의 일부를 제거**하고 성능 변화를 재평가하였다

> 🍭 **MNLI와 CoLA 데이터셋에서 실험을 수행한 결과**
>
> 어댑터가 각 레이어에 미치는 영향은 작지만, 전체적으로 **어댑터가 제거된 경우** 성능이 크게 저하됨을 확인했다.
>
> 특히, **상위 레이어**에서 어댑터의 영향이 더 컸음!
>
> → 이는 **하위 레이어**가 **공유되는 저수준 피처**를 추출하고, **상위 레이어**가 **작업별 특화된 피처**를 생성하는 역할을 한다는 관찰과 일치한다.

요약

- **어댑터 조정**은 **매우 파라미터 효율적인** 전이 학습 기법으로, **전체 미세 조정과 유사한 성능**을 유지하면서도 **적은 수의 파라미터**만 추가된다.
- **어댑터 크기**를 0.5~5%로 설정해도, **BERTLARGE**에서 **경쟁력 있는 성능**을 유지할 수 있으며, 대부분의 작업에서 **1% 이내**의 성능 차이를 보였다.

이 실험은 **어댑터 기반 조정**이 다양한 NLP 작업에서 **성능 저하 없이 파라미터 효율성**을 제공함을 입증한다.

02 | 논문 탐구

🚥 주제 1 : 기존 파인 튜닝과 비교하여 비교 표 만들기

항목	어댑터 튜닝 (Adapter Tuning)	전체 미세 조정(Full Fine-tuning)	가변 미세 조정 (Variable Fine-tuning)	특징 기반 전이 (Feature-based Transfer)
파라미터 효율성 (Param. Efficiency)	매우 효율적, 작업당 파라미터의 0.5%~5%만 추가	낮은 효율성, 각 작업에 대해 전체 모델 파라미터의 100%를 재학습해야 함	중간 정도 효율성, 상위 레이어(k개 레이어)만 재학습	중간 정도 효율성, 사전 학습된 임베딩을 사용하나 파라미터 튜닝은 없음
학습 복잡도 (Training Complexity)	단순함, 어댑터 레이어만 학습하며 기본 모델은 고정됨	매우 복잡함, 모델의 모든 레이어를 조정해야 함	중간 정도 복잡도, 특정 레이어만 조정	낮은 복잡도, 사전 학습된 임베딩을 다운스트림 모델에 입력함
새로운 작업에 대한 적용성	이전 레이어를 재학습하지 않고도 쉽게	새로운 작업에 대해 전체 모델을 재학습해야 하며, 이는 이전	새로운 작업에 대해 일부 레이어만 재학	사전 학습된 모델을 사용하지만, 새로운 작업에 맞게 재학습

(Applicability to New Tasks)	새로운 작업에 확장 가능	작업을 잊을 수 있는 문제(망각 문제)를 초래함	습하므로 전체 미세 조정보다 효율적	하지 않으면 적응성이 낮음
메모리 사용량 (Memory Usage)	낮은 메모리 사용, 소수의 파라미터만 추가되기 때문	높은 메모리 사용, 각 작업에 대해 전체 모델을 재학습해야 하므로 메모리 소모가 큼	중간 정도 메모리 사용, 일부 모델만 재학습	낮은 메모리 사용, 재학습이 없기 때문에 메모리 사용량이 적음
학습 안정성 (Training Stability)	매우 안정적, 초기화 시 거의 동일한 값으로 설정되므로 학습 불안정성이 적음	비교적 안정적이지만, 모델이 커질수록 학습 안정성이 떨어질 수 있음	전체 미세 조정보다 안정적이나, 일부 레이어는 성능 저하 가능	매우 안정적, 재학습이 없으므로 학습 불안정성이 없음
성능 (Performance)	전체 미세 조정과 비슷한 성능, 약 1% 이내 성능 차이	최첨단 성능 달성 가능하나, 높은 자원 소모	거의 전체 미세 조정과 유사한 성능을 발휘하지만, 파라미터는 더 적음	전체 미세 조정 또는 어댑터 튜닝보다는 낮은 성능을 보임
사용 사례 (Example Use Cases)	텍스트 분류, 추출형 QA (GLUE, SQuAD 등)	일반적인 NLP 작업 (예: 언어 모델링, 텍스트 분류)	높은 효율성과 중간 성능이 필요한 작업에 사용	텍스트 임베딩 (예: Word2Vec, GloVe, ELMo)
핵심 아키텍처 조정(Key Arch. Adjustments)	Transformer의 서브 레이어 사이에 병목 레이어를 추가하며, 거의 동일한 값으로 초기화	전체 모델을 재학습하며, 작업별 출력을 위해 상위 레이어 수정이 필요함	상위 k개의 레이어만 조정하고 하위 레이어는 고정됨	사전 학습된 임베딩을 새로운 레이어로 전달

🚥 주제 2 : 2.4 집중 탐구!

🍭 2.4 병목 아키텍처(bottleneck architecture) 설계

파라미터 수를 제한하기 위해 **병목 아키텍처**를 제안한다. 어댑터는 먼저 **d-차원 피처**를 **더 작은 차원 m**으로 투영한 뒤, 비선형성을 적용하고 다시 **d-차원**으로 투영한다.

- **추가되는 파라미터 수**는 **2md + d + m**이며, 여기서 **m ≪ d**로 설정해 각 작업에 추가되는 파라미터 수를 최소화한다.
- 실험적으로, 우리는 원래 모델의 **0.5%에서 8%** 정도의 파라미터만 추가하여 사용하였다.

병목 차원 m은 **성능**과 **파라미터 효율성** 사이에서 **절충점**을 제공하는 중요한 요소이다. 어댑터 모듈에는 **스킵 연결(skip-connection)**이 포함되어 있는데, 이는 투영 레이어가 **거의 0**에 가깝게 초기화될 경우, 모듈이 **거의 동일한 함수(approximate identity function)**로 작동하게 해준다.

어댑터 모듈과 함께 각 작업마다 **새로운 레이어 정규화 파라미터**도 학습된다. 이 기법은 **조건부 배치 정규화(conditional batch normalization)**, **FiLM**, **자기 조절(self-modulation)** 기법들과 유사하며, **각 레이어당 2d**의 파라미터만 추가된다. 하지만 **레이어 정규화 파라미터**만 학습하는 것은 충분한 성능을 보장하지 못하며, 이 부분은 3.4장에서 자세히 다룬다.

병목 아키텍처(bottleneck architecture)는 어댑터 모듈의 중요한 개념으로, 파라미터 효율성을 극대화하면서도 성능 저하를 최소화하기 위한 전략입니다. 이 개념을 좀 더 쉽게 이해할 수 있도록 단계별로 설명하겠습니다.

병목 아키텍처(bottleneck architecture)의 핵심 개념

1. **d-차원 피처(d-dimensional features)**:

- **d**는 Transformer 모델의 각 레이어에서 나오는 피처(특징)의 크기입니다. 예를 들어, BERT 모델에서 **d**는 일반적으로 768 또는 1024와 같은 큰 값입니다.
  - 이 **d-차원 피처**는 모델이 각 레이어에서 입력받은 데이터를 처리해 추출하는 특징들의 집합이라고 생각하면 됩니다.

2. **병목(bottleneck) 개념**:

- 병목 구조는, **d-차원 피처**를 **더 작은 차원인 m으로 줄이는 것**을 의미합니다. 여기서 **m은 d보다 훨씬 작은 값**입니다. 이렇게 피처의 차원을 줄이면, 학습해야 할 파라미터 수가 크게 줄어듭니다.

- 줄어든 피처는 비선형성을 적용한 후 다시 원래 크기인 **d-차원**으로 확장됩니다. 이 과정을 통해 파라미터 효율성을 높일 수 있습니다.

3. **왜 파라미터 수가 줄어드는가?**

- **병목 아키텍처**는 **2md + d + m**개의 파라미터를 사용합니다. **m이 d보다 훨씬 작은 값**으로 설정되므로, 추가되는 파라미터 수가 매우 적습니다.
- 실험에서 우리는 **m**을 조절하여, **0.5%에서 8%** 정도의 파라미터만 추가했을 때도 좋은 성능을 낼 수 있음을 확인하였습니다. 즉, 적은 수의 파라미터로도 원래 모델과 유사한 성능을 낼 수 있다는 것입니다.

4. **스킵 연결(skip connection)**:

- 어댑터 모듈에는 **스킵 연결**이 포함되어 있습니다. 이 연결은 피처가 변환되는 과정에서 원래의 입력 값을 그대로 다음 단계로 넘기는 역할을 합니다.
- 만약 **투영(projection) 레이어의 파라미터가 거의 0으로 초기화**되면, 어댑터 모듈은 **거의 동일한 함수(approximate identity function)**처럼 작동하여 네트워크의 원래 기능에 영향을 미치지 않게 됩니다. 이로 인해 모델의 **학습 안정성**이 높아집니다.

5. **레이어 정규화 파라미터(layer normalization parameters)**:

- 병목 구조 외에도, 각 작업마다 **새로운 레이어 정규화 파라미터**를 학습합니다. 이 기법은 **파라미터 효율적인 네트워크 적응(parameter-efficient adaptation)**을 가능하게 하며, 각 레이어당 **2d**의 파라미터만 필요합니다.

하지만 **레이어 정규화 파라미터만** 학습하는 것은 좋은 성능을 내기에는 부족하기 때문에, 이 기법만으로는 충분하지 않음을 3.4장에서 다룹니다.

6. 이미지 분석

PDF 내에서 참고할 수 있는 **어댑터 모듈**과 **병목 아키텍처** 관련 이미지는 `Figure 2` (Transformer에 어댑터 아키텍처를 적용한 다이어그램)입니다. 이 이미지는 어댑터가 Transformer 네트워크에 어떻게 통합되는지, 그리고 병목 구조가 각 서브 레이어 후에 어떻게 적용되는지를 시각적으로 보여줍니다. **Figure 2**는 병목 아키텍처의 위치와 역할을 명확히 이해하는 데 도움이 될 것입니다.

주제 3 : adaptor 그래서 어떻게 코드에서 구현하는건데?

`Adapter Tuning for NLP` 논문과 깃허브 프로젝트의 코드 구조를 연결하여 **Adapter Tuning** 개념을 좀 더 자세히 설명한다. 이 방식은 전체 모델을 미세 조정하는 대신 **소수의 파라미터만 학습**하는 방식으로, 대규모 사전 학습된 모델을 효율적으로 사용할 수 있도록 한다.

1. Adapter Tuning 개요

Adapter Tuning은 사전 학습된 모델(예: BERT)을 사용해 **전이 학습(Transfer Learning)**을 수행할 때 모든 파라미터를 미세 조정하지 않고, **특정 레이어**에 작은 Adapter 모듈을 추가해 그 부분만 학습하는 방식이다. 이를 통해 학습 시간과 메모리 사용량을 줄이면서도, 새로운 작업(task)에 맞는 최적의 성능을 유지할 수 있다.

왜 Adapter Tuning을 사용하는가?

- **효율성**: 사전 학습된 BERT 모델과 같은 대규모 모델을 모든 작업마다 새로 미세 조정하면 수많은 파라미터를 저장해야 하고, 학습 시간이 길어지며, 메모리 사용량도 크게 증가한다. 반면 Adapter Tuning은 필요한 파라미터 수가 적어 훨씬 가볍다.
- **성능 유지**: 전체 모델의 파라미터를 미세 조정하지 않더라도, Adapter 모듈만 적절히 조정하면 성능이 거의 떨어지지 않는다. 이는 학습해야 할 파라미터가 적음에도 불구하고, 원래의 사전 학습된 모델을 유지하면서도 새로운 작업에 쉽게 적응할 수 있다는 것을 의미한다.

2. 논문과 코드간의 연관성

(a) Transformer 레이어 사이에 Adapter 삽입

논문에서는 BERT와 같은 Transformer 기반 모델에서, **각 Transformer 레이어 사이**에 작은 **Adapter 모듈**을 삽입하는 것을 제안한다. 이 Adapter는 **multi-head attention**과 **feed-forward layer** 사이에 위치하며, 아래와 같은 병목 구조를 따른다:

- 먼저, 입력을 **차원을 줄여** (down-projection) 작은 크기의 벡터로 만든다.
- 그 벡터를 다시 원래 차원으로 **확장**(up-projection)하여 출력한다.

코드 연결:

- 이 논문에서 언급된 병목 구조는 깃허브 프로젝트의 `modeling.py` 파일에 반영되었을 가능성이 크다. BERT 모델의 Transformer 레이어에서, **attention**과 **feed-forward** 사이에 Adapter를 삽입하고, 해당 모듈만 학습하도록 구성할 수 있다. 이를 통해 기존 BERT 모델의 전체 파라미터는 그대로 유지되고, Adapter 모듈에 추가된 소수의 파라미터만 업데이트되는 구조가 된다.

```
class BertModel(object):
    # Bert 모델 생성자 내에서 Adapter 모듈이 각 레이어 사이에 추가
```

```
되었을 것으로 추정됨
    def __init__(self, config, is_training, input_ids, in
put_mask=None, token_type_ids=None, use_one_hot_embedding
s=False, scope=None):
        ...
        # Transformer 레이어 호출부에서 Adapter 모듈 삽입 가능
성
        with tf.variable_scope("encoder"):
            self.all_encoder_layers = transformer_model(
                input_tensor=self.embedding_output,
                attention_mask=attention_mask,
                hidden_size=config.hidden_size,
                num_hidden_layers=config.num_hidden_layer
s
            )
```

(b) 소수의 파라미터만 학습

논문에서 강조하는 또 다른 핵심 개념은, 사전 학습된 모델의 **모든 파라미터를 학습하지 않고** Adapter에 해당하는 소수의 파라미터만 학습하는 것이다. 이를 통해 **파라미터 효율성**을 극대화하고, 전체 모델을 미세 조정하는 것과 비슷한 성능을 유지할 수 있다.

코드 연결:

- `optimization.py` 파일에서 **AdamWeightDecayOptimizer**를 사용하여 학습 파라미터를 관리한다. 이때 Adapter 모듈에 해당하는 부분만 업데이트할 수 있도록 설정한다. 즉, Adapter에 속하지 않는 파라미터는 학습에서 제외하고, Adapter 관련 파라미터만 업데이트하는 방식으로 동작한다.

```
# AdamWeightDecayOptimizer 내에서 학습 가능한 파라미터만 업데이
트
optimizer = AdamWeightDecayOptimizer(
    learning_rate=learning_rate,
    weight_decay_rate=0.01,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-6,
    exclude_from_weight_decay=["LayerNorm", "bias"]
)

grads = tf.gradients(loss, tvars)
# Adapter 모듈에 해당하는 파라미터만 업데이트
```

```
train_op = optimizer.apply_gradients(zip(grads, tvars), g
lobal_step=global_step)
```

(c) 파라미터 효율성

논문에서 소개된 **효율성**의 개념은 Adapter 모듈이 기존 모델의 파라미터 수에 비해 매우 적은 양의 파라미터만을 추가로 학습시킨다는 것이다. 이를 통해 전체 모델을 미세 조정하는 것에 비해 훨씬 적은 자원으로 학습을 진행할 수 있다.

코드 연결:

- Adapter가 BERT 모델의 **attention** 및 **feed-forward** 레이어 사이에 삽입된 부분에서, 병목 구조를 사용하여 적은 수의 파라미터만 업데이트하는 점이 이 효율성을 반영한다. `modeling.py`나 `run_classifier.py`에서 학습 파라미터가 최소화된 구조를 찾을 수 있다.

3. 논문과 코드의 핵심 연결점 요약

- **Transformer 레이어 사이에 Adapter 삽입**: Transformer의 **각 계층 사이**에 Adapter 모듈이 추가되며, 이 Adapter는 병목 구조를 따른다.
- **소수의 파라미터만 학습**: BERT 모델의 기존 파라미터는 그대로 두고, **Adapter에 해당하는 파라미터만 업데이트**하도록 설계되었다.
- **파라미터 효율성**: 전체 모델을 미세 조정하지 않고, 적은 수의 파라미터만 학습하여도 비슷한 성능을 낼 수 있다.

결론적으로, 논문에서 제안하는 **Adapter Tuning** 전략이 깃허브 코드의 BERT 모델에 반영된 방식은, 기존 BERT 모델의 큰 파라미터 공간을 그대로 유지하면서도 **효율적으로 미세 조정**할 수 있도록 Transformer의 각 레이어 사이에 **Adapter 모듈을 삽입**하고, 이를 학습하는 구조로 구현되었음을 알 수 있다.

코드에서 소수의 파라미터 학습을 확인하는 방법

1. **모델의 학습 파라미터 설정 부분을 확인**:
   - `trainable_variables` 함수나 **파라미터 선택 로직**에서 어떤 파라미터가 학습 대상인지 확인할 수 있다.
   - Adapter Tuning의 경우 전체 모델의 모든 파라미터를 학습하지 않고, **일부 계층(예: Adapter 관련 파라미터)**만 선택적으로 학습한다.

   예를 들어, TensorFlow에서 학습 가능한 파라미터 목록을 확인할 때는 `tf.trainable_variables()` 함수가 사용되며, Adapter에 관련된 파라미터만 업데이트할 수 있도록 설정되어 있어야 한다.

```
# 모델의 trainable variables를 설정할 때 일부 파라미터만 선택하는 방식
tvars = [var for var in tf.trainable_variables() if 'adapter' in var.name]
```

2. **Adam 옵티마이저에서 학습되지 않을 파라미터를 제외**:
   - `AdamWeightDecayOptimizer`는 특정 파라미터에 가중치 감쇠(weight decay)를 적용하지 않거나 학습에서 제외할 수 있는 설정을 제공한다. 이를 통해 Adapter와 관련된 일부 파라미터만 학습하고, 다른 파라미터는 고정할 수 있다.

   예를 들어, `exclude_from_weight_decay` 리스트에 **LayerNorm**이나 **bias**와 같은 기존 파라미터들을 추가하고, Adapter 관련 파라미터만 학습하게 만들 수 있다.

```
optimizer = AdamWeightDecayOptimizer(
    learning_rate=learning_rate,
    weight_decay_rate=0.01,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-6,
    exclude_from_weight_decay=["LayerNorm", "bias"]  # Adapter 외의 파라미터 제외
)
```

   이 코드는 LayerNorm이나 bias와 같은 파라미터들은 학습되지 않도록 고정하고, 새로운 Adapter 모듈에 속하는 파라미터들만 학습하게끔 설정된다.

3. **Optimizer 적용 시 파라미터 범위를 좁혀서 학습**:
   - 학습할 때도 Adapter 파라미터들만 학습되도록 그래디언트 계산 시 특정 파라미터들만 포함하는지 확인할 수 있다.
   - 이때 `grads_and_vars` 리스트에 포함된 변수들이 바로 학습할 대상 파라미터들이며, Adapter 관련 파라미터만 포함되었는지 확인하는 것이 중요하다.

```
grads = tf.gradients(loss, tvars)  # tvars에는 Adapter 관련 파라미터만 포함
train_op = optimizer.apply_gradients(zip(grads, tvars), global_step=global_step)
```

   여기서 `tvars` 리스트는 학습할 파라미터들이며, Adapter 모듈에 속한 파라미터들만 포함하도록 필터링되어 있다면, 논문에서 제시한 "소수의 파라미터만 학습"하는 방법이 코드에 구현된 것을 확인할 수 있다.

코드의 학습 파라미터 제어를 확인할 수 있는 부분들:

1. *모델 클래스(`modeling.py`)**에서 **Adapter 관련 파라미터**가 어떻게 정의되고, 그 파라미터들만 선택적으로 학습하는지 확인할 수 있다.
2. *Optimizer(`optimization.py`)**에서 학습 파라미터를 제한하고, 일부 파라미터를 학습하지 않도록 제외하는 부분을 통해 Adapter Tuning이 구현되었는지 알 수 있다.
3. *학습 함수(`run_classifier.py`)**에서 실제로 그래디언트를 계산할 때, 어떤 파라미터들이 업데이트되는지를 확인할 수 있다.

 주제 4 : 기존 bert(논문에서 차용한 bert)와 adapter 모델 코드 비교 분석

03 | 실습 : 코드 분석

https://github.com/google-research/adapter-bert

https://github.com/google-research/adapter-bert

`tokenization.py`

```
# coding=utf-8
# Copyright 2018 The Google AI Language Team Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, softwa
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
# See the License for the specific language governing permissions a
# limitations under the License.
"""Tokenization classes."""

from __future__ import absolute_import
from __future__ import division
```

```
from __future__ import print_function

import collections
import re
import unicodedata
import six
import tensorflow as tf


def validate_case_matches_checkpoint(do_lower_case, init_checkpoint
  """Checks whether the casing config is consistent with the checkp

  # The casing has to be passed in by the user and there is no expl
  # as to whether it matches the checkpoint. The casing information
  # should have been stored in the bert_config.json file, but it's
  # we have to heuristically detect it to validate.

  if not init_checkpoint:
    return

  m = re.match("^.*?([A-Za-z0-9_-]+)/bert_model.ckpt", init_checkpo
  if m is None:
    return

  model_name = m.group(1)

  lower_models = [
      "uncased_L-24_H-1024_A-16", "uncased_L-12_H-768_A-12",
      "multilingual_L-12_H-768_A-12", "chinese_L-12_H-768_A-12"
  ]

  cased_models = [
      "cased_L-12_H-768_A-12", "cased_L-24_H-1024_A-16",
      "multi_cased_L-12_H-768_A-12"
  ]

  is_bad_config = False
  if model_name in lower_models and not do_lower_case:
    is_bad_config = True
    actual_flag = "False"
    case_name = "lowercased"
    opposite_flag = "True"
```

```
  if model_name in cased_models and do_lower_case:
    is_bad_config = True
    actual_flag = "True"
    case_name = "cased"
    opposite_flag = "False"

  if is_bad_config:
    raise ValueError(
        "You passed in `--do_lower_case=%s` with `--init_checkpoint
        "However, `%s` seems to be a %s model, so you "
        "should pass in `--do_lower_case=%s` so that the fine-tunin
        "how the model was pre-training. If this error is wrong, pl
        "just comment out this check." % (actual_flag, init_checkpo
                                          model_name, case_name, op
```

```
def convert_to_unicode(text):
  """Converts `text` to Unicode (if it's not already), assuming utf
  if six.PY3:
    if isinstance(text, str):
      return text
    elif isinstance(text, bytes):
      return text.decode("utf-8", "ignore")
    else:
      raise ValueError("Unsupported string type: %s" % (type(text)))
  elif six.PY2:
    if isinstance(text, str):
      return text.decode("utf-8", "ignore")
    elif isinstance(text, unicode):
      return text
    else:
      raise ValueError("Unsupported string type: %s" % (type(text)))
  else:
    raise ValueError("Not running on Python2 or Python 3?")


def printable_text(text):
  """Returns text encoded in a way suitable for print or `tf.loggin

  # These functions want `str` for both Python2 and Python3, but in
```

```
  # it's a Unicode string and in the other it's a byte string.
  if six.PY3:
    if isinstance(text, str):
      return text
    elif isinstance(text, bytes):
      return text.decode("utf-8", "ignore")
    else:
      raise ValueError("Unsupported string type: %s" % (type(text)))
  elif six.PY2:
    if isinstance(text, str):
      return text
    elif isinstance(text, unicode):
      return text.encode("utf-8")
    else:
      raise ValueError("Unsupported string type: %s" % (type(text)))
  else:
    raise ValueError("Not running on Python2 or Python 3?")


def load_vocab(vocab_file):
  """Loads a vocabulary file into a dictionary."""
  vocab = collections.OrderedDict()
  index = 0
  with tf.gfile.GFile(vocab_file, "r") as reader:
    while True:
      token = convert_to_unicode(reader.readline())
      if not token:
        break
      token = token.strip()
      vocab[token] = index
      index += 1
  return vocab


def convert_by_vocab(vocab, items):
  """Converts a sequence of [tokens|ids] using the vocab."""
  output = []
  for item in items:
    output.append(vocab[item])
  return output
```

```
def convert_tokens_to_ids(vocab, tokens):
  return convert_by_vocab(vocab, tokens)


def convert_ids_to_tokens(inv_vocab, ids):
  return convert_by_vocab(inv_vocab, ids)


def whitespace_tokenize(text):
  """Runs basic whitespace cleaning and splitting on a piece of tex
  text = text.strip()
  if not text:
    return []
  tokens = text.split()
  return tokens


class FullTokenizer(object):
  """Runs end-to-end tokenziation."""

  def __init__(self, vocab_file, do_lower_case=True):
    self.vocab = load_vocab(vocab_file)
    self.inv_vocab = {v: k for k, v in self.vocab.items()}
    self.basic_tokenizer = BasicTokenizer(do_lower_case=do_lower_ca
    self.wordpiece_tokenizer = WordpieceTokenizer(vocab=self.vocab)

  def tokenize(self, text):
    split_tokens = []
    for token in self.basic_tokenizer.tokenize(text):
      for sub_token in self.wordpiece_tokenizer.tokenize(token):
        split_tokens.append(sub_token)

    return split_tokens

  def convert_tokens_to_ids(self, tokens):
    return convert_by_vocab(self.vocab, tokens)

  def convert_ids_to_tokens(self, ids):
    return convert_by_vocab(self.inv_vocab, ids)


class BasicTokenizer(object):
```

```
  """Runs basic tokenization (punctuation splitting, lower casing,

  def __init__(self, do_lower_case=True):
    """Constructs a BasicTokenizer.

    Args:
      do_lower_case: Whether to lower case the input.
    """
    self.do_lower_case = do_lower_case

  def tokenize(self, text):
    """Tokenizes a piece of text."""
    text = convert_to_unicode(text)
    text = self._clean_text(text)

    # This was added on November 1st, 2018 for the multilingual and
    # models. This is also applied to the English models now, but i
    # matter since the English models were not trained on any Chine
    # and generally don't have any Chinese data in them (there are
    # characters in the vocabulary because Wikipedia does have some
    # words in the English Wikipedia.).
    text = self._tokenize_chinese_chars(text)

    orig_tokens = whitespace_tokenize(text)
    split_tokens = []
    for token in orig_tokens:
      if self.do_lower_case:
        token = token.lower()
        token = self._run_strip_accents(token)
      split_tokens.extend(self._run_split_on_punc(token))

    output_tokens = whitespace_tokenize(" ".join(split_tokens))
    return output_tokens

  def _run_strip_accents(self, text):
    """Strips accents from a piece of text."""
    text = unicodedata.normalize("NFD", text)
    output = []
    for char in text:
      cat = unicodedata.category(char)
      if cat == "Mn":
        continue
```

```
      output.append(char)
    return "".join(output)

  def _run_split_on_punc(self, text):
    """Splits punctuation on a piece of text."""
    chars = list(text)
    i = 0
    start_new_word = True
    output = []
    while i < len(chars):
      char = chars[i]
      if _is_punctuation(char):
        output.append([char])
        start_new_word = True
      else:
        if start_new_word:
          output.append([])
        start_new_word = False
        output[-1].append(char)
      i += 1

    return ["".join(x) for x in output]

  def _tokenize_chinese_chars(self, text):
    """Adds whitespace around any CJK character."""
    output = []
    for char in text:
      cp = ord(char)
      if self._is_chinese_char(cp):
        output.append(" ")
        output.append(char)
        output.append(" ")
      else:
        output.append(char)
    return "".join(output)

  def _is_chinese_char(self, cp):
    """Checks whether CP is the codepoint of a CJK character."""
    # This defines a "chinese character" as anything in the CJK Uni
    #   https://en.wikipedia.org/wiki/CJK_Unified_Ideographs_(Unico
    #
    # Note that the CJK Unicode block is NOT all Japanese and Korea
```

```
    # despite its name. The modern Korean Hangul alphabet is a dif
    # as is Japanese Hiragana and Katakana. Those alphabets are us
    # space-separated words, so they are not treated specially and
    # like the all of the other languages.
    if ((cp >= 0x4E00 and cp <= 0x9FFF) or  #
        (cp >= 0x3400 and cp <= 0x4DBF) or  #
        (cp >= 0x20000 and cp <= 0x2A6DF) or  #
        (cp >= 0x2A700 and cp <= 0x2B73F) or  #
        (cp >= 0x2B740 and cp <= 0x2B81F) or  #
        (cp >= 0x2B820 and cp <= 0x2CEAF) or
        (cp >= 0xF900 and cp <= 0xFAFF) or  #
        (cp >= 0x2F800 and cp <= 0x2FA1F)):  #
      return True

    return False

  def _clean_text(self, text):
    """Performs invalid character removal and whitespace cleanup or
    output = []
    for char in text:
      cp = ord(char)
      if cp == 0 or cp == 0xfffd or _is_control(char):
        continue
      if _is_whitespace(char):
        output.append(" ")
      else:
        output.append(char)
    return "".join(output)


class WordpieceTokenizer(object):
  """Runs WordPiece tokenziation."""

  def __init__(self, vocab, unk_token="[UNK]", max_input_chars_per_
    self.vocab = vocab
    self.unk_token = unk_token
    self.max_input_chars_per_word = max_input_chars_per_word

  def tokenize(self, text):
    """Tokenizes a piece of text into its word pieces.

    This uses a greedy longest-match-first algorithm to perform tok
```

```
    using the given vocabulary.

    For example:
      input = "unaffable"
      output = ["un", "##aff", "##able"]

    Args:
      text: A single token or whitespace separated tokens. This sho
        already been passed through `BasicTokenizer.

    Returns:
      A list of wordpiece tokens.
    """

    text = convert_to_unicode(text)

    output_tokens = []
    for token in whitespace_tokenize(text):
      chars = list(token)
      if len(chars) > self.max_input_chars_per_word:
        output_tokens.append(self.unk_token)
        continue

      is_bad = False
      start = 0
      sub_tokens = []
      while start < len(chars):
        end = len(chars)
        cur_substr = None
        while start < end:
          substr = "".join(chars[start:end])
          if start > 0:
            substr = "##" + substr
          if substr in self.vocab:
            cur_substr = substr
            break
          end -= 1
        if cur_substr is None:
          is_bad = True
          break
        sub_tokens.append(cur_substr)
        start = end
```

```
      if is_bad:
        output_tokens.append(self.unk_token)
      else:
        output_tokens.extend(sub_tokens)
    return output_tokens


def _is_whitespace(char):
  """Checks whether `chars` is a whitespace character."""
  # \t, \n, and \r are technically contorl characters but we treat
  # as whitespace since they are generally considered as such.
  if char == " " or char == "\t" or char == "\n" or char == "\r":
    return True
  cat = unicodedata.category(char)
  if cat == "Zs":
    return True
  return False


def _is_control(char):
  """Checks whether `chars` is a control character."""
  # These are technically control characters but we count them as w
  # characters.
  if char == "\t" or char == "\n" or char == "\r":
    return False
  cat = unicodedata.category(char)
  if cat in ("Cc", "Cf"):
    return True
  return False


def _is_punctuation(char):
  """Checks whether `chars` is a punctuation character."""
  cp = ord(char)
  # We treat all non-letter/number ASCII as punctuation.
  # Characters such as "^", "$", and "`" are not in the Unicode
  # Punctuation class but we treat them as punctuation anyways, for
  # consistency.
  if ((cp >= 33 and cp <= 47) or (cp >= 58 and cp <= 64) or
      (cp >= 91 and cp <= 96) or (cp >= 123 and cp <= 126)):
    return True
```

```
  cat = unicodedata.category(char)
  if cat.startswith("P"):
    return True
  return False
```

tokenization_test.py

```
# coding=utf-8
# Copyright 2018 The Google AI Language Team Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, softwa
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
# See the License for the specific language governing permissions a
# limitations under the License.
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import tempfile
import tokenization
import six
import tensorflow as tf


class TokenizationTest(tf.test.TestCase):

  def test_full_tokenizer(self):
    vocab_tokens = [
        "[UNK]", "[CLS]", "[SEP]", "want", "##want", "##ed", "wa",
        "##ing", ","
    ]
    with tempfile.NamedTemporaryFile(delete=False) as vocab_writer:
      if six.PY2:
        vocab_writer.write("".join([x + "\n" for x in vocab_tokens]
```

```
      else:
        vocab_writer.write("".join(
            [x + "\n" for x in vocab_tokens]).encode("utf-8"))

        vocab_file = vocab_writer.name

      tokenizer = tokenization.FullTokenizer(vocab_file)
      os.unlink(vocab_file)

      tokens = tokenizer.tokenize(u"UNwant\u00E9d,running")
      self.assertAllEqual(tokens, ["un", "##want", "##ed", ",", "runn

      self.assertAllEqual(
          tokenizer.convert_tokens_to_ids(tokens), [7, 4, 5, 10, 8, 9

    def test_chinese(self):
      tokenizer = tokenization.BasicTokenizer()

      self.assertAllEqual(
          tokenizer.tokenize(u"ah\u535A\u63A8zz"),
          [u"ah", u"\u535A", u"\u63A8", u"zz"])

    def test_basic_tokenizer_lower(self):
      tokenizer = tokenization.BasicTokenizer(do_lower_case=True)

      self.assertAllEqual(
          tokenizer.tokenize(u" \tHeLLo!how  \n Are yoU?  "),
          ["hello", "!", "how", "are", "you", "?"])
      self.assertAllEqual(tokenizer.tokenize(u"H\u00E9llo"), ["hello"

    def test_basic_tokenizer_no_lower(self):
      tokenizer = tokenization.BasicTokenizer(do_lower_case=False)

      self.assertAllEqual(
          tokenizer.tokenize(u" \tHeLLo!how  \n Are yoU?  "),
          ["HeLLo", "!", "how", "Are", "yoU", "?"])

    def test_wordpiece_tokenizer(self):
      vocab_tokens = [
          "[UNK]", "[CLS]", "[SEP]", "want", "##want", "##ed", "wa",
          "##ing"
      ]
```

```
    vocab = {}
    for (i, token) in enumerate(vocab_tokens):
      vocab[token] = i
    tokenizer = tokenization.WordpieceTokenizer(vocab=vocab)

    self.assertAllEqual(tokenizer.tokenize(""), [])

    self.assertAllEqual(
        tokenizer.tokenize("unwanted running"),
        ["un", "##want", "##ed", "runn", "##ing"])

    self.assertAllEqual(
        tokenizer.tokenize("unwantedX running"), ["[UNK]", "runn",

  def test_convert_tokens_to_ids(self):
    vocab_tokens = [
        "[UNK]", "[CLS]", "[SEP]", "want", "##want", "##ed", "wa",
        "##ing"
    ]

    vocab = {}
    for (i, token) in enumerate(vocab_tokens):
      vocab[token] = i

    self.assertAllEqual(
        tokenization.convert_tokens_to_ids(
            vocab, ["un", "##want", "##ed", "runn", "##ing"]), [7,

  def test_is_whitespace(self):
    self.assertTrue(tokenization._is_whitespace(u" "))
    self.assertTrue(tokenization._is_whitespace(u"\t"))
    self.assertTrue(tokenization._is_whitespace(u"\r"))
    self.assertTrue(tokenization._is_whitespace(u"\n"))
    self.assertTrue(tokenization._is_whitespace(u"\u00A0"))

    self.assertFalse(tokenization._is_whitespace(u"A"))
    self.assertFalse(tokenization._is_whitespace(u"-"))

  def test_is_control(self):
    self.assertTrue(tokenization._is_control(u"\u0005"))
```

```
    self.assertFalse(tokenization._is_control(u"A"))
    self.assertFalse(tokenization._is_control(u" "))
    self.assertFalse(tokenization._is_control(u"\t"))
    self.assertFalse(tokenization._is_control(u"\r"))
    self.assertFalse(tokenization._is_control(u"\U0001F4A9"))

  def test_is_punctuation(self):
    self.assertTrue(tokenization._is_punctuation(u"-"))
    self.assertTrue(tokenization._is_punctuation(u"$"))
    self.assertTrue(tokenization._is_punctuation(u"`"))
    self.assertTrue(tokenization._is_punctuation(u"."))

    self.assertFalse(tokenization._is_punctuation(u"A"))
    self.assertFalse(tokenization._is_punctuation(u" "))


if __name__ == "__main__":
  tf.test.main()
```

modeling.py

```
# coding=utf-8
# Copyright 2018 The Google AI Language Team Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, softwa
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
# See the License for the specific language governing permissions a
# limitations under the License.
"""The main BERT model and related functions."""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import collections
```

```
import copy
import json
import math
import re
import numpy as np
import six
import tensorflow as tf


class BertConfig(object):
  """Configuration for `BertModel`."""

  def __init__(self,
               vocab_size,
               hidden_size=768,
               num_hidden_layers=12,
               num_attention_heads=12,
               intermediate_size=3072,
               hidden_act="gelu",
               hidden_dropout_prob=0.1,
               attention_probs_dropout_prob=0.1,
               max_position_embeddings=512,
               type_vocab_size=16,
               initializer_range=0.02):
    """Constructs BertConfig.

    Args:
      vocab_size: Vocabulary size of `inputs_ids` in `BertModel`.
      hidden_size: Size of the encoder layers and the pooler layer.
      num_hidden_layers: Number of hidden layers in the Transformer
      num_attention_heads: Number of attention heads for each atten
        the Transformer encoder.
      intermediate_size: The size of the "intermediate" (i.e., feed
        layer in the Transformer encoder.
      hidden_act: The non-linear activation function (function or s
        encoder and pooler.
      hidden_dropout_prob: The dropout probability for all fully co
        layers in the embeddings, encoder, and pooler.
      attention_probs_dropout_prob: The dropout ratio for the atten
        probabilities.
      max_position_embeddings: The maximum sequence length that thi
        ever be used with. Typically set this to something large ju
```

```
        (e.g., 512 or 1024 or 2048).
      type_vocab_size: The vocabulary size of the `token_type_ids`
        `BertModel`.
      initializer_range: The stdev of the truncated_normal_initiali
        initializing all weight matrices.
    """
    self.vocab_size = vocab_size
    self.hidden_size = hidden_size
    self.num_hidden_layers = num_hidden_layers
    self.num_attention_heads = num_attention_heads
    self.hidden_act = hidden_act
    self.intermediate_size = intermediate_size
    self.hidden_dropout_prob = hidden_dropout_prob
    self.attention_probs_dropout_prob = attention_probs_dropout_pro
    self.max_position_embeddings = max_position_embeddings
    self.type_vocab_size = type_vocab_size
    self.initializer_range = initializer_range

  @classmethod
  def from_dict(cls, json_object):
    """Constructs a `BertConfig` from a Python dictionary of parame
    config = BertConfig(vocab_size=None)
    for (key, value) in six.iteritems(json_object):
      config.__dict__[key] = value
    return config

  @classmethod
  def from_json_file(cls, json_file):
    """Constructs a `BertConfig` from a json file of parameters."""
    with tf.gfile.GFile(json_file, "r") as reader:
      text = reader.read()
    return cls.from_dict(json.loads(text))

  def to_dict(self):
    """Serializes this instance to a Python dictionary."""
    output = copy.deepcopy(self.__dict__)
    return output

  def to_json_string(self):
    """Serializes this instance to a JSON string."""
    return json.dumps(self.to_dict(), indent=2, sort_keys=True) + "
```

```
class BertModel(object):
  """BERT model ("Bidirectional Encoder Representations from Transf

  Example usage:

  ```python
  # Already been converted into WordPiece token ids
  input_ids = tf.constant([[31, 51, 99], [15, 5, 0]])
  input_mask = tf.constant([[1, 1, 1], [1, 1, 0]])
  token_type_ids = tf.constant([[0, 0, 1], [0, 2, 0]])

  config = modeling.BertConfig(vocab_size=32000, hidden_size=512,
    num_hidden_layers=8, num_attention_heads=6, intermediate_size=1

  model = modeling.BertModel(config=config, is_training=True,
    input_ids=input_ids, input_mask=input_mask, token_type_ids=toke

  label_embeddings = tf.get_variable(...)
  pooled_output = model.get_pooled_output()
  logits = tf.matmul(pooled_output, label_embeddings)
  ...
  ```
  """

  def __init__(self,
               config,
               is_training,
               input_ids,
               input_mask=None,
               token_type_ids=None,
               use_one_hot_embeddings=False,
               scope=None,
               adapter_fn="feedforward_adapter"):
    """Constructor for BertModel.

    Args:
      config: `BertConfig` instance.
      is_training: bool. true for training model, false for eval mo
        whether dropout will be applied.
      input_ids: int32 Tensor of shape [batch_size, seq_length].
      input_mask: (optional) int32 Tensor of shape [batch_size, seq
```

```
    token_type_ids: (optional) int32 Tensor of shape [batch_size,
    use_one_hot_embeddings: (optional) bool. Whether to use one-h
      embeddings or tf.embedding_lookup() for the word embeddings
    scope: (optional) variable scope. Defaults to "bert".
    adapter_fn: (optional) string identifying trainable adapter t
      as input a Tensor and returns a Tensor of the same shape.

  Raises:
    ValueError: The config is invalid or one of the input tensor
      is invalid.
  """
  config = copy.deepcopy(config)
  if not is_training:
    config.hidden_dropout_prob = 0.0
    config.attention_probs_dropout_prob = 0.0

  input_shape = get_shape_list(input_ids, expected_rank=2)
  batch_size = input_shape[0]
  seq_length = input_shape[1]

  if input_mask is None:
    input_mask = tf.ones(shape=[batch_size, seq_length], dtype=tf

  if token_type_ids is None:
    token_type_ids = tf.zeros(shape=[batch_size, seq_length], dty

  with tf.variable_scope(scope, default_name="bert"):
    with tf.variable_scope("embeddings"):
      # Perform embedding lookup on the word ids.
      (self.embedding_output, self.embedding_table) = embedding_l
          input_ids=input_ids,
          vocab_size=config.vocab_size,
          embedding_size=config.hidden_size,
          initializer_range=config.initializer_range,
          word_embedding_name="word_embeddings",
          use_one_hot_embeddings=use_one_hot_embeddings)

      # Add positional embeddings and token type embeddings, then
      # normalize and perform dropout.
      self.embedding_output = embedding_postprocessor(
          input_tensor=self.embedding_output,
          use_token_type=True,
```

```
            token_type_ids=token_type_ids,
            token_type_vocab_size=config.type_vocab_size,
            token_type_embedding_name="token_type_embeddings",
            use_position_embeddings=True,
            position_embedding_name="position_embeddings",
            initializer_range=config.initializer_range,
            max_position_embeddings=config.max_position_embeddings,
            dropout_prob=config.hidden_dropout_prob)

        with tf.variable_scope("encoder"):
          # This converts a 2D mask of shape [batch_size, seq_length]
          # mask of shape [batch_size, seq_length, seq_length] which
          # for the attention scores.
          attention_mask = create_attention_mask_from_input_mask(
              input_ids, input_mask)

          # Run the stacked transformer.
          # `sequence_output` shape = [batch_size, seq_length, hidden
          self.all_encoder_layers = transformer_model(
              input_tensor=self.embedding_output,
              attention_mask=attention_mask,
              hidden_size=config.hidden_size,
              num_hidden_layers=config.num_hidden_layers,
              num_attention_heads=config.num_attention_heads,
              intermediate_size=config.intermediate_size,
              intermediate_act_fn=get_activation(config.hidden_act),
              hidden_dropout_prob=config.hidden_dropout_prob,
              attention_probs_dropout_prob=config.attention_probs_dro
              initializer_range=config.initializer_range,
              do_return_all_layers=True,
              adapter_fn=get_adapter(adapter_fn))

        self.sequence_output = self.all_encoder_layers[-1]
        # The "pooler" converts the encoded sequence tensor of shape
        # [batch_size, seq_length, hidden_size] to a tensor of shape
        # [batch_size, hidden_size]. This is necessary for segment-le
        # (or segment-pair-level) classification tasks where we need
        # dimensional representation of the segment.
        with tf.variable_scope("pooler"):
          # We "pool" the model by simply taking the hidden state cor
          # to the first token. We assume that this has been pre-trai
          first_token_tensor = tf.squeeze(self.sequence_output[:, 0:1
```

```
        self.pooled_output = tf.layers.dense(
            first_token_tensor,
            config.hidden_size,
            activation=tf.tanh,
            kernel_initializer=create_initializer(config.initialize
```

```
  def get_pooled_output(self):
    return self.pooled_output

  def get_sequence_output(self):
    """Gets final hidden layer of encoder.

    Returns:
      float Tensor of shape [batch_size, seq_length, hidden_size] c
      to the final hidden of the transformer encoder.
    """
    return self.sequence_output

  def get_all_encoder_layers(self):
    return self.all_encoder_layers

  def get_embedding_output(self):
    """Gets output of the embedding lookup (i.e., input to the tran

    Returns:
      float Tensor of shape [batch_size, seq_length, hidden_size] c
      to the output of the embedding layer, after summing the word
      embeddings with the positional embeddings and the token type
      then performing layer normalization. This is the input to the
    """
    return self.embedding_output

  def get_embedding_table(self):
    return self.embedding_table


def gelu(x):
  """Gaussian Error Linear Unit.

  This is a smoother version of the RELU.
```

```
  Original paper: https://arxiv.org/abs/1606.08415
  Args:
    x: float Tensor to perform activation.

  Returns:
    `x` with the GELU activation applied.
  """
  cdf = 0.5 * (1.0 + tf.tanh(
      (np.sqrt(2 / np.pi) * (x + 0.044715 * tf.pow(x, 3)))))
  return x * cdf


def get_activation(activation_string):
  """Maps a string to a Python function, e.g., "relu" => `tf.nn.rel

  Args:
    activation_string: String name of the activation function.

  Returns:
    A Python function corresponding to the activation function. If
    `activation_string` is None, empty, or "linear", this will retu
    If `activation_string` is not a string, it will return `activat

  Raises:
    ValueError: The `activation_string` does not correspond to a kn
      activation.
  """

  # We assume that anything that"s not a string is already an activ
  # function, so we just return it.
  if not isinstance(activation_string, six.string_types):
    return activation_string

  if not activation_string:
    return None

  act = activation_string.lower()
  if act == "linear":
    return None
  elif act == "relu":
    return tf.nn.relu
  elif act == "gelu":
```

```
    return gelu
  elif act == "tanh":
    return tf.tanh
  else:
    raise ValueError("Unsupported activation: %s" % act)


def feedforward_adapter(input_tensor, hidden_size=64, init_scale=1e
  """A feedforward adapter layer with a bottleneck.

  Implements a bottleneck layer with a user-specified nonlinearity
  identity residual connection. All variables created are added to
  "adapters" collection.

  Args:
    input_tensor: input Tensor of shape [batch size, hidden dimensi
    hidden_size: dimension of the bottleneck layer.
    init_scale: Scale of the initialization distribution used for w

  Returns:
    Tensor of the same shape as x.
  """
  with tf.variable_scope("adapters"):
    in_size = input_tensor.get_shape().as_list()[1]
    w1 = tf.get_variable(
        "weights1", [in_size, hidden_size],
        initializer=tf.truncated_normal_initializer(stddev=init_sca
        collections=["adapters", tf.GraphKeys.GLOBAL_VARIABLES])
    b1 = tf.get_variable(
        "biases1", [1, hidden_size],
        initializer=tf.zeros_initializer(),
        collections=["adapters", tf.GraphKeys.GLOBAL_VARIABLES])
    net = tf.tensordot(input_tensor, w1, [[1], [0]]) + b1

    net = gelu(net)

    w2 = tf.get_variable(
        "weights2", [hidden_size, in_size],
        initializer=tf.truncated_normal_initializer(stddev=init_sca
        collections=["adapters", tf.GraphKeys.GLOBAL_VARIABLES])
    b2 = tf.get_variable(
        "biases2", [1, in_size],
```

```
        initializer=tf.zeros_initializer(),
        collections=["adapters", tf.GraphKeys.GLOBAL_VARIABLES])
    net = tf.tensordot(net, w2, [[1], [0]]) + b2

  return net + input_tensor


def get_adapter(function_string):
  """Maps a string to a Python function.

  Args:
    function_string: String name of the adapter function.

  Returns:
    A Python function corresponding to the adatper function.
    `function_string` is None or empty, will return None.
    If `function_string` is not a string, it will return `function_

  Raises:
    ValueError: The `function_string` does not correspond to a know
      adapter.
  """

  # We assume that anything that"s not a string is already an adapt
  # function, so we just return it.
  if not isinstance(function_string, six.string_types):
    return function_string

  if not function_string:
    return None

  fn = function_string.lower()
  if fn == "feedforward_adapter":
    return feedforward_adapter
  else:
    raise ValueError("Unsupported adapters: %s" % fn)


def get_assignment_map_from_checkpoint(tvars, init_checkpoint):
  """Compute the union of the current variables and checkpoint vari
  assignment_map = {}
  initialized_variable_names = {}
```

```
  name_to_variable = collections.OrderedDict()
  for var in tvars:
    name = var.name
    m = re.match("^(.*):\\d+$", name)
    if m is not None:
      name = m.group(1)
    name_to_variable[name] = var

  init_vars = tf.train.list_variables(init_checkpoint)

  assignment_map = collections.OrderedDict()
  for x in init_vars:
    (name, var) = (x[0], x[1])
    if name not in name_to_variable:
      continue
    assignment_map[name] = name
    initialized_variable_names[name] = 1
    initialized_variable_names[name + ":0"] = 1

  return (assignment_map, initialized_variable_names)


def dropout(input_tensor, dropout_prob):
  """Perform dropout.

  Args:
    input_tensor: float Tensor.
    dropout_prob: Python float. The probability of dropping out a v
      *keeping* a dimension as in `tf.nn.dropout`).

  Returns:
    A version of `input_tensor` with dropout applied.
  """
  if dropout_prob is None or dropout_prob == 0.0:
    return input_tensor

  output = tf.nn.dropout(input_tensor, 1.0 - dropout_prob)
  return output


def layer_norm(input_tensor, name=None):
```

```
  """Run layer normalization on the last dimension of the tensor."'
  return tf.contrib.layers.layer_norm(
      inputs=input_tensor, begin_norm_axis=-1, begin_params_axis=-1
      variables_collections=["layer_norm", tf.GraphKeys.GLOBAL_VARI


def layer_norm_and_dropout(input_tensor, dropout_prob, name=None):
  """Runs layer normalization followed by dropout."""
  output_tensor = layer_norm(input_tensor, name)
  output_tensor = dropout(output_tensor, dropout_prob)
  return output_tensor
```

```
def create_initializer(initializer_range=0.02):
  """Creates a `truncated_normal_initializer` with the given range.
  return tf.truncated_normal_initializer(stddev=initializer_range)


def embedding_lookup(input_ids,
                     vocab_size,
                     embedding_size=128,
                     initializer_range=0.02,
                     word_embedding_name="word_embeddings",
                     use_one_hot_embeddings=False):
  """Looks up words embeddings for id tensor.

  Args:
    input_ids: int32 Tensor of shape [batch_size, seq_length] conta
      ids.
    vocab_size: int. Size of the embedding vocabulary.
    embedding_size: int. Width of the word embeddings.
    initializer_range: float. Embedding initialization range.
    word_embedding_name: string. Name of the embedding table.
    use_one_hot_embeddings: bool. If True, use one-hot method for w
      embeddings. If False, use `tf.gather()`.

  Returns:
    float Tensor of shape [batch_size, seq_length, embedding_size].
  """
  # This function assumes that the input is of shape [batch_size, s
  # num_inputs].
```

```
  #
  # If the input is a 2D tensor of shape [batch_size, seq_length],
  # reshape to [batch_size, seq_length, 1].
  if input_ids.shape.ndims == 2:
    input_ids = tf.expand_dims(input_ids, axis=[-1])

  embedding_table = tf.get_variable(
      name=word_embedding_name,
      shape=[vocab_size, embedding_size],
      initializer=create_initializer(initializer_range))

  flat_input_ids = tf.reshape(input_ids, [-1])
  if use_one_hot_embeddings:
    one_hot_input_ids = tf.one_hot(flat_input_ids, depth=vocab_size
    output = tf.matmul(one_hot_input_ids, embedding_table)
  else:
    output = tf.gather(embedding_table, flat_input_ids)

  input_shape = get_shape_list(input_ids)

  output = tf.reshape(output,
                      input_shape[0:-1] + [input_shape[-1] * embedd
  return (output, embedding_table)


def embedding_postprocessor(input_tensor,
                            use_token_type=False,
                            token_type_ids=None,
                            token_type_vocab_size=16,
                            token_type_embedding_name="token_type_e
                            use_position_embeddings=True,
                            position_embedding_name="position_embed
                            initializer_range=0.02,
                            max_position_embeddings=512,
                            dropout_prob=0.1):
  """Performs various post-processing on a word embedding tensor.

  Args:
    input_tensor: float Tensor of shape [batch_size, seq_length,
      embedding_size].
    use_token_type: bool. Whether to add embeddings for `token_type
    token_type_ids: (optional) int32 Tensor of shape [batch_size, s
```

```
      Must be specified if `use_token_type` is True.
    token_type_vocab_size: int. The vocabulary size of `token_type_
    token_type_embedding_name: string. The name of the embedding ta
      for token type ids.
    use_position_embeddings: bool. Whether to add position embeddin
      position of each token in the sequence.
    position_embedding_name: string. The name of the embedding tabl
      for positional embeddings.
    initializer_range: float. Range of the weight initialization.
    max_position_embeddings: int. Maximum sequence length that migh
      used with this model. This can be longer than the sequence le
      input_tensor, but cannot be shorter.
    dropout_prob: float. Dropout probability applied to the final o

  Returns:
    float tensor with same shape as `input_tensor`.

  Raises:
    ValueError: One of the tensor shapes or input values is invalid
  """
  input_shape = get_shape_list(input_tensor, expected_rank=3)
  batch_size = input_shape[0]
  seq_length = input_shape[1]
  width = input_shape[2]

  output = input_tensor

  if use_token_type:
    if token_type_ids is None:
      raise ValueError("`token_type_ids` must be specified if"
                       "`use_token_type` is True.")
    token_type_table = tf.get_variable(
        name=token_type_embedding_name,
        shape=[token_type_vocab_size, width],
        initializer=create_initializer(initializer_range))
    # This vocab will be small so we always do one-hot here, since
    # faster for a small vocabulary.
    flat_token_type_ids = tf.reshape(token_type_ids, [-1])
    one_hot_ids = tf.one_hot(flat_token_type_ids, depth=token_type_
    token_type_embeddings = tf.matmul(one_hot_ids, token_type_table
    token_type_embeddings = tf.reshape(token_type_embeddings,
                                       [batch_size, seq_length, wid
```

```
    output += token_type_embeddings

  if use_position_embeddings:
    assert_op = tf.assert_less_equal(seq_length, max_position_embed
    with tf.control_dependencies([assert_op]):
      full_position_embeddings = tf.get_variable(
          name=position_embedding_name,
          shape=[max_position_embeddings, width],
          initializer=create_initializer(initializer_range))
      # Since the position embedding table is a learned variable, w
      # using a (long) sequence length `max_position_embeddings`. T
      # sequence length might be shorter than this, for faster trai
      # tasks that do not have long sequences.
      #
      # So `full_position_embeddings` is effectively an embedding t
      # for position [0, 1, 2, ..., max_position_embeddings-1], and
      # sequence has positions [0, 1, 2, ... seq_length-1], so we c
      # perform a slice.
      position_embeddings = tf.slice(full_position_embeddings, [0,
                                     [seq_length, -1])
      num_dims = len(output.shape.as_list())

      # Only the last two dimensions are relevant (`seq_length` and
      # we broadcast among the first dimensions, which is typically
      # the batch size.
      position_broadcast_shape = []
      for _ in range(num_dims - 2):
        position_broadcast_shape.append(1)
      position_broadcast_shape.extend([seq_length, width])
      position_embeddings = tf.reshape(position_embeddings,
                                       position_broadcast_shape)
      output += position_embeddings

  output = layer_norm_and_dropout(output, dropout_prob)
  return output
```

```
def create_attention_mask_from_input_mask(from_tensor, to_mask):
  """Create 3D attention mask from a 2D tensor mask.

  Args:
    from_tensor: 2D or 3D Tensor of shape [batch_size, from_seq_len
```

```
    to_mask: int32 Tensor of shape [batch_size, to_seq_length].

  Returns:
    float Tensor of shape [batch_size, from_seq_length, to_seq_leng
  """
  from_shape = get_shape_list(from_tensor, expected_rank=[2, 3])
  batch_size = from_shape[0]
  from_seq_length = from_shape[1]

  to_shape = get_shape_list(to_mask, expected_rank=2)
  to_seq_length = to_shape[1]

  to_mask = tf.cast(
      tf.reshape(to_mask, [batch_size, 1, to_seq_length]), tf.float

  # We don't assume that `from_tensor` is a mask (although it could
  # don't actually care if we attend *from* padding tokens (only *t
  # tokens so we create a tensor of all ones.
  #
  # `broadcast_ones` = [batch_size, from_seq_length, 1]
  broadcast_ones = tf.ones(
      shape=[batch_size, from_seq_length, 1], dtype=tf.float32)

  # Here we broadcast along two dimensions to create the mask.
  mask = broadcast_ones * to_mask

  return mask


def attention_layer(from_tensor,
                    to_tensor,
                    attention_mask=None,
                    num_attention_heads=1,
                    size_per_head=512,
                    query_act=None,
                    key_act=None,
                    value_act=None,
                    attention_probs_dropout_prob=0.0,
                    initializer_range=0.02,
                    do_return_2d_tensor=False,
                    batch_size=None,
                    from_seq_length=None,
```

```
                    to_seq_length=None):
  """Performs multi-headed attention from `from_tensor` to `to_tens

  This is an implementation of multi-headed attention based on "Att
  is all you Need". If `from_tensor` and `to_tensor` are the same,
  this is self-attention. Each timestep in `from_tensor` attends to
  corresponding sequence in `to_tensor`, and returns a fixed-with v

  This function first projects `from_tensor` into a "query" tensor
  `to_tensor` into "key" and "value" tensors. These are (effectivel
  of tensors of length `num_attention_heads`, where each tensor is
  [batch_size, seq_length, size_per_head].

  Then, the query and key tensors are dot-producted and scaled. The
  softmaxed to obtain attention probabilities. The value tensors ar
  interpolated by these probabilities, then concatenated back to a
  tensor and returned.

  In practice, the multi-headed attention are done with transposes
  reshapes rather than actual separate tensors.

  Args:
    from_tensor: float Tensor of shape [batch_size, from_seq_length
      from_width].
    to_tensor: float Tensor of shape [batch_size, to_seq_length, to
    attention_mask: (optional) int32 Tensor of shape [batch_size,
      from_seq_length, to_seq_length]. The values should be 1 or 0.
      attention scores will effectively be set to -infinity for any
      the mask that are 0, and will be unchanged for positions that
    num_attention_heads: int. Number of attention heads.
    size_per_head: int. Size of each attention head.
    query_act: (optional) Activation function for the query transfo
    key_act: (optional) Activation function for the key transform.
    value_act: (optional) Activation function for the value transfo
    attention_probs_dropout_prob: (optional) float. Dropout probabi
      attention probabilities.
    initializer_range: float. Range of the weight initializer.
    do_return_2d_tensor: bool. If True, the output will be of shape
      * from_seq_length, num_attention_heads * size_per_head]. If F
      output will be of shape [batch_size, from_seq_length, num_att
      * size_per_head].
    batch_size: (Optional) int. If the input is 2D, this might be t
```

```
      of the 3D version of the `from_tensor` and `to_tensor`.
    from_seq_length: (Optional) If the input is 2D, this might be t
      of the 3D version of the `from_tensor`.
    to_seq_length: (Optional) If the input is 2D, this might be the
      of the 3D version of the `to_tensor`.

  Returns:
    float Tensor of shape [batch_size, from_seq_length,
      num_attention_heads * size_per_head]. (If `do_return_2d_tensc
      true, this will be of shape [batch_size * from_seq_length,
      num_attention_heads * size_per_head]).

  Raises:
    ValueError: Any of the arguments or tensor shapes are invalid.
  """

  def transpose_for_scores(input_tensor, batch_size, num_attention_
                           seq_length, width):
    output_tensor = tf.reshape(
        input_tensor, [batch_size, seq_length, num_attention_heads,

    output_tensor = tf.transpose(output_tensor, [0, 2, 1, 3])
    return output_tensor

  from_shape = get_shape_list(from_tensor, expected_rank=[2, 3])
  to_shape = get_shape_list(to_tensor, expected_rank=[2, 3])

  if len(from_shape) != len(to_shape):
    raise ValueError(
        "The rank of `from_tensor` must match the rank of `to_tensc

  if len(from_shape) == 3:
    batch_size = from_shape[0]
    from_seq_length = from_shape[1]
    to_seq_length = to_shape[1]
  elif len(from_shape) == 2:
    if (batch_size is None or from_seq_length is None or to_seq_len
      raise ValueError(
          "When passing in rank 2 tensors to attention_layer, the v
          "for `batch_size`, `from_seq_length`, and `to_seq_length`
          "must all be specified.")
```

```
  # Scalar dimensions referenced here:
  #   B = batch size (number of sequences)
  #   F = `from_tensor` sequence length
  #   T = `to_tensor` sequence length
  #   N = `num_attention_heads`
  #   H = `size_per_head`

  from_tensor_2d = reshape_to_matrix(from_tensor)
  to_tensor_2d = reshape_to_matrix(to_tensor)

  # `query_layer` = [B*F, N*H]
  query_layer = tf.layers.dense(
      from_tensor_2d,
      num_attention_heads * size_per_head,
      activation=query_act,
      name="query",
      kernel_initializer=create_initializer(initializer_range))

  # `key_layer` = [B*T, N*H]
  key_layer = tf.layers.dense(
      to_tensor_2d,
      num_attention_heads * size_per_head,
      activation=key_act,
      name="key",
      kernel_initializer=create_initializer(initializer_range))

  # `value_layer` = [B*T, N*H]
  value_layer = tf.layers.dense(
      to_tensor_2d,
      num_attention_heads * size_per_head,
      activation=value_act,
      name="value",
      kernel_initializer=create_initializer(initializer_range))

  # `query_layer` = [B, N, F, H]
  query_layer = transpose_for_scores(query_layer, batch_size,
                                     num_attention_heads, from_seq_
                                     size_per_head)

  # `key_layer` = [B, N, T, H]
  key_layer = transpose_for_scores(key_layer, batch_size, num_atter
                                   to_seq_length, size_per_head)
```

```
# Take the dot product between "query" and "key" to get the raw
# attention scores.
# `attention_scores` = [B, N, F, T]
attention_scores = tf.matmul(query_layer, key_layer, transpose_b=
attention_scores = tf.multiply(attention_scores,
                               1.0 / math.sqrt(float(size_per_hea

if attention_mask is not None:
  # `attention_mask` = [B, 1, F, T]
  attention_mask = tf.expand_dims(attention_mask, axis=[1])

  # Since attention_mask is 1.0 for positions we want to attend a
  # masked positions, this operation will create a tensor which i
  # positions we want to attend and -10000.0 for masked positions
  adder = (1.0 - tf.cast(attention_mask, tf.float32)) * -10000.0

  # Since we are adding it to the raw scores before the softmax,
  # effectively the same as removing these entirely.
  attention_scores += adder

# Normalize the attention scores to probabilities.
# `attention_probs` = [B, N, F, T]
attention_probs = tf.nn.softmax(attention_scores)

# This is actually dropping out entire tokens to attend to, which
# seem a bit unusual, but is taken from the original Transformer
attention_probs = dropout(attention_probs, attention_probs_dropou

# `value_layer` = [B, T, N, H]
value_layer = tf.reshape(
    value_layer,
    [batch_size, to_seq_length, num_attention_heads, size_per_hea

# `value_layer` = [B, N, T, H]
value_layer = tf.transpose(value_layer, [0, 2, 1, 3])

# `context_layer` = [B, N, F, H]
context_layer = tf.matmul(attention_probs, value_layer)

# `context_layer` = [B, F, N, H]
context_layer = tf.transpose(context_layer, [0, 2, 1, 3])
```

```
  if do_return_2d_tensor:
    # `context_layer` = [B*F, N*H]
    context_layer = tf.reshape(
        context_layer,
        [batch_size * from_seq_length, num_attention_heads * size_p
  else:
    # `context_layer` = [B, F, N*H]
    context_layer = tf.reshape(
        context_layer,
        [batch_size, from_seq_length, num_attention_heads * size_pe

  return context_layer
```

```
def transformer_model(input_tensor,
                      attention_mask=None,
                      hidden_size=768,
                      num_hidden_layers=12,
                      num_attention_heads=12,
                      intermediate_size=3072,
                      intermediate_act_fn=gelu,
                      hidden_dropout_prob=0.1,
                      attention_probs_dropout_prob=0.1,
                      initializer_range=0.02,
                      do_return_all_layers=False,
                      adapter_fn=None):
  """Multi-headed, multi-layer Transformer from "Attention is All Y

  This is almost an exact implementation of the original Transforme

  See the original paper:
  https://arxiv.org/abs/1706.03762

  Also see:
  https://github.com/tensorflow/tensor2tensor/blob/master/tensor2te

  Args:
    input_tensor: float Tensor of shape [batch_size, seq_length, hi
    attention_mask: (optional) int32 Tensor of shape [batch_size, s
      seq_length], with 1 for positions that can be attended to and
```

```
      positions that should not be.
    hidden_size: int. Hidden size of the Transformer.
    num_hidden_layers: int. Number of layers (blocks) in the Transf
    num_attention_heads: int. Number of attention heads in the Tran
    intermediate_size: int. The size of the "intermediate" (a.k.a.,
      forward) layer.
    intermediate_act_fn: function. The non-linear activation functi
      to the output of the intermediate/feed-forward layer.
    hidden_dropout_prob: float. Dropout probability for the hidden
    attention_probs_dropout_prob: float. Dropout probability of the
      probabilities.
    initializer_range: float. Range of the initializer (stddev of t
      normal).
    do_return_all_layers: Whether to also return all layers or just
      layer.
    adapter_fn: (optional) trainable adapter function that takes as
      Tensor and returns a Tensor of the same shape.

  Returns:
    float Tensor of shape [batch_size, seq_length, hidden_size], th
    hidden layer of the Transformer.

  Raises:
    ValueError: A Tensor shape or parameter is invalid.
  """
  if hidden_size % num_attention_heads != 0:
    raise ValueError(
        "The hidden size (%d) is not a multiple of the number of at
        "heads (%d)" % (hidden_size, num_attention_heads))

  attention_head_size = int(hidden_size / num_attention_heads)
  input_shape = get_shape_list(input_tensor, expected_rank=3)
  batch_size = input_shape[0]
  seq_length = input_shape[1]
  input_width = input_shape[2]

  # The Transformer performs sum residuals on all layers so the inp
  # to be the same as the hidden size.
  if input_width != hidden_size:
    raise ValueError("The width of the input tensor (%d) != hidden
                     (input_width, hidden_size))
```

```
  # We keep the representation as a 2D tensor to avoid re-shaping i
  # forth from a 3D tensor to a 2D tensor. Re-shapes are normally f
  # the GPU/CPU but may not be free on the TPU, so we want to minim
  # help the optimizer.
  prev_output = reshape_to_matrix(input_tensor)

  all_layer_outputs = []
  for layer_idx in range(num_hidden_layers):
    with tf.variable_scope("layer_%d" % layer_idx):
      layer_input = prev_output

      with tf.variable_scope("attention"):
        attention_heads = []
        with tf.variable_scope("self"):
          attention_head = attention_layer(
              from_tensor=layer_input,
              to_tensor=layer_input,
              attention_mask=attention_mask,
              num_attention_heads=num_attention_heads,
              size_per_head=attention_head_size,
              attention_probs_dropout_prob=attention_probs_dropout_
              initializer_range=initializer_range,
              do_return_2d_tensor=True,
              batch_size=batch_size,
              from_seq_length=seq_length,
              to_seq_length=seq_length)
          attention_heads.append(attention_head)

        attention_output = None
        if len(attention_heads) == 1:
          attention_output = attention_heads[0]
        else:
          # In the case where we have other sequences, we just conc
          # them to the self-attention head before the projection.
          attention_output = tf.concat(attention_heads, axis=-1)

        # Run a linear projection of `hidden_size` then add a resid
        # with `layer_input`.
        with tf.variable_scope("output"):
          attention_output = tf.layers.dense(
              attention_output,
              hidden_size,
```

```
              kernel_initializer=create_initializer(initializer_ran
          attention_output = dropout(attention_output, hidden_dropo
          if adapter_fn:
            attention_output = adapter_fn(attention_output)
          attention_output = layer_norm(attention_output + layer_in

      # The activation is only applied to the "intermediate" hidden
      with tf.variable_scope("intermediate"):
        intermediate_output = tf.layers.dense(
            attention_output,
            intermediate_size,
            activation=intermediate_act_fn,
            kernel_initializer=create_initializer(initializer_range

      # Down-project back to `hidden_size` then add the residual.
      with tf.variable_scope("output"):
        layer_output = tf.layers.dense(
            intermediate_output,
            hidden_size,
            kernel_initializer=create_initializer(initializer_range
        layer_output = dropout(layer_output, hidden_dropout_prob)
        if adapter_fn:
          layer_output = adapter_fn(layer_output)
        layer_output = layer_norm(layer_output + attention_output)
        prev_output = layer_output
        all_layer_outputs.append(layer_output)

  if do_return_all_layers:
    final_outputs = []
    for layer_output in all_layer_outputs:
      final_output = reshape_from_matrix(layer_output, input_shape)
      final_outputs.append(final_output)
    return final_outputs
  else:
    final_output = reshape_from_matrix(prev_output, input_shape)
    return final_output


def get_shape_list(tensor, expected_rank=None, name=None):
  """Returns a list of the shape of tensor, preferring static dimen

  Args:
```

```
    tensor: A tf.Tensor object to find the shape of.
    expected_rank: (optional) int. The expected rank of `tensor`. I
      specified and the `tensor` has a different rank, and exceptio
      thrown.
    name: Optional name of the tensor for the error message.

  Returns:
    A list of dimensions of the shape of tensor. All static dimensi
    be returned as python integers, and dynamic dimensions will be
    as tf.Tensor scalars.
  """
  if name is None:
    name = tensor.name

  if expected_rank is not None:
    assert_rank(tensor, expected_rank, name)

  shape = tensor.shape.as_list()

  non_static_indexes = []
  for (index, dim) in enumerate(shape):
    if dim is None:
      non_static_indexes.append(index)

  if not non_static_indexes:
    return shape

  dyn_shape = tf.shape(tensor)
  for index in non_static_indexes:
    shape[index] = dyn_shape[index]
  return shape


def reshape_to_matrix(input_tensor):
  """Reshapes a >= rank 2 tensor to a rank 2 tensor (i.e., a matrix
  ndims = input_tensor.shape.ndims
  if ndims < 2:
    raise ValueError("Input tensor must have at least rank 2. Shape
                     (input_tensor.shape))
  if ndims == 2:
    return input_tensor
```

```
  width = input_tensor.shape[-1]
  output_tensor = tf.reshape(input_tensor, [-1, width])
  return output_tensor


def reshape_from_matrix(output_tensor, orig_shape_list):
  """Reshapes a rank 2 tensor back to its original rank >= 2 tenso
  if len(orig_shape_list) == 2:
    return output_tensor

  output_shape = get_shape_list(output_tensor)

  orig_dims = orig_shape_list[0:-1]
  width = output_shape[-1]

  return tf.reshape(output_tensor, orig_dims + [width])


def assert_rank(tensor, expected_rank, name=None):
  """Raises an exception if the tensor rank is not of the expected

  Args:
    tensor: A tf.Tensor to check the rank of.
    expected_rank: Python integer or list of integers, expected ra
    name: Optional name of the tensor for the error message.

  Raises:
    ValueError: If the expected shape doesn't match the actual sha
  """
  if name is None:
    name = tensor.name

  expected_rank_dict = {}
  if isinstance(expected_rank, six.integer_types):
    expected_rank_dict[expected_rank] = True
  else:
    for x in expected_rank:
      expected_rank_dict[x] = True

  actual_rank = tensor.shape.ndims
  if actual_rank not in expected_rank_dict:
    scope_name = tf.get_variable_scope().name
```

```
    raise ValueError(
        "For the tensor `%s` in scope `%s`, the actual rank "
        "`%d` (shape = %s) is not equal to the expected rank `%s`"
        (name, scope_name, actual_rank, str(tensor.shape), str(expe
```

modeling_test.py

```
# coding=utf-8
# Copyright 2018 The Google AI Language Team Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, softwa
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
# See the License for the specific language governing permissions a
# limitations under the License.
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import collections
import json
import random
import re

import modeling
import six
import tensorflow as tf


class BertModelTest(tf.test.TestCase):

  class BertModelTester(object):

    def __init__(self,
                 parent,
                 batch_size=13,
```

```
                 seq_length=7,
                 is_training=True,
                 use_input_mask=True,
                 use_token_type_ids=True,
                 vocab_size=99,
                 hidden_size=32,
                 num_hidden_layers=5,
                 num_attention_heads=4,
                 intermediate_size=37,
                 hidden_act="gelu",
                 hidden_dropout_prob=0.1,
                 attention_probs_dropout_prob=0.1,
                 max_position_embeddings=512,
                 type_vocab_size=16,
                 initializer_range=0.02,
                 scope=None):
      self.parent = parent
      self.batch_size = batch_size
      self.seq_length = seq_length
      self.is_training = is_training
      self.use_input_mask = use_input_mask
      self.use_token_type_ids = use_token_type_ids
      self.vocab_size = vocab_size
      self.hidden_size = hidden_size
      self.num_hidden_layers = num_hidden_layers
      self.num_attention_heads = num_attention_heads
      self.intermediate_size = intermediate_size
      self.hidden_act = hidden_act
      self.hidden_dropout_prob = hidden_dropout_prob
      self.attention_probs_dropout_prob = attention_probs_dropout_p
      self.max_position_embeddings = max_position_embeddings
      self.type_vocab_size = type_vocab_size
      self.initializer_range = initializer_range
      self.scope = scope

    def create_model(self):
      input_ids = BertModelTest.ids_tensor([self.batch_size, self.s
                                           self.vocab_size)

      input_mask = None
      if self.use_input_mask:
        input_mask = BertModelTest.ids_tensor(
```

```
            [self.batch_size, self.seq_length], vocab_size=2)

      token_type_ids = None
      if self.use_token_type_ids:
        token_type_ids = BertModelTest.ids_tensor(
            [self.batch_size, self.seq_length], self.type_vocab_siz

      config = modeling.BertConfig(
          vocab_size=self.vocab_size,
          hidden_size=self.hidden_size,
          num_hidden_layers=self.num_hidden_layers,
          num_attention_heads=self.num_attention_heads,
          intermediate_size=self.intermediate_size,
          hidden_act=self.hidden_act,
          hidden_dropout_prob=self.hidden_dropout_prob,
          attention_probs_dropout_prob=self.attention_probs_dropout
          max_position_embeddings=self.max_position_embeddings,
          type_vocab_size=self.type_vocab_size,
          initializer_range=self.initializer_range)

      model = modeling.BertModel(
          config=config,
          is_training=self.is_training,
          input_ids=input_ids,
          input_mask=input_mask,
          token_type_ids=token_type_ids,
          scope=self.scope)

      outputs = {
          "embedding_output": model.get_embedding_output(),
          "sequence_output": model.get_sequence_output(),
          "pooled_output": model.get_pooled_output(),
          "all_encoder_layers": model.get_all_encoder_layers(),
      }
      return outputs

    def check_output(self, result):
      self.parent.assertAllEqual(
          result["embedding_output"].shape,
          [self.batch_size, self.seq_length, self.hidden_size])

      self.parent.assertAllEqual(
```

```
          result["sequence_output"].shape,
          [self.batch_size, self.seq_length, self.hidden_size])

      self.parent.assertAllEqual(result["pooled_output"].shape,
                                 [self.batch_size, self.hidden_size

  def test_default(self):
    self.run_tester(BertModelTest.BertModelTester(self))

  def test_config_to_json_string(self):
    config = modeling.BertConfig(vocab_size=99, hidden_size=37)
    obj = json.loads(config.to_json_string())
    self.assertEqual(obj["vocab_size"], 99)
    self.assertEqual(obj["hidden_size"], 37)

  def run_tester(self, tester):
    with self.test_session() as sess:
      ops = tester.create_model()
      init_op = tf.group(tf.global_variables_initializer(),
                         tf.local_variables_initializer())
      sess.run(init_op)
      output_result = sess.run(ops)
      tester.check_output(output_result)

      self.assert_all_tensors_reachable(sess, [init_op, ops])

  @classmethod
  def ids_tensor(cls, shape, vocab_size, rng=None, name=None):
    """Creates a random int32 tensor of the shape within the vocab
    if rng is None:
      rng = random.Random()

    total_dims = 1
    for dim in shape:
      total_dims *= dim

    values = []
    for _ in range(total_dims):
      values.append(rng.randint(0, vocab_size - 1))

    return tf.constant(value=values, dtype=tf.int32, shape=shape, r
```

```
  def assert_all_tensors_reachable(self, sess, outputs):
    """Checks that all the tensors in the graph are reachable from
    graph = sess.graph

    ignore_strings = [
        "^.*/assert_less_equal/.*$",
        "^.*/dilation_rate$",
        "^.*/Tensordot/concat$",
        "^.*/Tensordot/concat/axis$",
        "^testing/.*$",
    ]

    ignore_regexes = [re.compile(x) for x in ignore_strings]

    unreachable = self.get_unreachable_ops(graph, outputs)
    filtered_unreachable = []
    for x in unreachable:
      do_ignore = False
      for r in ignore_regexes:
        m = r.match(x.name)
        if m is not None:
          do_ignore = True
      if do_ignore:
        continue
      filtered_unreachable.append(x)
    unreachable = filtered_unreachable

    self.assertEqual(
        len(unreachable), 0, "The following ops are unreachable: %s
        (" ".join([x.name for x in unreachable])))

  @classmethod
  def get_unreachable_ops(cls, graph, outputs):
    """Finds all of the tensors in graph that are unreachable from
    outputs = cls.flatten_recursive(outputs)
    output_to_op = collections.defaultdict(list)
    op_to_all = collections.defaultdict(list)
    assign_out_to_in = collections.defaultdict(list)

    for op in graph.get_operations():
      for x in op.inputs:
        op_to_all[op.name].append(x.name)
```

```
      for y in op.outputs:
        output_to_op[y.name].append(op.name)
        op_to_all[op.name].append(y.name)
      if str(op.type) == "Assign":
        for y in op.outputs:
          for x in op.inputs:
            assign_out_to_in[y.name].append(x.name)

    assign_groups = collections.defaultdict(list)
    for out_name in assign_out_to_in.keys():
      name_group = assign_out_to_in[out_name]
      for n1 in name_group:
        assign_groups[n1].append(out_name)
        for n2 in name_group:
          if n1 != n2:
            assign_groups[n1].append(n2)

    seen_tensors = {}
    stack = [x.name for x in outputs]
    while stack:
      name = stack.pop()
      if name in seen_tensors:
        continue
      seen_tensors[name] = True

      if name in output_to_op:
        for op_name in output_to_op[name]:
          if op_name in op_to_all:
            for input_name in op_to_all[op_name]:
              if input_name not in stack:
                stack.append(input_name)

      expanded_names = []
      if name in assign_groups:
        for assign_name in assign_groups[name]:
          expanded_names.append(assign_name)

      for expanded_name in expanded_names:
        if expanded_name not in stack:
          stack.append(expanded_name)

    unreachable_ops = []
```

```
    for op in graph.get_operations():
      is_unreachable = False
      all_names = [x.name for x in op.inputs] + [x.name for x in op
      for name in all_names:
        if name not in seen_tensors:
          is_unreachable = True
      if is_unreachable:
        unreachable_ops.append(op)
    return unreachable_ops

  @classmethod
  def flatten_recursive(cls, item):
    """Flattens (potentially nested) a tuple/dictionary/list to a l
    output = []
    if isinstance(item, list):
      output.extend(item)
    elif isinstance(item, tuple):
      output.extend(list(item))
    elif isinstance(item, dict):
      for (_, v) in six.iteritems(item):
        output.append(v)
    else:
      return [item]

    flat_output = []
    for x in output:
      flat_output.extend(cls.flatten_recursive(x))
    return flat_output


if __name__ == "__main__":
  tf.test.main()
```

optimazation.py

```
# coding=utf-8
# Copyright 2018 The Google AI Language Team Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
```

```
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, softwa
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
# See the License for the specific language governing permissions a
# limitations under the License.
"""Functions and classes related to optimization (weight updates)."

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import re
import tensorflow as tf


def create_optimizer(loss, init_lr, num_train_steps, num_warmup_ste
  """Creates an optimizer training op."""
  global_step = tf.train.get_or_create_global_step()

  learning_rate = tf.constant(value=init_lr, shape=[], dtype=tf.flo

  # Implements linear decay of the learning rate.
  learning_rate = tf.train.polynomial_decay(
      learning_rate,
      global_step,
      num_train_steps,
      end_learning_rate=0.0,
      power=1.0,
      cycle=False)

  # Implements linear warmup. I.e., if global_step < num_warmup_ste
  # learning rate will be `global_step/num_warmup_steps * init_lr`.
  if num_warmup_steps:
    global_steps_int = tf.cast(global_step, tf.int32)
    warmup_steps_int = tf.constant(num_warmup_steps, dtype=tf.int32

    global_steps_float = tf.cast(global_steps_int, tf.float32)
    warmup_steps_float = tf.cast(warmup_steps_int, tf.float32)

    warmup_percent_done = global_steps_float / warmup_steps_float
```

```
    warmup_learning_rate = init_lr * warmup_percent_done

    is_warmup = tf.cast(global_steps_int < warmup_steps_int, tf.flo
    learning_rate = (
        (1.0 - is_warmup) * learning_rate + is_warmup * warmup_lear

  # It is recommended that you use this optimizer for fine tuning,
  # is how the model was trained (note that the Adam m/v variables
  # loaded from init_checkpoint.)
  optimizer = AdamWeightDecayOptimizer(
      learning_rate=learning_rate,
      weight_decay_rate=0.01,
      adapter_weight_decay_rate=0.01,
      beta_1=0.9,
      beta_2=0.999,
      epsilon=1e-6,
      exclude_from_weight_decay=["LayerNorm", "layer_norm", "bias"]

  if use_tpu:
    optimizer = tf.contrib.tpu.CrossShardOptimizer(optimizer)

  tvars = []
  for collection in ["adapters", "layer_norm", "head"]:
    tvars += tf.get_collection(collection)
  grads = tf.gradients(loss, tvars)

  # This is how the model was pre-trained.
  (grads, _) = tf.clip_by_global_norm(grads, clip_norm=1.0)

  train_op = optimizer.apply_gradients(
      zip(grads, tvars), global_step=global_step)

  # Normally the global step update is done inside of `apply_gradie
  # However, `AdamWeightDecayOptimizer` doesn't do this. But if you
  # a different optimizer, you should probably take this line out.
  new_global_step = global_step + 1
  train_op = tf.group(train_op, [global_step.assign(new_global_step
  return train_op


class AdamWeightDecayOptimizer(tf.train.Optimizer):
  """A basic Adam optimizer that includes "correct" L2 weight decay
```

```
def __init__(self,
             learning_rate,
             weight_decay_rate=0.0,
             adapter_weight_decay_rate=0.0,
             beta_1=0.9,
             beta_2=0.999,
             epsilon=1e-6,
             exclude_from_weight_decay=None,
             name="AdamWeightDecayOptimizer"):
  """Constructs a AdamWeightDecayOptimizer."""
  super(AdamWeightDecayOptimizer, self).__init__(False, name)

  self.learning_rate = learning_rate
  self.weight_decay_rate = weight_decay_rate
  self.adapter_weight_decay_rate = adapter_weight_decay_rate
  self.beta_1 = beta_1
  self.beta_2 = beta_2
  self.epsilon = epsilon
  self.exclude_from_weight_decay = exclude_from_weight_decay
  self._adapter_variable_names = {
      self._get_variable_name(v.name) for v in tf.get_collection(
  }

def apply_gradients(self, grads_and_vars, global_step=None, name=
  """See base class."""
  assignments = []
  for (grad, param) in grads_and_vars:
    if grad is None or param is None:
      continue

    param_name = self._get_variable_name(param.name)

    m = tf.get_variable(
        name=param_name + "/adam_m",
        shape=param.shape.as_list(),
        dtype=tf.float32,
        trainable=False,
        initializer=tf.zeros_initializer())
    v = tf.get_variable(
        name=param_name + "/adam_v",
        shape=param.shape.as_list(),
```

```
          dtype=tf.float32,
          trainable=False,
          initializer=tf.zeros_initializer())

      # Standard Adam update.
      next_m = (
          tf.multiply(self.beta_1, m) + tf.multiply(1.0 - self.beta
      next_v = (
          tf.multiply(self.beta_2, v) + tf.multiply(1.0 - self.beta
                                                    tf.square(grad)

      update = next_m / (tf.sqrt(next_v) + self.epsilon)

      # Just adding the square of the weights to the loss function
      # the correct way of using L2 regularization/weight decay wit
      # since that will interact with the m and v parameters in str
      #
      # Instead we want ot decay the weights in a manner that doesn
      # with the m/v parameters. This is equivalent to adding the s
      # of the weights to the loss with plain (non-momentum) SGD.
      if self._do_use_weight_decay(param_name):
        if param_name in self._adapter_variable_names:
          update += self.adapter_weight_decay_rate * param
        else:
          update += self.weight_decay_rate * param

      update_with_lr = self.learning_rate * update

      next_param = param - update_with_lr

      assignments.extend(
          [param.assign(next_param),
           m.assign(next_m),
           v.assign(next_v)])
    return tf.group(*assignments, name=name)

  def _do_use_weight_decay(self, param_name):
    """Whether to use L2 weight decay for `param_name`."""
    if param_name in self._adapter_variable_names:
      if not self.adapter_weight_decay_rate:
        return False
    else:
```

```
      if not self.weight_decay_rate:
        return False

    if self.exclude_from_weight_decay:
      for r in self.exclude_from_weight_decay:
        if re.search(r, param_name) is not None:
          return False

    return True

  def _get_variable_name(self, param_name):
    """Get the variable name from the tensor name."""
    m = re.match("^(.*):\\d+$", param_name)
    if m is not None:
      param_name = m.group(1)
    return param_name
```

optimization_test.py

```
# coding=utf-8
# Copyright 2018 The Google AI Language Team Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, softwa
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
# See the License for the specific language governing permissions a
# limitations under the License.
"""Functions and classes related to optimization (weight updates)."

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import re
import tensorflow as tf
```

```
def create_optimizer(loss, init_lr, num_train_steps, num_warmup_ste
  """Creates an optimizer training op."""
  global_step = tf.train.get_or_create_global_step()

  learning_rate = tf.constant(value=init_lr, shape=[], dtype=tf.flo

  # Implements linear decay of the learning rate.
  learning_rate = tf.train.polynomial_decay(
      learning_rate,
      global_step,
      num_train_steps,
      end_learning_rate=0.0,
      power=1.0,
      cycle=False)

  # Implements linear warmup. I.e., if global_step < num_warmup_ste
  # learning rate will be `global_step/num_warmup_steps * init_lr`.
  if num_warmup_steps:
    global_steps_int = tf.cast(global_step, tf.int32)
    warmup_steps_int = tf.constant(num_warmup_steps, dtype=tf.int32

    global_steps_float = tf.cast(global_steps_int, tf.float32)
    warmup_steps_float = tf.cast(warmup_steps_int, tf.float32)

    warmup_percent_done = global_steps_float / warmup_steps_float
    warmup_learning_rate = init_lr * warmup_percent_done

    is_warmup = tf.cast(global_steps_int < warmup_steps_int, tf.flo
    learning_rate = (
        (1.0 - is_warmup) * learning_rate + is_warmup * warmup_lear

  # It is recommended that you use this optimizer for fine tuning,
  # is how the model was trained (note that the Adam m/v variables
  # loaded from init_checkpoint.)
  optimizer = AdamWeightDecayOptimizer(
      learning_rate=learning_rate,
      weight_decay_rate=0.01,
      adapter_weight_decay_rate=0.01,
      beta_1=0.9,
      beta_2=0.999,
      epsilon=1e-6,
```

```
      exclude_from_weight_decay=["LayerNorm", "layer_norm", "bias"]

  if use_tpu:
    optimizer = tf.contrib.tpu.CrossShardOptimizer(optimizer)

  tvars = []
  for collection in ["adapters", "layer_norm", "head"]:
    tvars += tf.get_collection(collection)
  grads = tf.gradients(loss, tvars)

  # This is how the model was pre-trained.
  (grads, _) = tf.clip_by_global_norm(grads, clip_norm=1.0)

  train_op = optimizer.apply_gradients(
      zip(grads, tvars), global_step=global_step)

  # Normally the global step update is done inside of `apply_gradie
  # However, `AdamWeightDecayOptimizer` doesn't do this. But if you
  # a different optimizer, you should probably take this line out.
  new_global_step = global_step + 1
  train_op = tf.group(train_op, [global_step.assign(new_global_step
  return train_op


class AdamWeightDecayOptimizer(tf.train.Optimizer):
  """A basic Adam optimizer that includes "correct" L2 weight decay

  def __init__(self,
               learning_rate,
               weight_decay_rate=0.0,
               adapter_weight_decay_rate=0.0,
               beta_1=0.9,
               beta_2=0.999,
               epsilon=1e-6,
               exclude_from_weight_decay=None,
               name="AdamWeightDecayOptimizer"):
    """Constructs a AdamWeightDecayOptimizer."""
    super(AdamWeightDecayOptimizer, self).__init__(False, name)

    self.learning_rate = learning_rate
    self.weight_decay_rate = weight_decay_rate
    self.adapter_weight_decay_rate = adapter_weight_decay_rate
```

```
    self.beta_1 = beta_1
    self.beta_2 = beta_2
    self.epsilon = epsilon
    self.exclude_from_weight_decay = exclude_from_weight_decay
    self._adapter_variable_names = {
        self._get_variable_name(v.name) for v in tf.get_collection(
    }

  def apply_gradients(self, grads_and_vars, global_step=None, name=
    """See base class."""
    assignments = []
    for (grad, param) in grads_and_vars:
      if grad is None or param is None:
        continue

      param_name = self._get_variable_name(param.name)

      m = tf.get_variable(
          name=param_name + "/adam_m",
          shape=param.shape.as_list(),
          dtype=tf.float32,
          trainable=False,
          initializer=tf.zeros_initializer())
      v = tf.get_variable(
          name=param_name + "/adam_v",
          shape=param.shape.as_list(),
          dtype=tf.float32,
          trainable=False,
          initializer=tf.zeros_initializer())

      # Standard Adam update.
      next_m = (
          tf.multiply(self.beta_1, m) + tf.multiply(1.0 - self.beta
      next_v = (
          tf.multiply(self.beta_2, v) + tf.multiply(1.0 - self.beta
                                                    tf.square(grad)

      update = next_m / (tf.sqrt(next_v) + self.epsilon)

      # Just adding the square of the weights to the loss function
      # the correct way of using L2 regularization/weight decay wit
      # since that will interact with the m and v parameters in str
```

```
      #
      # Instead we want ot decay the weights in a manner that doesn
      # with the m/v parameters. This is equivalent to adding the s
      # of the weights to the loss with plain (non-momentum) SGD.
      if self._do_use_weight_decay(param_name):
        if param_name in self._adapter_variable_names:
          update += self.adapter_weight_decay_rate * param
        else:
          update += self.weight_decay_rate * param

      update_with_lr = self.learning_rate * update

      next_param = param - update_with_lr

      assignments.extend(
          [param.assign(next_param),
           m.assign(next_m),
           v.assign(next_v)])
    return tf.group(*assignments, name=name)

  def _do_use_weight_decay(self, param_name):
    """Whether to use L2 weight decay for `param_name`."""
    if param_name in self._adapter_variable_names:
      if not self.adapter_weight_decay_rate:
        return False
    else:
      if not self.weight_decay_rate:
        return False

    if self.exclude_from_weight_decay:
      for r in self.exclude_from_weight_decay:
        if re.search(r, param_name) is not None:
          return False

    return True

  def _get_variable_name(self, param_name):
    """Get the variable name from the tensor name."""
    m = re.match("^(.*):\\d+$", param_name)
    if m is not None:
```

```
        param_name = m.group(1)
      return param_name
```

Adapter-BERT 프로젝트 분석

1. 사전 학습 (Unsupervised Pre-Training)

BERT 모델의 사전 학습은 대규모 텍스트 데이터를 이용해 일반적인 언어적 패턴을 학습하는 과정입니다. 이 프로젝트에서 사용된 토크나이저는 `tokenization.py` 파일에 정의된 **WordPieceTokenizer** 클래스를 사용하여 텍스트를 서브워드 단위로 변환합니다. 이를 통해 모델은 희귀한 단어 및 신조어에 대한 OOV 문제를 해결할 수 있습니다【19:17†source】.

2. 과제별 미세 조정 (Fine-Tuning)

미세 조정은 `run_classifier.py`에서 정의된 **MRPC**와 같은 특정 작업을 위해 수행됩니다. 해당 파일에서는 MRPC 작업에 대한 학습 및 평가 과정이 포함되어 있습니다. `run_classifier.py`는 BERT 모델을 조정하여 해당 작업에서 더 나은 성능을 낼 수 있도록 돕습니다【19:12†source】.

3. 최적화 전략 (Optimization Strategies)

Adam 옵티마이저는 `optimization.py` 파일에 정의되어 있으며, 학습 과정에서 학습률을 조절해주는 **AdamWeightDecayOptimizer** 클래스가 있습니다. 최적화는 `optimization_test.py`에서 이를 테스트하여 Adam 옵티마이저가 효과적으로 동작하는지 확인하는 코드를 포함하고 있습니다【19:19†source】.

4. 모델 구조 (Model Architecture)

모델은 `modeling.py` 파일에서 정의된 **BertModel**을 사용합니다. BERT 모델의 주요 구조는 트랜스포머 아키텍처 기반으로 설계되어 있으며, 각 입력에 대한 문맥적 정보를 학습합니다. 이 파일에서는 BERT의 여러 레이어와 **self-attention 메커니즘**을 통해 모델을 구성합니다【19:9†source】【19:14†source】.

5. 손실 함수 및 보조 목적 (Loss Functions and Auxiliary Objectives)

손실 함수는 BERT 모델이 최종적으로 예측한 결과와 실제 레이블 간의 차이를 최소화하는 역할을 합니다. `run_classifier.py`에서 분류 작업에 대한 손실 함수가 정의되어 있으며, **softmax**와 **cross-entropy** 손실을 사용하여 예측과 실제 값의 차이를 최소화합니다【19:5†source】【19:7†source】.

6. 평가 및 로깅 (Evaluation and Logging)

모델의 학습이 끝난 후, `run_classifier.py` 파일에서 **평가 단계**가 진행됩니다. **MRPC**와 같은 데이터셋에 대한 평가 결과가 로깅되고, 이는 로그 파일로 저장되어 나중에 분석할 수 있습니다【19:18†source】.

결론

Adapter-BERT는 BERT의 사전 학습 모델을 활용하여 다양한 NLP 과제에 맞게 미세 조정할 수 있는 효율적인 구조로 설계되었습니다. 이를 통해 미세 조정 시 적은 수의 파라미터만 업데이트해도 좋은 성능을 낼 수 있습니다. TensorFlow 기반의 모델 최적화 및 평가 전략도 적절히 적용되어 있습니다.

코드 파일 순서

1. `requirements.txt`: 프로젝트에서 사용되는 주요 라이브러리 및 의존성을 정의하는 파일입니다. TensorFlow와 같은 핵심 패키지 버전을 확인함으로써 환경 설정을 준비하는 데 유용합니다.
2. `tokenization.py`: 텍스트 데이터를 서브워드 단위로 변환하는 토크나이징 과정을 정의한 파일입니다. BERT 모델에 입력될 텍스트 데이터를 어떻게 전처리하고 준비하는지 이해하는 것이 중요하므로 이 부분을 먼저 분석하는 것이 좋습니다.
3. `tokenization_test.py`: 토크나이징 과정이 의도한 대로 작동하는지 확인하는 테스트 파일입니다. 토큰화된 출력이 적절한지 확인하는 테스트 케이스가 포함되어 있습니다.
4. `modeling.py`: BERT 모델의 전체 아키텍처가 정의된 파일로, 트랜스포머 구조와 self-attention 메커니즘을 다룹니다. 모델의 주요 구조와 학습 흐름을 파악하는 데 중요한 파일입니다.
5. `modeling_test.py`: 모델 구조가 올바르게 동작하는지 검증하는 테스트 케이스를 포함한 파일입니다. 각 레이어와 출력이 의도한 대로 작동하는지 확인하는 테스트입니다.
6. `optimization.py`: 모델 학습에 사용되는 최적화 전략이 정의된 파일로, Adam 옵티마이저와 학습률 스케줄링 방법이 포함되어 있습니다. 최적화 과정이 어떻게 이루어지는지 확인하는 데 중요한 파일입니다.
7. `optimization_test.py`: 최적화 전략이 제대로 작동하는지 검증하는 테스트 파일입니다. Adam 옵티마이저와 학습률 조정이 제대로 이루어지는지 확인하는 테스트 코드가 포함되어 있습니다.
8. `run_classifier.py`: BERT 모델을 특정 작업에 맞춰 미세 조정하고 학습하는 과정을 담당하는 파일입니다. 데이터 로딩, 학습 루프, 손실 함수, 평가 방식이 여기에 정의되어 있으며, 모델을 학습시키고 평가하는 데 가장 중요한 파일입니다.
9. `README.md`: 프로젝트의 목적, 사용법, 설치 방법 등이 설명된 파일입니다. 프로젝트의 전반적인 구조와 기능을 이해하는 데 도움을 줍니다.

`tokenization.py` (텍스트 전처리 및 토크나이징)

🍭 이 파일은 텍스트 데이터를 전처리하고, 모델이 입력할 수 있는 형태로 변환하는 역할을 합니다. 특히, BERT와 같은 대규모 모델에 사용되는 서브워드 토크나이징 기법인 WordPiece를 사용하여 텍스트를 서브워드 단위로 나누어 처리합니다.

1. 라이브러리 임포트

이 파일은 다음과 같은 주요 라이브러리를 사용합니다:

- **re**: 정규 표현식을 사용하여 텍스트 처리.
- **collections**: 어휘 빈도 계산 등에 사용되는 `OrderedDict`.
- six : Python 2와 Python 3의 호환성을 제공하는 라이브러리이다. Python 2와 3의 문법과 API 차이 때문에 발생하는 문제를 해결하기 위해 `six` 라이브러리를 사용하면, 코드가 두 버전 모두에서 작동할 수 있게 만든다.
  - **주요 기능**:
    - `six.PY2`: 현재 Python 버전이 2인지 확인한다.
    - `six.PY3`: 현재 Python 버전이 3인지 확인한다.
    - `six.iteritems()`, `six.text_type` 등 여러 함수와 타입이 Python 2와 3 간의 차이를 자동으로 처리해 준다.
- **unicodedata**: 유니코드 관련 작업을 수행할 수 있는 표준 라이브러리이다. 주로 유니코드 문자의 분류, 변환, 또는 이름을 조회하는 기능을 제공한다. 예를 들어, 유니코드 문자를 정규화하거나, 문자에 대한 특정 속성(예: 숫자, 문자)을 확인할 수 있다.
  - **주요 기능**:
    - `unicodedata.normalize()`: 유니코드 문자열을 정규화하여 특정 표준에 맞게 변환한다.
    - `unicodedata.category()`: 주어진 문자의 유니코드 범주(예: 문자, 숫자, 구두점)를 반환한다.
    - `unicodedata.name()`: 문자의 공식 유니코드 이름을 반환한다.

```
import collections
import re
import unicodedata
import six
```

2. WordPieceTokenizer 클래스

이 클래스는 WordPiece 방식을 통해 텍스트를 서브워드로 분할합니다. WordPiece는 자주 사용되는 문자 쌍을 병합하여 어휘 집합을 구성하며, 이는 OOV(Out of Vocabulary) 문제를 해결하는 데 유용합니다.

주요 메서드:

- `tokenize`: 텍스트를 서브워드로 토큰화합니다.
  - 입력 텍스트를 공백 기준으로 분할한 후, 각 단어에 대해 서브워드로 나누는 과정이 수행됩니다.
  - Greedy longest-match-first 알고리즘을 사용하여 가능한 가장 긴 서브워드를 선택합니다.

```
class WordpieceTokenizer(object):
    def __init__(self, vocab, unk_token="[UNK]", max_inpu
t_chars_per_word=100):
        self.vocab = vocab
        self.unk_token = unk_token
        self.max_input_chars_per_word = max_input_chars_p
er_word

    def tokenize(self, text):
        output_tokens = []
        for token in whitespace_tokenize(text):
            chars = list(token)
            if len(chars) > self.max_input_chars_per_wor
d:
                output_tokens.append(self.unk_token)
                continue

            is_bad = False
            start = 0
            sub_tokens = []
            while start < len(chars):
                end = len(chars)
                cur_substr = None
                while start < end:
                    substr = "".join(chars[start:end])
                    if start > 0:
                        substr = "##" + substr
                    if substr in self.vocab:
                        cur_substr = substr
                        break
                    end -= 1
                if cur_substr is None:
                    is_bad = True
```

```
                    break
                sub_tokens.append(cur_substr)
                start = end

            if is_bad:
                output_tokens.append(self.unk_token)
            else:
                output_tokens.extend(sub_tokens)
        return output_tokens
```

- **설명**: `tokenize` 메서드는 입력된 텍스트를 공백을 기준으로 분리한 후, 각 단어를 서브워드로 변환합니다. 단어의 길이가 너무 길면 `[UNK]` 토큰을 반환하고, 그렇지 않으면 가능한 가장 긴 서브워드로 분할하여 토큰화합니다.

3. BasicTokenizer 클래스

이 클래스는 기본적인 텍스트 전처리를 담당합니다. 입력 텍스트를 소문자로 변환하거나 악센트를 제거하고, 문장 부호를 분리하는 기능을 제공합니다.

주요 메서드:

- `tokenize`: 입력 텍스트를 공백으로 분리하여 토큰으로 변환하고, 소문자 변환 및 악센트 제거 등의 전처리를 수행합니다.

```
class BasicTokenizer(object):
    def __init__(self, do_lower_case=True):
        self.do_lower_case = do_lower_case

    def tokenize(self, text):
        text = self._clean_text(text)
        text = self._tokenize_chinese_chars(text)
        orig_tokens = whitespace_tokenize(text)
        split_tokens = []
        for token in orig_tokens:
            if self.do_lower_case:
                token = token.lower()
                token = self._run_strip_accents(token)
            split_tokens.extend(self._run_split_on_punc(t
oken))
        return whitespace_tokenize(" ".join(split_token
s))
```

- **설명**: 이 메서드는 입력 텍스트를 전처리하여 소문자로 변환하고, 악센트를 제거하며, 중국어 문자를 처리한 후 문장 부호를 기준으로 분리합니다.

4. FullTokenizer 클래스

이 클래스는 **BasicTokenizer**와 **WordPieceTokenizer**를 결합하여 완전한 토크나이징을 수행합니다. 기본적인 전처리를 거친 후 서브워드로 변환하는 과정이 포함됩니다.

주요 메서드:

- `tokenize`: BasicTokenizer로 기본 토큰화를 수행한 후, WordPieceTokenizer를 사용해 서브워드 단위로 분리합니다.

```
class FullTokenizer(object):
    def __init__(self, vocab_file, do_lower_case=True):
        self.vocab = load_vocab(vocab_file)
        self.basic_tokenizer = BasicTokenizer(do_lower_ca
se=do_lower_case)
        self.wordpiece_tokenizer = WordpieceTokenizer(voc
ab=self.vocab)

    def tokenize(self, text):
        split_tokens = []
        for token in self.basic_tokenizer.tokenize(text):
            for sub_token in self.wordpiece_tokenizer.tok
enize(token):
                split_tokens.append(sub_token)
        return split_tokens
```

- **설명**: 이 메서드는 먼저 BasicTokenizer를 사용해 텍스트를 기본적으로 토큰화한 후, 각 토큰을 WordPieceTokenizer를 사용해 서브워드로 변환합니다.

5. 유틸리티 함수

텍스트 전처리를 지원하는 여러 유틸리티 함수가 제공됩니다.

- **convert_tokens_to_ids**: 토큰을 어휘집에 있는 ID로 변환합니다.

```
def convert_tokens_to_ids(vocab, tokens):
    return [vocab[token] for token in tokens]
```

- **convert_ids_to_tokens**: ID를 다시 토큰으로 변환합니다.

```
def convert_ids_to_tokens(inv_vocab, ids):
    return [inv_vocab[id] for id in ids]
```

- **whitespace_tokenize**: 기본적인 공백 제거 및 텍스트 분리를 처리합니다.

```
def whitespace_tokenize(text):
    text = text.strip()
    if not text:
        return []
    return text.split()
```

6. 코드의 핵심 기능 요약

- **WordPiece**: 서브워드 단위로 텍스트를 분리하여 처리함으로써, 희귀 단어와 신조어에 대한 효율적인 학습을 지원합니다.
- **토큰화 과정**: 입력된 텍스트는 BasicTokenizer를 통해 기본적으로 전처리된 후, WordPieceTokenizer를 사용해 서브워드로 나누어집니다.
- **텍스트 변환**: 최종적으로 텍스트는 토큰 ID로 변환되며, 이를 통해 모델의 입력으로 사용됩니다.

이 파일은 텍스트 데이터를 모델에 맞는 형식으로 전처리하여, BERT와 같은 모델에서 높은 성능을 발휘할 수 있도록 돕는 중요한 역할을 합니다.

🤫 `From __future__` 가 뭐야!!

`from __future__ import absolute_import`, `division`, `print_function`은 Python 2에서 Python 3의 기능을 사용할 수 있도록 해주는 모듈입니다. 이 세 가지는 Python 2와 Python 3 간의 차이를 호환하기 위해 Python 2 코드에 포함되는 경우가 많습니다. 각각의 기능에 대해 설명하면 다음과 같습니다:

1. `from __future__ import absolute_import`

- **기능**: Python 2에서 **절대 임포트** 방식을 활성화합니다.
- **설명**: Python 2에서는 모듈을 상대적으로 임포트하는 방식이 기본이었기 때문에, 동일한 이름의 로컬 모듈과 표준 라이브러리 모듈이 충돌할 수 있었습니다. `absolute_import`는 이러한 충돌을 방지하고 Python 3처럼 절대 경로를 우선하는 방식으로 임포트를 처리합니다.
- **예시**:

```
# 예를 들어, 현재 디렉토리에 있는 "math.py" 모듈 대신 표준 라이브러리 "math" 모듈을 임포트하게 됩니다.
import math
```

2. `from __future__ import division`

- **기능**: Python 2에서 **소수점 나눗셈**을 기본으로 활성화합니다.
- **설명**: Python 2에서는 `/` 연산자가 두 정수의 나눗셈을 수행할 때 정수 나눗셈(몫만 반환)을 기본으로 했습니다. `division`을 사용하면 Python 3처럼 `/` 연산자가 항상 소수점 나눗셈을 수행합니다. 정수 나눗셈을 원하면 `//` 연산자를 사용해야 합니다.
- **예시**:

```
# Python 2의 기본 동작 (정수 나눗셈)
print(3 / 2)  # 결과: 1

# __future__ import 이후 (소수점 나눗셈)
print(3 / 2)  # 결과: 1.5
```

3. `from __future__ import print_function`

- **기능**: Python 2에서 **Python 3의 print 함수** 문법을 사용할 수 있게 합니다.
- **설명**: Python 2에서는 `print`가 키워드로 사용되었으나, Python 3에서는 함수로 변경되었습니다. `print_function`을 사용하면 Python 3처럼 `print()` 함수를 호출해야 하며, 인자 옵션(예: `end`나 `file`)도 사용할 수 있습니다.

- **예시**:

```
# Python 2 기본 동작 (print가 키워드)
print "Hello"  # 결과: Hello

# __future__ import 이후 (print 함수 사용)
print("Hello")  # 결과: Hello
```

`tokenization_test.py` (토크나이저 테스트 파일)

🍭 이 파일은 `tokenization.py`에 정의된 기능들을 테스트하기 위한 유닛 테스트입니다. 주로 `FullTokenizer`, `WordpieceTokenizer`, `BasicTokenizer` 클래스의 기능을 검증합니다. 다양한 테스트 케이스를 통해 토큰화 결과가 기대하는 대로 나오는지 확인합니다.

1. 라이브러리 임포트

테스트를 위해 필요한 라이브러리를 로드합니다:

```
import os
import tempfile
import tokenization
import six
import tensorflow as tf
```

2. TokenizationTest 클래스

이 클래스는 여러 테스트 메서드를 포함하며, 토큰화 과정이 제대로 동작하는지 확인합니다.

3. `test_full_tokenizer` 메서드

- **기능**: `FullTokenizer` 클래스의 토큰화 결과를 테스트합니다.
- **작동 방식**:
  - 임시로 vocab 파일을 생성하고, `FullTokenizer`를 사용해 `tokenize` 및 `convert_tokens_to_ids` 메서드를 검증합니다.
  - 예시 텍스트 `"UNwantéd,running"`을 토큰화하고 기대값인 `["un", "##want", "##ed", ",", "runn", "##ing"]`와 비교합니다.
  - 토큰 ID 변환도 테스트됩니다.

```
def test_full_tokenizer(self):
    vocab_tokens = ["[UNK]", "[CLS]", "[SEP]", "want", "#
#want", "##ed", "wa", "un", "runn", "##ing", ","]
    with tempfile.NamedTemporaryFile(delete=False) as voc
ab_writer:
        if six.PY2:
            vocab_writer.write("".join([x + "\\n" for x i
n vocab_tokens]))
        else:
            vocab_writer.write("".join([x + "\\n" for x i
n vocab_tokens]).encode("utf-8"))
```

```
        vocab_file = vocab_writer.name

    tokenizer = tokenization.FullTokenizer(vocab_file)
    os.unlink(vocab_file)

    tokens = tokenizer.tokenize(u"UNwant\\u00E9d,runnin
g")
    self.assertAllEqual(tokens, ["un", "##want", "##ed",
",", "runn", "##ing"])
    self.assertAllEqual(tokenizer.convert_tokens_to_ids(t
okens), [7, 4, 5, 10, 8, 9])
```

4. `test_basic_tokenizer_lower` 메서드

- **기능**: `BasicTokenizer` 클래스의 소문자 변환 기능을 테스트합니다.
- **작동 방식**:
  - 텍스트를 소문자로 변환한 뒤 결과가 올바른지 확인합니다.
  - 예시로 `" \\tHeLLo!how \\n Are yoU? "` 문자열이 `["hello", "!", "how", "are", "you", "?"]`로 잘 변환되는지 검증합니다.

```
def test_basic_tokenizer_lower(self):
    tokenizer = tokenization.BasicTokenizer(do_lower_case
=True)
    self.assertAllEqual(tokenizer.tokenize(u" \\tHeLLo!ho
w  \\n Are yoU?  "), ["hello", "!", "how", "are", "you",
"?"])
    self.assertAllEqual(tokenizer.tokenize(u"H\\u00E9ll
o"), ["hello"])
```

5. `test_wordpiece_tokenizer` 메서드

- **기능**: `WordpieceTokenizer` 클래스의 기능을 테스트합니다.
- **작동 방식**:
  - `WordpieceTokenizer`가 서브워드 토큰화를 제대로 수행하는지 확인합니다.
  - `"unwanted running"`이라는 텍스트가 `[un, ##want, ##ed, runn, ##ing]`으로 잘 토큰화되는지 검증합니다.

```
def test_wordpiece_tokenizer(self):
    vocab_tokens = ["[UNK]", "[CLS]", "[SEP]", "want", "#
```

```
#want", "##ed", "wa", "un", "runn", "##ing"]
    vocab = {token: i for i, token in enumerate(vocab_tok
ens)}
    tokenizer = tokenization.WordpieceTokenizer(vocab=voc
ab)

    self.assertAllEqual(tokenizer.tokenize("unwanted runn
ing"), ["un", "##want", "##ed", "runn", "##ing"])
    self.assertAllEqual(tokenizer.tokenize("unwantedX run
ning"), ["[UNK]", "runn", "##ing"])
```

6. 기타 테스트 메서드

- `test_is_whitespace`: `whitespace_tokenize` 함수가 공백 문자를 잘 처리하는지 테스트합니다.
- `test_is_control`: 제어 문자가 제대로 구분되는지 검증합니다.
- `test_is_punctuation`: 구두점 문자가 제대로 분리되는지 테스트합니다.

7. 결론

이 파일은 토크나이저의 다양한 기능을 종합적으로 테스트하며, 토큰화 결과가 올바르게 나오는지 확인하는 중요한 역할을 합니다. BERT 모델과 같은 NLP 모델에서 텍스트를 적절히 처리하고 변환하는지 확인하기 위한 필수적인 테스트 파일입니다.

`modeling.py` (BERT 모델 정의 및 관련 함수)

1. 파일 개요

`modeling.py` 파일은 BERT 모델의 핵심 구조를 정의한 파일입니다. 주로 BERT 모델의 구성 요소와 학습을 위한 여러 주요 함수들을 포함하고 있습니다. 이 파일은 텍스트 데이터를 처리하여 BERT 모델의 입력으로 변환하고, 다양한 구성 요소를 설정하는 역할을 합니다.

2. `BertConfig` 클래스

`BertConfig` 클래스는 BERT 모델의 구성 요소들을 정의하는 설정 클래스입니다. 모델의 크기, 레이어 수, 드롭아웃 확률, 어텐션 헤드 수 등 여러 하이퍼파라미터를 지정합니다.

주요 인수:

- `vocab_size`: 어휘 집합의 크기.
- `hidden_size`: 은닉층의 크기.
- `num_hidden_layers`: Transformer 인코더 내 은닉 레이어 수.
- `num_attention_heads`: 어텐션 레이어의 헤드 수.
- `intermediate_size`: 피드포워드 레이어의 크기.
- `hidden_act`: 활성화 함수 (기본값: `gelu`).
- `hidden_dropout_prob`: 은닉층 드롭아웃 확률.
- `attention_probs_dropout_prob`: 어텐션 확률 드롭아웃 비율.
- `max_position_embeddings`: 최대 문장 길이.
- `type_vocab_size`: 토큰 타입 아이디 크기 (예: 문장의 첫 번째/두 번째 부분을 구분하는 데 사용).
- `initializer_range`: 초기 가중치 설정 범위.

예시:

```
class BertConfig(object):
    def __init__(self,
                 vocab_size,
                 hidden_size=768,
                 num_hidden_layers=12,
                 num_attention_heads=12,
                 intermediate_size=3072,
                 hidden_act="gelu",
                 hidden_dropout_prob=0.1,
                 attention_probs_dropout_prob=0.1,
```

```
                    max_position_embeddings=512,
                    type_vocab_size=16,
                    initializer_range=0.02):
        ...
```

3. `BertModel` 클래스

`BertModel` 클래스는 실제 BERT 모델의 구조를 정의합니다. 이 클래스는 입력으로 주어진 텍스트를 처리하고, 학습 또는 추론을 위한 출력을 생성하는 역할을 합니다.

주요 메서드:

- **생성자**: 모델 구성 설정 (`BertConfig`), 학습 여부 (`is_training`), 입력 ID (`input_ids`), 입력 마스크 (`input_mask`), 토큰 타입 ID (`token_type_ids`), 단어 임베딩 방식 (`use_one_hot_embeddings`), 스코프 등을 받아 모델을 구성합니다.
- **주요 메서드**:
  - `embedding_lookup`: 단어 임베딩을 수행합니다.
  - `embedding_postprocessor`: 임베딩 후처리, 토큰 타입 및 위치 임베딩 추가.
  - `transformer_model`: 실제 Transformer 블록에서 어텐션과 피드포워드를 적용해 문장 표현을 학습합니다.
  - `get_pooled_output`: 문장의 마지막 [CLS] 토큰에 해당하는 표현을 반환합니다.

예시:

```
class BertModel(object):
    def __init__(self,
                 config,
                 is_training,
                 input_ids,
                 input_mask=None,
                 token_type_ids=None,
                 use_one_hot_embeddings=False,
                 scope=None):
        # Embedding 단계
        with tf.variable_scope(scope, default_name="ber
t"):
            with tf.variable_scope("embeddings"):
                (self.embedding_output, self.embedding_ta
ble) = embedding_lookup(
                    input_ids=input_ids,
                    vocab_size=config.vocab_size,
                    embedding_size=config.hidden_size,
```

```
                    initializer_range=config.initializer_
range)
                self.embedding_output = embedding_postpro
cessor(
                    input_tensor=self.embedding_output,
                    use_token_type=True,
                    token_type_ids=token_type_ids,
                    position_embedding_name="position_emb
eddings",
                    dropout_prob=config.hidden_dropout_pr
ob)

            # Encoder 단계 (Transformer 블록)
            with tf.variable_scope("encoder"):
                attention_mask = create_attention_mask_fr
om_input_mask(input_ids, input_mask)
                self.all_encoder_layers = transformer_mod
el(
                    input_tensor=self.embedding_output,
                    attention_mask=attention_mask,
                    hidden_size=config.hidden_size,
                    num_hidden_layers=config.num_hidden_l
ayers)
```

4. Transformer 모델

`transformer_model` 함수는 BERT의 핵심인 트랜스포머 블록을 구현한 함수입니다. 입력된 임베딩을 바탕으로 어텐션 메커니즘과 피드포워드 네트워크를 사용하여 텍스트의 문맥적 의미를 학습합니다.

예시:

```
def transformer_model(input_tensor, attention_mask, hidde
n_size, num_hidden_layers, num_attention_heads):
    for layer_idx in range(num_hidden_layers):
        with tf.variable_scope("layer_%d" % layer_idx):
            attention_heads = []
            with tf.variable_scope("attention"):
                attention_output = attention_layer(from_t
ensor=input_tensor, to_tensor=input_tensor, attention_mas
k=attention_mask)
            # 피드포워드 및 후처리
```

```
            with tf.variable_scope("intermediate"):
                intermediate_output = dense_layer_2d(inpu
t_tensor=attention_output, hidden_size=intermediate_size)
            # 출력을 next input으로
            input_tensor = intermediate_output
    return input_tensor
```

5. 기타 주요 함수

- `create_attention_mask_from_input_mask`: 입력 마스크로부터 어텐션 마스크를 생성하여 패딩된 위치를 무시합니다.
- `embedding_lookup`: 주어진 단어 ID에 대응하는 임베딩 벡터를 반환합니다.
- `embedding_postprocessor`: 위치 임베딩과 토큰 타입 임베딩을 추가하고, 드롭아웃을 적용합니다.

6. 파일의 역할

- 이 파일은 BERT 모델의 모든 주요 구성 요소를 정의하며, 텍스트 데이터를 처리해 입력으로 사용할 수 있도록 합니다.
- 모델의 학습 및 추론 단계에서 BERT의 인코더 블록(트랜스포머)을 거쳐 문장의 문맥적 정보를 추출합니다.

이 파일은 BERT의 핵심적인 구조를 다루며, 모델 학습 및 추론 과정에서 필수적인 역할을 합니다.

🤫 Model.py 상세 분석

`modeling.py` 파일에서 주요 클래스를 분석하고, 각 클래스와 그 내부 함수들을 하나씩 자세히 설명하겠다.

1. `BertConfig` 클래스

이 클래스는 BERT 모델의 설정을 관리하는 클래스다. BERT 모델을 사용할 때 필요한 여러 파라미터들을 저장하고, 이 파라미터를 기반으로 모델을 생성할 수 있다. 이 클래스는 주로 JSON 파일 또는 파이썬 딕셔너리에서 설정 정보를 불러와 `BertConfig` 객체를 생성하는 기능을 제공한다.

주요 함수들:

1. `from_dict(cls, json_object)`:
   - **기능**: 파이썬 딕셔너리로부터 `BertConfig` 객체를 생성한다.
   - **동작 방식**:
     - 빈 `BertConfig` 객체를 생성하고, 딕셔너리의 키와 값을 이용해 객체의 속성들을 설정한다.
     - `json_object`라는 딕셔너리에서 각 키와 값을 순차적으로 읽고, 이를 `config.__dict__[key] = value` 방식으로 `BertConfig`의 속성에 할당한다.
   - **예시**:

     ```
     json_object = {'vocab_size': 30522, 'hidden_size': 768}
     config = BertConfig.from_dict(json_object)
     ```

   - **결과**: `config` 객체는 `vocab_size=30522`, `hidden_size=768`의 값을 갖는다.
2. `from_json_file(cls, json_file)`:
   - **기능**: JSON 파일로부터 `BertConfig` 객체를 생성한다.
   - **동작 방식**:
     - 주어진 `json_file` 경로에서 파일을 열고 JSON 텍스트를 읽어들인 후, 이를 파이썬 딕셔너리로 변환한다.
     - 변환된 딕셔너리를 이용해 `from_dict()` 함수를 호출하여 `BertConfig` 객체를 생성한다.
   - **예시**:

     ```
     config = BertConfig.from_json_file('config.json')
     ```

- **결과**: 파일에 저장된 파라미터들로 설정된 `BertConfig` 객체를 반환한다.

3. `to_dict(self)`:
   - **기능**: `BertConfig` 객체를 파이썬 딕셔너리로 변환한다.
   - **동작 방식**: 객체의 속성들을 모두 딕셔너리로 반환하며, 이를 통해 설정값을 쉽게 확인할 수 있다.
4. `to_json_string(self)`:
   - **기능**: `BertConfig` 객체를 JSON 문자열로 변환한다.
   - **동작 방식**: 객체의 속성을 JSON 형식의 문자열로 변환해 반환한다.

2. `BertModel` 클래스

이 클래스는 BERT 모델의 구조를 구현한 클래스다. 주로 Transformer 레이어로 구성되며, 입력된 문장을 처리하여 문맥적인 표현을 학습하고 다양한 NLP 작업에 활용할 수 있는 표현을 생성한다.

주요 함수들:

1. `__init__(self, config, is_training, input_ids, ...)`:
   - **기능**: BERT 모델의 인스턴스를 초기화한다.
   - **동작 방식**:
     - 주어진 `config`를 이용해 모델의 크기, 레이어 수, 어텐션 헤드 수 등 모델의 구조를 설정한다.
     - 입력 데이터(`input_ids`, `input_mask`, `token_type_ids`)를 기반으로 임베딩을 계산하고, 이를 Transformer 모델에 전달한다.
     - 최종적으로 각 문장의 표현을 생성한다.
2. `embedding_lookup`:
   - **기능**: 입력 토큰 ID에 대응하는 임베딩 벡터를 찾아 반환한다.
   - **동작 방식**:
     - 입력 토큰을 기반으로 단어 임베딩을 찾아 벡터 형태로 변환한다.
     - 이 벡터는 모델의 입력으로 사용된다.
3. `embedding_postprocessor`:
   - **기능**: 임베딩 후처리를 수행하여, 토큰 타입 임베딩 및 위치 임베딩을 추가한다.
   - **동작 방식**: 입력 임베딩 벡터에 토큰 타입과 위치 정보를 결합하고, 드롭아웃을 적용한다.
4. `transformer_model`:

- **기능**: Transformer 모델의 핵심 함수로, 문맥적 의존성을 학습한다.
- **동작 방식**:
  - 입력 임베딩 벡터를 Transformer 블록에 전달하고, 어텐션 메커니즘을 통해 각 단어의 상관관계를 학습한다.
  - 피드포워드 네트워크를 통해 각 단어의 표현을 학습한 후, 최종적으로 모든 레이어의 출력을 반환한다.

5. `get_pooled_output`:
   - **기능**: 문장의 마지막 [CLS] 토큰에 해당하는 표현을 반환한다.
   - **동작 방식**: BERT 모델의 마지막 출력에서 [CLS] 토큰의 표현을 추출하고, 이를 문장 수준의 표현으로 활용한다.

요약

- `BertConfig` **클래스**는 BERT 모델을 설정하는 클래스이며, JSON 파일 또는 딕셔너리로부터 설정을 불러와 객체를 생성할 수 있다.
- `BertModel` **클래스**는 BERT 모델의 구조를 정의하는 클래스이며, Transformer 레이어를 통해 입력 문장의 문맥적 표현을 학습하고, 문장 또는 단어 수준의 출력을 생성한다.

각 클래스는 BERT 모델의 기본 구조와 설정을 관리하며, 이를 통해 사전 학습된 모델을 다양한 NLP 작업에 적용할 수 있다.

`modeling.py` 파일의 전반적인 구조는 BERT 모델을 구현하고 다양한 전처리 및 후처리 과정을 처리하는 코드로 구성되어 있다. 이 파일은 BERT 모델의 기본 구조와 함께 Transformer 레이어, 임베딩 처리, 드롭아웃, 어텐션, 그리고 BERT 모델의 파라미터 효율적 미세 조정을 지원하는 Adapter 모듈을 포함하고 있다.

전반적인 구조

1. **라이브러리 임포트**
   - TensorFlow 및 기타 필요한 모듈들을 임포트하여 모델을 구축하는 데 사용된다. 주요 라이브러리로는 `tensorflow`, `six`, `copy` 등이 있다.
2. `BertConfig` **클래스**
   - **설정 클래스**: 이 클래스는 BERT 모델의 설정값(하이퍼파라미터)을 관리한다. `vocab_size`, `hidden_size`, `num_attention_heads` 등의 모델의 구조를 결정하는 파라미터들이 포함되어 있다.
   - **주요 함수들**:
     - `from_dict`: 파라미터 딕셔너리로부터 `BertConfig` 객체를 생성한다.

    - `from_json_file`: JSON 파일에서 파라미터를 읽어 `BertConfig` 객체를 생성한다.
    - `to_dict` 및 `to_json_string`: `BertConfig` 객체를 딕셔너리 또는 JSON 형식의 문자열로 변환한다.

3. `BertModel` **클래스**
    - **BERT 모델 클래스**: BERT 모델의 실제 구조를 구현한 클래스이다. 입력 데이터를 임베딩하고, Transformer 레이어를 통해 문맥적 표현을 학습하며, 최종적으로 필요한 작업을 수행할 수 있는 출력값을 생성한다.
    - **주요 함수들**:
        - `__init__`: BERT 모델의 초기화 함수로, 설정값에 맞게 BERT 모델의 구조를 설정하고 필요한 레이어들을 구성한다.
        - `embedding_lookup`: 입력된 텍스트의 토큰 ID를 임베딩 벡터로 변환하는 함수이다.
        - `embedding_postprocessor`: 토큰 타입 임베딩과 위치 임베딩을 추가하여, 각 토큰의 의미와 순서를 나타내는 정보를 더한다.
        - `get_pooled_output`: 마지막 [CLS] 토큰의 표현을 추출해 문장 분류 등의 작업에 사용한다.
        - `transformer_model`: BERT 모델의 핵심 부분인 Transformer 레이어를 실행해 문맥적 정보를 학습한다.

4. **Transformer 및 어텐션 관련 함수**
    - **Transformer 모델**은 BERT 모델의 핵심 구조로, 입력 문장의 각 토큰이 다른 토큰들과 상호작용하는 방식을 학습하는 메커니즘이다. 다중 헤드 어텐션과 피드포워드 네트워크로 이루어져 있으며, 이를 통해 문맥적 의존성을 학습한다.
    - **어텐션 마스크 생성**: 패딩된 토큰을 무시하고 유효한 토큰들 간의 관계를 학습할 수 있도록 어텐션 마스크를 생성하는 함수가 포함되어 있다.

5. **Adapter 모듈**
    - **파라미터 효율적인 학습**을 지원하는 모듈이다. Adapter는 모델의 모든 파라미터를 학습하는 대신, 특정 레이어에 작은 모듈을 추가해 소수의 파라미터만 학습한다. 이를 통해 성능을 유지하면서도 학습 시간을 단축하고, 메모리 사용량을 줄일 수 있다.

6. **활용 예시 및 추가 유틸리티 함수**
    - **get_activation**: 활성화 함수를 가져오는 함수로, `relu`나 `gelu`와 같은 활성화 함수가 사용된다.
    - **create_attention_mask_from_input_mask**: 입력 마스크로부터 어텐션 마스크를 생성하는 함수로, 각 토큰의 유효성을 결정하고 어텐션 계산 시 이를 반

영한다.

요약

- `BertConfig`: BERT 모델의 설정을 관리하는 클래스.
- `BertModel`: BERT 모델의 구조를 정의하고, 임베딩, Transformer 레이어, 어텐션, 드롭아웃 등 다양한 처리 과정을 수행하는 클래스.
- **Transformer 및 어텐션 관련 함수**: 다중 헤드 어텐션과 피드포워드 네트워크를 통해 입력 문장의 관계를 학습하는 메커니즘.
- **Adapter 모듈**: 파라미터 효율적인 학습을 위한 모듈로, 모델의 일부만 학습하는 방식을 제공.

이 파일은 BERT 모델을 구현하고, 이를 다양한 NLP 작업에 적용할 수 있도록 하는 주요 구성 요소들을 포함하고 있다.

🤫 `with` 문

1. `with` 문에 대한 설명

`with` 문은 파이썬에서 **컨텍스트 관리자**를 사용해 특정 블록의 실행을 제어하는 데 사용된다. 여기서는 **TensorFlow의 `tf.variable_scope`**를 컨텍스트 관리자로 사용하고 있다.

- *`tf.variable_scope`*는 변수를 정의할 때 해당 변수의 **이름 범위**를 지정해주기 위해 사용된다. 이렇게 하면 코드에서 같은 이름의 변수가 있을 때도, 각 변수가 속한 범위(scope)에 따라 구분할 수 있다. 즉, 이름 충돌을 피하고, 모델을 더 구조적으로 구성할 수 있다.

예를 들어:

```
with tf.variable_scope("scope1"):
    v = tf.get_variable("v", shape=[1])  # 이 변수는 scope1/v로 이름이 지정됨

with tf.variable_scope("scope2"):
    v = tf.get_variable("v", shape=[1])  # 이 변수는 scope2/v로 이름이 지정됨
```

이 코드에서는 `scope="bert"`로 지정된 범위에서 변수를 정의하게 된다.

2. 코드 설명

초기 설정

```
config = copy.deepcopy(config)
if not is_training:
  config.hidden_dropout_prob = 0.0
  config.attention_probs_dropout_prob = 0.0
```

- **deepcopy(config)**: `config` 객체를 깊은 복사하여 원본을 수정하지 않도록 함.
- **is_training**이 `False`일 때, **드롭아웃 확률을 0으로 설정**하여 학습 중이 아닐 때는 드롭아웃이 적용되지 않도록 한다.

입력 데이터 처리

```
input_shape = get_shape_list(input_ids, expected_rank=2)
batch_size = input_shape[0]
seq_length = input_shape[1]
```

```
if input_mask is None:
  input_mask = tf.ones(shape=[batch_size, seq_length], dt
ype=tf.int32)

if token_type_ids is None:
  token_type_ids = tf.zeros(shape=[batch_size, seq_lengt
h], dtype=tf.int32)
```

- `input_shape`: 입력 토큰 ID의 모양을 가져와 배치 크기(`batch_size`)와 시퀀스 길이(`seq_length`)를 설정한다. 이는 이후 모델에서 입력의 크기를 처리하는 데 사용된다.
- `input_mask`: 패딩 토큰을 무시하는 데 사용된다. 만약 입력 마스크가 없으면, 입력의 모든 토큰을 유효한 토큰으로 간주하는 마스크를 생성한다.
- `token_type_ids`: 두 문장이 있는 경우, 각 문장이 다른 타입이라는 것을 나타내는 ID를 설정한다. 이 값이 없으면 0으로 설정해 모든 토큰을 같은 타입으로 처리한다.

임베딩 처리

```
with tf.variable_scope(scope, default_name="bert"):
  with tf.variable_scope("embeddings"):
    (self.embedding_output, self.embedding_table) = embed
ding_lookup(
        input_ids=input_ids,
        vocab_size=config.vocab_size,
        embedding_size=config.hidden_size,
        initializer_range=config.initializer_range,
        word_embedding_name="word_embeddings",
        use_one_hot_embeddings=use_one_hot_embeddings)
```

- `embedding_lookup`: `input_ids`를 기반으로 각 토큰에 해당하는 임베딩 벡터를 찾아낸다. `vocab_size`와 `hidden_size`에 맞춰 임베딩 크기를 설정하고, 단어 임베딩 테이블을 생성한다.
  - 결과: `embedding_output`은 입력된 문장의 임베딩 벡터를 포함한 텐서이며, 이는 모델의 입력으로 사용된다.

임베딩 후처리

```
self.embedding_output = embedding_postprocessor(
    input_tensor=self.embedding_output,
    use_token_type=True,
    token_type_ids=token_type_ids,
```

```
    token_type_vocab_size=config.type_vocab_size,
    token_type_embedding_name="token_type_embeddings",
    use_position_embeddings=True,
    position_embedding_name="position_embeddings",
    initializer_range=config.initializer_range,
    max_position_embeddings=config.max_position_embedding
s,
    dropout_prob=config.hidden_dropout_prob)
```

- `embedding_postprocessor`: 입력된 임베딩에 추가적인 정보를 더한다.
  - **토큰 타입 임베딩**: 문장 구분을 위해 `token_type_ids`를 사용해 각 토큰에 문장 타입을 추가한다.
  - **위치 임베딩**: 토큰의 위치 정보를 추가하여 순서를 고려할 수 있게 한다.
  - **드롭아웃**: 학습 중일 경우, 드롭아웃을 적용해 모델이 과적합되지 않도록 한다.

Transformer 레이어 처리

```
with tf.variable_scope("encoder"):
  attention_mask = create_attention_mask_from_input_mask
(input_ids, input_mask)

  self.all_encoder_layers = transformer_model(
      input_tensor=self.embedding_output,
      attention_mask=attention_mask,
      hidden_size=config.hidden_size,
      num_hidden_layers=config.num_hidden_layers,
      num_attention_heads=config.num_attention_heads,
      intermediate_size=config.intermediate_size,
      intermediate_act_fn=get_activation(config.hidden_ac
t),
      hidden_dropout_prob=config.hidden_dropout_prob,
      attention_probs_dropout_prob=config.attention_probs
_dropout_prob,
      initializer_range=config.initializer_range,
      do_return_all_layers=True,
      adapter_fn=get_adapter(adapter_fn))
```

- `create_attention_mask_from_input_mask`: 2D 마스크(`input_mask`)를 3D 마스크로 변환하여 어텐션 스코어 계산에 사용한다. 이 마스크는 패딩 토큰을 무시하고 유효한 토큰 간의 상호작용만 고려하도록 한다.

- `transformer_model`: BERT 모델의 핵심인 Transformer 레이어를 실행한다.
  - **다중 헤드 어텐션**: 각 토큰이 다른 토큰들과 어떻게 연관되어 있는지 학습한다.
  - **피드포워드 네트워크**: 어텐션을 통해 학습된 정보를 더 깊이 처리한다.
  - `adapter_fn`: Adapter 모듈을 통해 파라미터 효율적인 학습을 지원한다.

요약:

- `with` **문**은 TensorFlow에서 변수의 이름 범위를 정의하는 역할을 하며, 동일한 이름의 변수가 있을 때 이를 구분하는 데 사용된다.
- 이 코드에서는 BERT 모델이 입력 데이터를 임베딩으로 변환한 후, Transformer 레이어를 통해 문맥적 표현을 학습한다. 어텐션 마스크를 적용해 유효한 토큰들 간의 상호작용을 학습하며, 드롭아웃과 Adapter 모듈을 통해 학습을 효율적으로 진행한다.

`modeling_test.py` 파일 분석

1. 파일 개요

`modeling_test.py` 파일은 BERT 모델의 테스트 케이스들을 포함하는 유닛 테스트 파일입니다. 이 파일은 BERT 모델의 구조와 동작이 정상적으로 작동하는지 확인하기 위해 다양한 테스트를 정의하고, `modeling.py`에 있는 모델 구성 요소들이 제대로 구현되었는지 검증합니다. 주요 목표는 `BertModel` 클래스와 관련된 메서드들이 의도한 대로 작동하는지 확인하는 것입니다.

2. BertModelTest 클래스

이 클래스는 `tf.test.TestCase`를 상속받아 TensorFlow 환경에서 테스트를 실행할 수 있도록 설정됩니다.

3. BertModelTester 클래스

이 내부 클래스는 BERT 모델을 생성하고 테스트할 수 있는 설정을 제공합니다. `create_model()` 메서드를 통해 BERT 모델을 실제로 생성하여 각 테스트 케이스에서 호출하고 결과를 검증하는 역할을 합니다.

주요 인수:

- `batch_size`: 테스트에 사용되는 배치 크기.
- `seq_length`: 입력 시퀀스 길이.
- `vocab_size`: 어휘 집합 크기.
- `hidden_size`: 모델의 은닉층 크기.
- `num_hidden_layers`: Transformer 인코더의 은닉층 개수.
- `num_attention_heads`: 어텐션 레이어의 헤드 수.
- `intermediate_size`: 피드포워드 레이어의 크기.
- `hidden_act`: 활성화 함수 (기본값: `gelu`).
- `hidden_dropout_prob`: 은닉층 드롭아웃 확률.
- `attention_probs_dropout_prob`: 어텐션 확률 드롭아웃 비율.
- `max_position_embeddings`: 최대 시퀀스 길이.
- `type_vocab_size`: 토큰 타입 어휘 집합 크기.
- `initializer_range`: 초기 가중치 범위.

```
class BertModelTester(object):
    def __init__(self, parent, batch_size=13, seq_length=
7, is_training=True, ...):
        self.parent = parent
```

```
        self.batch_size = batch_size
        self.seq_length = seq_length
        self.is_training = is_training
        self.vocab_size = vocab_size
        self.hidden_size = hidden_size
        self.num_hidden_layers = num_hidden_layers
        self.num_attention_heads = num_attention_heads
        self.intermediate_size = intermediate_size
        self.hidden_act = hidden_act
        ...
```

4. 모델 생성 및 테스트

`create_model()` 메서드는 입력 텐서를 생성한 후, `BertModel`을 초기화하여 구성합니다. 생성된 모델의 출력값을 가져와 embedding, sequence, pooled output 등을 반환하고, 해당 결과가 예상된 출력 크기와 일치하는지 확인합니다.

```
def create_model(self):
    input_ids = BertModelTest.ids_tensor([self.batch_siz
e, self.seq_length], self.vocab_size)
    input_mask = BertModelTest.ids_tensor([self.batch_siz
e, self.seq_length], vocab_size=2)
    token_type_ids = BertModelTest.ids_tensor([self.batch
_size, self.seq_length], self.type_vocab_size)
    config = modeling.BertConfig(vocab_size=self.vocab_si
ze, hidden_size=self.hidden_size, ...)
    model = modeling.BertModel(config=config, is_training
=self.is_training, input_ids=input_ids, ...)
    outputs = {
        "embedding_output": model.get_embedding_output(),
        "sequence_output": model.get_sequence_output(),
        "pooled_output": model.get_pooled_output(),
        "all_encoder_layers": model.get_all_encoder_layer
s(),
    }
    return outputs
```

5. check_output() 메서드

이 메서드는 모델 출력값의 크기가 올바른지 검증하는 기능을 수행합니다.
`embedding_output`, `sequence_output`, `pooled_output`이 예상한 크기를 가지고 있는지 확인합니다.

```
def check_output(self, result):
    self.parent.assertAllEqual(result["embedding_outpu
t"].shape, [self.batch_size, self.seq_length, self.hidden
_size])
    self.parent.assertAllEqual(result["sequence_output"].
shape, [self.batch_size, self.seq_length, self.hidden_siz
e])
    self.parent.assertAllEqual(result["pooled_output"].sh
ape, [self.batch_size, self.hidden_size])
```

6. 테스트 메서드

`test_default()` 메서드는 기본적인 BERT 모델 생성과 테스트를 실행하며, `test_config_to_json_string()` 메서드는 BERT 설정(`BertConfig`)을 JSON 형식으로 변환하여 설정이 정상적으로 유지되는지 검증합니다.

```
def test_default(self):
    self.run_tester(BertModelTest.BertModelTester(self))

def test_config_to_json_string(self):
    config = modeling.BertConfig(vocab_size=99, hidden_si
ze=37)
    obj = json.loads(config.to_json_string())
    self.assertEqual(obj["vocab_size"], 99)
    self.assertEqual(obj["hidden_size"], 37)
```

7. 기타 유틸리티 메서드

- **ids_tensor()**: 랜덤한 값으로 구성된 Tensor를 생성하여 입력 데이터로 사용합니다.
- **assert_all_tensors_reachable()**: 그래프 상의 모든 텐서가 출력에 도달할 수 있는지 확인하는 유틸리티 메서드입니다.

```
@classmethod
def ids_tensor(cls, shape, vocab_size, rng=None, name=Non
e):
    if rng is None:
        rng = random.Random()
    values = [rng.randint(0, vocab_size - 1) for _ in ran
ge(total_dims)]
```

```
    return tf.constant(value=values, dtype=tf.int32, shap
e=shape, name=name)
```

8. 결론

이 테스트 파일은 BERT 모델의 주요 구성 요소들이 올바르게 동작하는지 검증합니다. `BertModel`을 다양한 설정으로 생성하고, 출력 크기 및 JSON 변환이 예상대로 이루어지는지 확인하여 모델이 제대로 동작하는지 보장합니다.

`optimization.py` (최적화 및 가중치 업데이트 관련 함수들)

1. 파일 개요

`optimization.py` 파일은 BERT 모델을 학습시키기 위한 최적화 함수들을 포함하고 있습니다. 주로 **AdamWeightDecayOptimizer** 클래스를 통해 학습률 조정 및 가중치 감쇠(weight decay)를 관리하고, 훈련 단계를 최적화합니다. 또한, **learning rate scheduling**과 **warmup** 등의 기능을 통해 학습 성능을 최적화합니다.

2. create_optimizer 함수

`create_optimizer` 함수는 학습 손실(loss)을 기반으로 Adam 최적화 기법을 설정합니다. 또한, 학습률 스케줄링과 warmup 단계를 포함하여 학습 중에 학습률이 점진적으로 감소하거나 증가하도록 관리합니다.

```
def create_optimizer(loss, init_lr, num_train_steps, num_
warmup_steps, use_tpu):
  global_step = tf.train.get_or_create_global_step()
  learning_rate = tf.constant(value=init_lr, shape=[], dt
ype=tf.float32)

  # 학습률이 일정하게 감소하도록 설정
  learning_rate = tf.train.polynomial_decay(
      learning_rate,
      global_step,
      num_train_steps,
      end_learning_rate=0.0,
      power=1.0,
      cycle=False)

  # 학습 초기에 warmup 단계를 설정하여 학습률을 점진적으로 증가시킴
  if num_warmup_steps:
    global_steps_int = tf.cast(global_step, tf.int32)
    warmup_steps_int = tf.constant(num_warmup_steps, dtyp
e=tf.int32)
    global_steps_float = tf.cast(global_steps_int, tf.flo
at32)
    warmup_steps_float = tf.cast(warmup_steps_int, tf.flo
at32)
    warmup_percent_done = global_steps_float / warmup_ste
ps_float
    warmup_learning_rate = init_lr * warmup_percent_done
    is_warmup = tf.cast(global_steps_int < warmup_steps_i
nt, tf.float32)
```

```
    learning_rate = (1.0 - is_warmup) * learning_rate + i
s_warmup * warmup_learning_rate

  optimizer = AdamWeightDecayOptimizer(
      learning_rate=learning_rate,
      weight_decay_rate=0.01,
      beta_1=0.9,
      beta_2=0.999,
      epsilon=1e-6,
      exclude_from_weight_decay=["LayerNorm", "bias"])

  if use_tpu:
    optimizer = tf.contrib.tpu.CrossShardOptimizer(optimi
zer)

  tvars = tf.trainable_variables()
  grads = tf.gradients(loss, tvars)
  (grads, _) = tf.clip_by_global_norm(grads, 1.0)

  train_op = optimizer.apply_gradients(zip(grads, tvars),
global_step=global_step)
  new_global_step = global_step + 1
  train_op = tf.group(train_op, [global_step.assign(new_g
lobal_step)])

  return train_op
```

3. AdamWeightDecayOptimizer 클래스

이 클래스는 Adam 옵티마이저에 가중치 감쇠(weight decay)를 추가한 방식입니다. 일반적인 Adam 옵티마이저와의 차이점은 L2 정규화를 통해 가중치를 감쇠시키는 방식이 다르다는 점입니다. 이 방식은 특정 변수들에 대해 가중치 감쇠를 적용하지 않도록 예외 처리할 수도 있습니다.

```
class AdamWeightDecayOptimizer(tf.train.Optimizer):
    def __init__(self, learning_rate, weight_decay_rate=
0.0, beta_1=0.9, beta_2=0.999, epsilon=1e-6, exclude_from
_weight_decay=None, name="AdamWeightDecayOptimizer"):
        super(AdamWeightDecayOptimizer, self).__init__(Fa
lse, name)
        self.learning_rate = learning_rate
        self.weight_decay_rate = weight_decay_rate
```

```
        self.beta_1 = beta_1
        self.beta_2 = beta_2
        self.epsilon = epsilon
        self.exclude_from_weight_decay = exclude_from_wei
ght_decay

    def apply_gradients(self, grads_and_vars, global_step
=None, name=None):
        assignments = []
        for grad, param in grads_and_vars:
            if grad is None or param is None:
                continue
            param_name = self._get_variable_name(param.na
me)
            m = tf.get_variable(name=param_name + "/adam_
m", shape=param.shape.as_list(), dtype=tf.float32, initia
lizer=tf.zeros_initializer())
            v = tf.get_variable(name=param_name + "/adam_
v", shape=param.shape.as_list(), dtype=tf.float32, initia
lizer=tf.zeros_initializer())

            # 표준 Adam 업데이트
            next_m = tf.multiply(self.beta_1, m) + tf.mul
tiply(1.0 - self.beta_1, grad)
            next_v = tf.multiply(self.beta_2, v) + tf.mul
tiply(1.0 - self.beta_2, tf.square(grad))
            update = next_m / (tf.sqrt(next_v) + self.eps
ilon)

            if self._do_use_weight_decay(param_name):
                update += self.weight_decay_rate * param

            update_with_lr = self.learning_rate * update
            next_param = param - update_with_lr
            assignments.extend([param.assign(next_param),
m.assign(next_m), v.assign(next_v)])

        return tf.group(*assignments, name=name)
```

4. 파일의 주요 기능

- **create_optimizer**: 학습률 조정, warmup, 가중치 감쇠를 포함한 최적화 작업을 담당하는 함수.
- **AdamWeightDecayOptimizer**: 표준 Adam 옵티마이저와 다르게 L2 정규화를 통해 가중치를 감쇠시키는 방식 적용.

5. 파일의 역할

이 파일은 BERT 모델을 효율적으로 학습시키기 위해 필요한 다양한 최적화 전략을 제공합니다. Adam 옵티마이저에 가중치 감쇠와 학습률 스케줄링, warmup 같은 기능을 추가하여 모델이 빠르고 안정적으로 학습되도록 돕습니다.

`optimization_test.py` 파일 분석

1. 파일 개요

`optimization_test.py` 파일은 `optimization.py`에서 정의된 최적화 기능을 테스트하는 유닛 테스트 파일입니다. 이 파일에서는 Adam 옵티마이저를 사용하여 가중치를 업데이트하고, 그 결과를 검증하는 테스트가 포함되어 있습니다. 이를 통해 가중치 감쇠와 학습률 조정 등의 최적화 기법이 제대로 작동하는지 확인합니다.

2. `OptimizationTest` 클래스

`OptimizationTest` 클래스는 `tf.test.TestCase`를 상속받아 TensorFlow 환경에서 테스트가 실행되도록 설정합니다.

```
class OptimizationTest(tf.test.TestCase):
```

3. `test_adam()` 메서드

이 메서드는 Adam 옵티마이저를 테스트하기 위한 주요 테스트 케이스입니다. 가중치 변수 `w`를 생성하고, 손실 함수로부터 그래디언트를 계산한 후, AdamWeightDecayOptimizer를 사용해 가중치를 업데이트합니다. 100번의 업데이트 후 가중치가 예상된 값과 일치하는지 확인합니다.

```
def test_adam(self):
    with self.test_session() as sess:
        w = tf.get_variable(
            "w",
            shape=[3],
            initializer=tf.constant_initializer([0.1, -0.
2, -0.1]))
        x = tf.constant([0.4, 0.2, -0.5])
        loss = tf.reduce_mean(tf.square(x - w))
        tvars = tf.trainable_variables()
        grads = tf.gradients(loss, tvars)
        global_step = tf.train.get_or_create_global_step
()
        optimizer = optimization.AdamWeightDecayOptimizer
(learning_rate=0.2)
        train_op = optimizer.apply_gradients(zip(grads, t
vars), global_step)
        init_op = tf.group(tf.global_variables_initialize
r(),
                           tf.local_variables_initializer
())
```

```
        sess.run(init_op)
        for _ in range(100):
            sess.run(train_op)
        w_np = sess.run(w)
        self.assertAllClose(w_np.flat, [0.4, 0.2, -0.5],
rtol=1e-2, atol=1e-2)
```

핵심 동작:

1. **가중치 생성**: 가중치 변수 `w`는 `[0.1, -0.2, -0.1]` 값으로 초기화됩니다.
2. **손실 함수 계산**: `w`와 상수 `x` 사이의 차이의 제곱을 손실 함수로 설정합니다.
3. **최적화 설정**: Adam 옵티마이저와 가중치 감쇠(weight decay)를 적용하는 `AdamWeightDecayOptimizer`가 설정됩니다.
4. **훈련 실행**: 100번의 훈련을 통해 가중치가 업데이트됩니다.
5. **검증**: 가중치가 최종적으로 `[0.4, 0.2, -0.5]`에 근접한지 확인합니다.

4. 결론

이 테스트 파일은 `optimization.py`의 Adam 옵티마이저가 정상적으로 작동하는지 확인하는 데 사용됩니다. 가중치 업데이트 및 손실 함수의 최소화를 통해 최적화가 성공적으로 수행되었는지 검증하며, 이를 통해 모델 훈련이 잘 진행되는지 확인할 수 있습니다.