РЕФЕРАТ

Отчет содержит 47 стр., 1 рис., 2 табл., 2 прил.

Ключевые слова: генерация текста, нейронные сети, NLP, Transformer, дообучение, ruGPT-3.

В работе представлено решение задачи дообучения языковой модели архитектуры Transformer для генерации стилизованного текста из заданной предметной области.

СОДЕРЖАНИЕ

Введение 5
1 Основная часть 6
1.1 Анализ того и сего 6
1.2 Существующие подходы к созданию всячины 6
2 Конструкторский раздел 10
2.1 Архитектура всячины 10
2.1.1 Протестируем подпункт 10
2.1.1.1 А теперь подподпункт 10
2.2 Подсистема всякой ерунды 11
2.2.1 Блок-схема всякой ерунды 11
3 Технологический раздел 12
4 Экспериментальный раздел 14
5 Организационно-экономический раздел 15
5.1 Протестируем специальные символы. 15
6 Промышленная экология и безопасность 16
Заключение 17
Список использованных источников 18
Приложение А Листинги исходного кода 19
Приложение Б Еще картинки 47

ОПРЕДЕЛЕНИЯ

В настоящем отчете о НИР применяют следующие термины с соответствующими определениями.

Обратный прокси — тип прокси-сервера, который ретранслирует.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

MAX — Maximum — максимальное значение параметра.

API — application programming interface — внешний интерфейс взаимодействия с приложением.

ВВЕДЕНИЕ

Целью работы является создание всякой всячины. Для достижения поставленной цели необходимо решить следующие задачи:

— проанализировать существующую всячину;

— спроектировать свою, новую всячину;

— изготовить всякую всячину;

— проверить её работоспособность.

Проверяем как у нас работают сокращения, обозначения и определения — MAX, API с обратным прокси.

1 Основная часть

В данном разделе анализируется и классифицируется существующая всячина и пути создания новой всячины. А вот отступ справа в 1 см. — это хоть и по ГОСТ, но ведь диагноз же...

1.1 Анализ того и сего

В [**?**] указано, что...

Кстати, про картинки. Во-первых, для фигур следует использовать `[ht]`. Если и после этого картинки вставляются «не по ГОСТ», т.е. слишком далеко от места ссылки, — значит у вас в РПЗ **слишком мало текста**! Хотя и ужасный параметр `!ht` у окружения `figure` тоже никто не отменял, только при его использовании документ получается страшный, как в ворде, поэтому просьба так не делать по возможности.

1.2 Существующие подходы к созданию всячины

Известны следующие подходы...

а) Перечисление с номерами.

б) Номера первого уровня. Да, ГОСТ требует именно так — сначала буквы, на втором уровне — цифры. Чуть ниже будет вариант «нормальной» нумерации и советы по её изменению. Да, мне так нравится: на первом уровне выравнивание элементов как у обычных абзацев. Проверим теперь вложенные списки.

1) Номера второго уровня.

2) Номера второго уровня. Проверяем на длиииином-предлиииииииииииинной строке, что получается.... Сойдёт.

в) По мнению Лукьяненко, человеческий мозг старается подвести любую проблему к выбору из трех вариантов.

г) Четвёртый (и последний) элемент списка.

Теперь мы покажем, как изменить нумерацию на «нормальную», если вам этого захочется. Пара команд в начале документа поможет нам.

1) Изменим нумерацию на более привычную...

2) ... нарушим этим гост.

   а) Но, пожалуй, так лучше.

В заключение покажем произвольные маркеры в списках. Для них нужен пакет **enumerate**.

1. Маркер с арабской цифрой и с точкой.

2. Маркер с арабской цифрой и с точкой.

I. Римская цифра с точкой.

II. Римская цифра с точкой.

В отчётах могут быть и таблицы — см. табл. 1.1 и 1.2. Небольшая таблица делается при помощи **tabular** внутри **table** (последний полностью аналогичен **figure**, но добавляет другую подпись).

Таблица 1.1 — Пример короткой таблицы с коротким названием

Тело	F	V	E	$F+V-E-2$
Тетраэдр	4	4	6	0
Куб	6	8	12	0
Октаэдр	8	6	12	0
Додекаэдр	20	12	30	0
Икосаэдр	12	20	30	0
Эйлер	666	9000	42	$+\infty$

Для больших таблиц следует использовать пакет **longtable**, позволяющий создавать таблицы на несколько страниц по ГОСТ.

Для того, чтобы длинный текст разбивался на много строк в пределах одной ячейки, надо в качестве ее формата задавать `p` и указывать явно ширину: в мм/дюймах (`110mm`), относительно ширины страницы (`0.22\textwidth`) и т.п.

Можно также использовать уменьшенный шрифт — но, пожалуйста, тогда уж во **всей** таблице сразу.

Таблица 1.2 — Пример длинной таблицы с длинным названием на много длинных-длинных строк

Вид шума	Громкость, дБ	Комментарий
Порог слышимости	0	
Шепот в тихой библиотеке	30	
Обычный разговор	60-70	
Звонок телефона	80	Конечно, это было до эпохи мобильников
Уличный шум	85	(внутри машины)
Гудок поезда	90	
Шум электрички	95	
Порог здоровой нормы	90-95	Длительное пребывание на более громком шуме может привести к ухудшению слуха
Мотоцикл	100	
Power Mower	107	(модель бензокосилки)
Бензопила	110	(Doom в целом вреден для здоровья)
Рок-концерт	115	
Порог боли	125	feel the pain
Клепальный молоток	125	(автор сам не знает, что это)
Порог опасности	140	Даже кратковременное пребывание на шуме большего уровня может привести к необратимым последствиям
Реактивный двигатель	140	
	180	Необратимое полное повреждение слуховых органов

Продолжение на след. стр.

Продолжение таблицы 1.2

Самый громкий возможный звук	194	Интересно, почему?..

2 Конструкторский раздел

В данном разделе проектируется новая всячина.

2.1 Архитектура всячины

2.1.1 Протестируем подпункт

2.1.1.1 А теперь подподпункт

Проверка параграфа. Вроде работает.

Вторая проверка параграфа. Опять работает.

Вот.

— Это список с «палочками».

— Хотя он и по ГОСТ, но…

1) Для списка, начинающегося с заглавной буквы, лучше список с цифрами.

Формула (2.1) совершено бессмысленна.

$$a = cb \tag{2.1}$$

А формула (2.2) имеет некоторый смысл. Кроме этого она пытается иллюстрировать применение окружения **eqndesc** которое размещает формулу совместно с её описанием. Однако обратите внимание на нумерацию формул (2.2) и (2.3), попробуйте добавить **[H]** к такой формуле.

Окружение `cases` опять работает (см. (2.3)), спасибо И. Короткову за исправления..

$$a = \begin{cases} 3x + 5y + z, \text{если хорошо} \\ 7x - 2y + 4z, \text{если плохо} \\ -6x + 3y + 2z, \text{если совсем плохо} \end{cases} \tag{2.3}$$

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{+\infty} A_k \cos\left(k\frac{2\pi}{\tau}x + \theta_k\right) \tag{2.2}$$

где A_k — амплитуда k-го гармонического колебания,
A_k — амплитуда k-го гармонического колебания,
$k\frac{2\pi}{\tau} = k\omega$ — круговая частота гармонического колебания,
θ_k — начальная фаза k-го колебания.

2.2 Подсистема всякой ерунды

2.2.1 Блок-схема всякой ерунды

Кстати о заголовках

У нас есть и **subsubsection**. Только лучше её не нумеровать.

В данном разделе описано изготовление и требование всячины. Кстати, в Latex нужно эскейпить подчёркивание (писать «`some\_function`» для **some_function**).

Для вставки кода есть пакет `minted`. Он хорош всем кроме: необходимости Python (есть во всех нормальных (нет, Windows, я не про тебя) ОС) и Pygments и того, что нормально работает лишь в XeLaTeX.

Можно пользоваться расширенным BFN:

```
 letter = "A" | "B" | "C" | "D" | "E" | "F" | "G"
       | "H" | "I" | "J" | "K" | "L" | "M" | "N"
       | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
       | "V" | "W" | "X" | "Y" | "Z" ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
symbol = "[" | "]" | "{" | "}" | "(" | ")" | "<" | ">"
       | "'" | '"' | "=" | "|" | "." | "," | ";" ;
character = letter | digit | symbol | "_" ;

identifier = letter , { letter | digit | "_" } ;
terminal = "'" , character , { character } , "'"
         | '"' , character , { character } , '"' ;

lhs = identifier ;
rhs = identifier
     | terminal
     | "[" , rhs , "]"
     | "{" , rhs , "}"
     | "(" , rhs , ")"
     | rhs , "|" , rhs
     | rhs , "," , rhs ;

rule = lhs , "=" , rhs , ";" ;
grammar = { rule } ;
```

Листинг 1 — EBNF определённый через EBNF

Можно также использовать окружение **verbatim**, если **listings** чем-то не устраивает. Только следует помнить, что табы в нём «съедаются». Существует так же команда **\verbatiminput** для вставки файла.

```
a_b = a + b; // русский комментарий
if (a_b > 0)
    a_b = 0;
```

4 Экспериментальный раздел

В данном разделе проводятся вычислительные эксперименты. А на рис. **??** показана схема мыслительного процесса автора...

5 Организационно-экономический раздел

5.1 Протестируем специальные символы.

И заодно переключение шрифтов.

"--* Прямая речь "--- <<после ,,тире'' неразрывный пробел>>

cyrillicfonttt — Прямая речь — «после „тире“ неразрывный пробел».

cyrillicfontsf — Прямая речь — «после „тире“ неразрывный пробел».

\cyrillicfont — Прямая речь — «после „тире“ неразрывный пробел».

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

6 Промышленная экология и безопасность

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

1) First itemtext
    а) First itemtext
        — First itemtext
        — Second itemtext
        — Last itemtext
        — First itemtext
        — Second itemtext
    б) Last itemtext
    в) First itemtext
    г) Second itemtext
    д) Last itemtext
2) First itemtext
3) Second itemtext
4) Last itemtext
5) First itemtext

ЗАКЛЮЧЕНИЕ

В результате проделанной работы стало ясно, что ничего не ясно...

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А

ЛИСТИНГИ ИСХОДНОГО КОДА

Листинг А.1 — Графический интерфейс пользователя

```
import streamlit as st

from generate import load_tokenizer_and_model, generate, CACHE_DIR


def initialize() -> None:
    """Initialize session state and set page config"""

    st.set_page_config(
        page_title="Autoplot AI",
        page_icon="□",
        layout="wide",
        initial_sidebar_state="collapsed"
    )

    if "model" not in st.session_state or "tokenizer" not in st.session_state:
        with st.spinner("Loading model"):
            tokenizer, model = load_tokenizer_and_model(CACHE_DIR)
            st.session_state["tokenizer"] = tokenizer
            st.session_state["model"] = model

    if "text_versions" not in st.session_state:
        st.session_state["text_versions"] = [""]


def main() -> None:
    """User interface logic"""

    text_versions = st.session_state["text_versions"]
    tokenizer = st.session_state["tokenizer"]
    model = st.session_state["model"]

    button_cols = st.columns(3)
    with button_cols[0]:
        continue_btn = st.button("Дополнить")

    with button_cols[1]:
        undo_btn = st.button("Отменить")

    with button_cols[2]:
        st.download_button("Скачать результат", text_versions[-1], "result.txt")
```

```
42
43      text_container = st.empty()
44      text_area_attrs = {"label": "Текст", "height": 500}
45
46      with text_container:
47          working_text = st.text_area(value=text_versions[-1],
                **text_area_attrs)
48
49      if continue_btn:
50          if len(working_text) == 0:
51              working_text = "Место действия — "
52
53          working_text = working_text[:-100] + generate(model, tokenizer,
                working_text[-100:])[0]
54
55          with text_container:
56              st.text_area(value=working_text, **text_area_attrs)
57
58      if text_versions[-1] != working_text:
59          text_versions.append(working_text)
60          st.experimental_rerun()
61
62      if undo_btn and len(text_versions) > 1:
63          text_versions.pop()
64          working_text = text_versions[-1]
65          with text_container:
66              st.text_area(value=working_text, **text_area_attrs)
67
68
69  if __name__ == "__main__":
70      initialize()
71      main()
```

Листинг А.2 — Модуль генерации текста

```
1   import time
2   import os
3   import sys
4   import random
5
6   from zipfile import ZipFile
7
8   import numpy as np
9   import torch
10
11  from transformers import GPT2LMHeadModel, GPT2Tokenizer
12
```

```
13
14 USE_CUDA = True
15 CACHE_DIR = os.path.join(os.curdir, "model_cache")
16 SEED = random.randint(0, 1000)
17
18 if not os.path.isdir(CACHE_DIR):
19     print("Extracting model...")
20     with ZipFile("model.zip") as f:
21         f.extractall(CACHE_DIR)
22
23 device = "cuda" if torch.cuda.is_available() and USE_CUDA else "cpu"
24
25 print(f"Running on {device}")
26
27
28 def load_tokenizer_and_model(model_name_or_path):
29     print("Loading tokenizer and model from " + CACHE_DIR)
30     tokenizer = GPT2Tokenizer.from_pretrained(model_name_or_path)
31     model = GPT2LMHeadModel.from_pretrained(model_name_or_path).to(device)
32     return tokenizer, model
33
34
35 def generate(
36     model, tok, text,
37     do_sample=True, max_length=50, repetition_penalty=5.0,
38     top_k=5, top_p=0.95, temperature=1,
39     num_beams=None,
40     no_repeat_ngram_size=3
41     ):
42     input_ids = tok.encode(text, return_tensors="pt").to(device)
43     out = model.generate(
44         input_ids.to(device),
45         max_length=max_length,
46         repetition_penalty=repetition_penalty,
47         do_sample=do_sample,
48         top_k=top_k, top_p=top_p, temperature=temperature,
49         num_beams=num_beams, no_repeat_ngram_size=no_repeat_ngram_size
50     )
51     return list(map(tok.decode, out))
52
53
54 def main(beginning):
55     np.random.seed(SEED)
56     torch.manual_seed(SEED)
57
58     tok, model = load_tokenizer_and_model(CACHE_DIR)
```

```
59
60      print("Generating")
61      prev_timestamp = time.time()
62      generated = generate(model, tok, beginning, max_length=200, top_p=0.95,
            temperature=0.7)
63      time_spent = time.time() - prev_timestamp
64
65      print(generated[0])
66
67      print(f"Elapsed time: {time_spent} s.")
68
69
70  if __name__ == "__main__":
71      main(sys.argv[1])
```

Листинг А.3 — Скрипт для валидации и обработки данных

```
1   import os
2   import re
3   import shutil
4
5   from collections import namedtuple
6   from sys import argv
7   from typing import List, Union
8
9
10  DATA_PATH = argv[1]
11  if DATA_PATH[-1] != "/":
12      DATA_PATH += "/"
13
14  OUT_PATH = "humanized"
15
16  Block = namedtuple("Block", ["tag", "content"])
17
18
19  def parse(text: str) -> List[Union[Block, str]]:
20
21      text = re.sub("<<", "«", text)
22      text = re.sub(">>", "»", text)
23      text = re.sub(r"\s+|\n", " ", text)
24      s = re.sub(r"(</?\w+>)", r"[CUT]\1[CUT]", text)
25      cut = list(filter(lambda t: len(t) > 0, map(str.strip, s.split("[CUT]"))))
26
27      open_tag_pat = re.compile(r"<\w+>")
28
29      cur_errors = []
30
```

```
31      def parse_list(l: List[str]) -> List[Union[Block, str]]:
32          it = iter(l)
33          out = []
34          while True:
35              try:
36                  el = next(it)
37              except StopIteration:
38                  break
39
40              if re.match(open_tag_pat, el):
41                  tag = el[1:-1]
42                  next_it = []
43                  while not re.match(f"<\W{tag}>", el):
44                      try:
45                          el = next(it)
46                      except StopIteration:
47                          # raise SyntaxError(f"<{tag}> was not closed:
                                {out[-1]}; {' '.join(next_it)}")
48                          cur_errors.append(f"<{tag}> was not closed: {out[-1]
                                if len(out) > 0 else ''}; {' '.join(next_it)}")
49                          break
50                      else:
51                          next_it.append(el)
52                  if len(next_it) > 0:
53                      out.append(Block(tag, parse_list(next_it[:-1])))
54              else:
55                  out.append(el)
56
57          for e in out:
58              if isinstance(e, str) and re.match(r"</?.+>", e):
59                  i = out.index(e)
60                  cur_errors.append(f"Found tag in processed data: {e};
                        {out[max(i - 2, 0):i + 1]}")
61          return out
62
63      return parse_list(cut), cur_errors
64
65
66  cur_name = ""
67  def humanize(s: List[Union[Block, str]]) -> str:
68      sentences = []
69
70      global cur_name
71
72      for el in s:
73          if isinstance(el, str):
```

```
74              sentences.append(el.strip())
75          else:
76              if el.tag == "header":
77                  continue
78              elif el.tag == "footer":
79                  continue
80              elif el.tag == "remark":
81                  sentences += ["\n" + "Ремарка —", humanize(el.content)]
82              elif el.tag == "author":
83                  sentences += ["\n" + "Слова автора —", humanize(el.content)]
84              elif el.tag == "title":
85                  sentences += ["\n\n" + "Заголовок —", humanize(el.content),
                        "\n"]
86              elif el.tag == "place":
87                  sentences += ["\n" + "Место действия —",
                        humanize(el.content).strip(".") + "."]
88              elif el.tag == "time":
89                  sentences += ["\n" + "Время действия —",
                        humanize(el.content).strip(".").lower() + "."]
90              elif el.tag == "chars":
91                  sentences += ["\n" + "Действующие лица —",
                        humanize(el.content).strip(".") + "."]
92              elif el.tag == "name":
93                  cur_name = humanize(el.content).strip().capitalize()
94              elif el.tag == "line":
95                  sentences += ["\n" + cur_name, "говорит:", "«" +
                        humanize(el.content).strip(".") + "»"]
96              elif el.tag == "how":
97                  sentences.append(humanize(el.content).lower())
98
99      sentences = filter(lambda t: not re.match(r"^\W*$", t) or t == "\n",
            sentences)
100     sentences = " ".join(sentences)
101     if sentences[0] == "\n":
102         sentences = sentences[1:]
103     return sentences
104
105
106 if __name__ == "__main__":
107     paths = []
108     for root, _, files in os.walk(DATA_PATH):
109         for file in files:
110             paths.append(os.path.join(root, file))
111
112     parsed = []
113     errors = []
```

```
114     for path in paths:
115         with open(path) as file:
116             try:
117                 content = file.read()
118             except Exception as e:
119                 print(e, path)
120                 raise
121 
122         try:
123             t, e = parse(content)
124             if len(e) > 0:
125                 raise SyntaxError("\n\n\n".join(e))
126             parsed.append((path, t))
127         except SyntaxError as e:
128             errors.append((path, e))
129             continue
130 
131     print(f"Errors occured in {len(errors)} files")
132     print(f"Successfully parsed {len(parsed)} files")
133 
134     if os.path.isdir(os.path.join("errors", "data")):
135         for f in os.listdir(os.path.join("errors", "data")):
136             os.remove(os.path.join(os.path.join("errors", "data"), f))
137     for p, e in errors:
138         os.makedirs(os.path.join("errors", "data"), exist_ok=True)
139         path = os.path.join("errors", "data", os.path.split(p)[-1])
140         with open(path + ".log", "w") as f:
141             f.write(str(e))
142 
143         os.system(f"cp '{p}' '{path}'")
144         # break
145 
146     if os.path.isdir(OUT_PATH):
147         for f in os.listdir(OUT_PATH):
148             os.remove(os.path.join(OUT_PATH, f))
149     os.makedirs(OUT_PATH, exist_ok=True)
150 
151     for i, (path, script) in enumerate(parsed):
152         text = humanize(script)
153 
154         new_path = os.path.join(OUT_PATH, re.sub(DATA_PATH, "", path))
155         os.makedirs(os.path.split(new_path)[0], exist_ok=True)
156         with open(new_path, "w") as f:
157             f.write(text)
158 
159     if os.path.isdir("invalid_files"):
```

```
160             for f in os.listdir("invalid_files"):
161                 os.remove(os.path.join("invalid_files", f))
162         for path, _ in errors:
163             os.makedirs("invalid_files", exist_ok=True)
164             shutil.copy(path, "invalid_files")
```

Листинг А.4 — Скрипт для обучения модели

```
1  # coding=utf-8
2  # Copyright 2018 The Google AI Language Team Authors and The HuggingFace Inc.
       team.
3  # Copyright (c) 2018, NVIDIA CORPORATION.  All rights reserved.
4  #
5  # Licensed under the Apache License, Version 2.0 (the "License");
6  # you may not use this file except in compliance with the License.
7  # You may obtain a copy of the License at
8  #
9  #     http://www.apache.org/licenses/LICENSE-2.0
10 #
11 # Unless required by applicable law or agreed to in writing, software
12 # distributed under the License is distributed on an "AS IS" BASIS,
13 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 # See the License for the specific language governing permissions and
15 # limitations under the License.
16
17 import argparse
18 import glob
19 import logging
20 import os
21 import pickle
22 import random
23 import re
24 import shutil
25 import sys
26 from typing import Dict, List, Tuple
27
28 import numpy as np
29 import torch
30 from torch.nn.utils.rnn import pad_sequence
31 from torch.utils.data import DataLoader, Dataset, RandomSampler,
       SequentialSampler
32 from torch.utils.data.distributed import DistributedSampler
33 from tqdm import tqdm, trange
34 from transformers import (
35     MODEL_WITH_LM_HEAD_MAPPING,
36     WEIGHTS_NAME,
37     AdamW,
```

```
38      AutoConfig,
39      AutoModelWithLMHead,
40      AutoTokenizer,
41      PreTrainedModel,
42      PreTrainedTokenizer,
43      get_linear_schedule_with_warmup,
44  )
45  from torch.utils.tensorboard import SummaryWriter
46
47  logger = logging.getLogger(__name__)
48  logger.setLevel(logging.DEBUG)
49  handler = logging.StreamHandler(sys.stdout)
50  handler.setLevel(logging.DEBUG)
51  log_format = logging.Formatter('[%(asctime)s] [%(levelname)s] - %(message)s')
52  handler.setFormatter(log_format)
53  logger.addHandler(handler)
54
55  MODEL_CONFIG_CLASSES = list(MODEL_WITH_LM_HEAD_MAPPING.keys())
56  MODEL_TYPES = tuple(conf.model_type for conf in MODEL_CONFIG_CLASSES)
57
58
59  class TextsDataset(Dataset):
60      """Texts one by one"""
61
62      def __init__(self, tokenizer: PreTrainedTokenizer, path: str,
            block_size=2048):
63          assert os.path.isdir(path)
64
65          block_size = block_size - (tokenizer.max_len -
                tokenizer.max_len_single_sentence)
66
67          logger.info("Creating features from dataset file at %s", path)
68
69          self.examples = []
70          try:
71              for file in os.listdir(path):
72                  file_path = os.path.join(path, file)
73                  with open(file_path, encoding="utf-8") as f:
74                      text = f.read()
75
76                  tokenized_text =
                        tokenizer.convert_tokens_to_ids(tokenizer.tokenize(text))
77
78                  logger.info(f"Tokenized {file_path}: tokens len:
                        {len(tokenized_text)}")
79
```

```
80                  for i in range(0, len(tokenized_text) - block_size + 1,
                        block_size):  # Truncate in block of block_size
81                      self.examples.append(tokenizer.build_inputs_with_special_tokens(to
                            i + block_size]))
82          except Exception as e:
83              logger.exception(e)
84
85          logger.info(f"Created dataset of size {len(self.examples)}")
86
87      def __len__(self):
88          return len(self.examples)
89
90      def __getitem__(self, item):
91          return torch.tensor(self.examples[item], dtype=torch.long)
92
93
94  def load_and_cache_examples(args, tokenizer, evaluate=False):
95      file_path = args.eval_data_file if evaluate else args.train_data_file
96      return TextsDataset(tokenizer, file_path, block_size=args.block_size)
97
98
99  def set_seed(args):
100     random.seed(args.seed)
101     np.random.seed(args.seed)
102     torch.manual_seed(args.seed)
103     if args.n_gpu > 0:
104         torch.cuda.manual_seed_all(args.seed)
105
106
107 def _sorted_checkpoints(args, checkpoint_prefix="checkpoint",
        use_mtime=False) -> List[str]:
108     ordering_and_checkpoint_path = []
109
110     glob_checkpoints = glob.glob(os.path.join(args.output_dir,
            "{}-*".format(checkpoint_prefix)))
111
112     for path in glob_checkpoints:
113         if use_mtime:
114             ordering_and_checkpoint_path.append((os.path.getmtime(path),
                    path))
115         else:
116             regex_match = re.match(".*{}-([0-9]+)".format(checkpoint_prefix),
                    path)
117             if regex_match and regex_match.groups():
118                 ordering_and_checkpoint_path.append((int(regex_match.groups()[0]),
                        path))
```

```
119
120     checkpoints_sorted = sorted(ordering_and_checkpoint_path)
121     checkpoints_sorted = [checkpoint[1] for checkpoint in checkpoints_sorted]
122     return checkpoints_sorted
123
124
125 def _rotate_checkpoints(args, checkpoint_prefix="checkpoint",
        use_mtime=False) -> None:
126     if not args.save_total_limit:
127         return
128     if args.save_total_limit <= 0:
129         return
130
131     # Check if we should delete older checkpoint(s)
132     checkpoints_sorted = _sorted_checkpoints(args, checkpoint_prefix,
            use_mtime)
133     if len(checkpoints_sorted) <= args.save_total_limit:
134         return
135
136     number_of_checkpoints_to_delete = max(0, len(checkpoints_sorted) -
            args.save_total_limit)
137     checkpoints_to_be_deleted =
            checkpoints_sorted[:number_of_checkpoints_to_delete]
138     for checkpoint in checkpoints_to_be_deleted:
139         logger.info("Deleting older checkpoint [{}] due to
                args.save_total_limit".format(checkpoint))
140         shutil.rmtree(checkpoint)
141
142
143 def mask_tokens(inputs: torch.Tensor, tokenizer: PreTrainedTokenizer, args)
        -> Tuple[torch.Tensor, torch.Tensor]:
144     """ Prepare masked tokens inputs/labels for masked language modeling: 80%
            MASK, 10% random, 10% original. """
145
146     if tokenizer.mask_token is None:
147         raise ValueError(
148             "This tokenizer does not have a mask token which is necessary for
                    masked language modeling. Remove the --mlm flag if you want
                    to use this tokenizer."
149         )
150
151     labels = inputs.clone()
152     # We sample a few tokens in each sequence for masked-LM training
153     # (with probability args.mlm_probability defaults to 0.15 in Bert/RoBERTa)
154     probability_matrix = torch.full(labels.shape, args.mlm_probability)
155     special_tokens_mask = [
```

```
156             tokenizer.get_special_tokens_mask(val,
                    already_has_special_tokens=True) for val in labels.tolist()
157         ]
158         probability_matrix.masked_fill_(torch.tensor(special_tokens_mask,
                dtype=torch.bool), value=0.0)
159         if tokenizer._pad_token is not None:
160             padding_mask = labels.eq(tokenizer.pad_token_id)
161             probability_matrix.masked_fill_(padding_mask, value=0.0)
162         masked_indices = torch.bernoulli(probability_matrix).bool()
163         labels[~masked_indices] = -100  # We only compute loss on masked tokens
164 
165         # 80% of the time, we replace masked input tokens with
                tokenizer.mask_token ([MASK])
166         indices_replaced = torch.bernoulli(torch.full(labels.shape, 0.8)).bool()
                & masked_indices
167         inputs[indices_replaced] =
                tokenizer.convert_tokens_to_ids(tokenizer.mask_token)
168 
169         # 10% of the time, we replace masked input tokens with random word
170         indices_random = torch.bernoulli(torch.full(labels.shape, 0.5)).bool() &
                masked_indices & ~indices_replaced
171         random_words = torch.randint(len(tokenizer), labels.shape,
                dtype=torch.long)
172         inputs[indices_random] = random_words[indices_random]
173 
174         # The rest of the time (10% of the time) we keep the masked input tokens
                unchanged
175         return inputs, labels
176 
177 
178 def train(args, train_dataset, model: PreTrainedModel, tokenizer:
        PreTrainedTokenizer) -> Tuple[int, float]:
179         """ Train the model """
180         if args.local_rank in [-1, 0]:
181             tb_writer = SummaryWriter()
182 
183         args.train_batch_size = args.per_gpu_train_batch_size * max(1, args.n_gpu)
184 
185         def collate(examples: List[torch.Tensor]):
186             if tokenizer._pad_token is None:
187                 return pad_sequence(examples, batch_first=True)
188             return pad_sequence(examples, batch_first=True,
                    padding_value=tokenizer.pad_token_id)
189 
190         train_sampler = RandomSampler(train_dataset) if args.local_rank == -1
                else DistributedSampler(train_dataset)
```

```
    train_dataloader = DataLoader(
        train_dataset, sampler=train_sampler,
            batch_size=args.train_batch_size, collate_fn=collate
    )

    if args.max_steps > 0:
        t_total = args.max_steps
        args.num_train_epochs = args.max_steps // (len(train_dataloader) //
            args.gradient_accumulation_steps) + 1
    else:
        t_total = len(train_dataloader) // args.gradient_accumulation_steps *
            args.num_train_epochs

    model = model.module if hasattr(model, "module") else model  # Take care
        of distributed/parallel training
    model.resize_token_embeddings(len(tokenizer))

    # Prepare optimizer and schedule (linear warmup and decay)
    no_decay = ["bias", "LayerNorm.weight"]
    optimizer_grouped_parameters = [
        {
            "params": [p for n, p in model.named_parameters() if not any(nd
                in n for nd in no_decay)],
            "weight_decay": args.weight_decay,
        },
        {"params": [p for n, p in model.named_parameters() if any(nd in n for
            nd in no_decay)], "weight_decay": 0.0},
    ]
    optimizer = AdamW(optimizer_grouped_parameters, lr=args.learning_rate,
        eps=args.adam_epsilon)
    scheduler = get_linear_schedule_with_warmup(
        optimizer, num_warmup_steps=args.warmup_steps,
            num_training_steps=t_total
    )

    # Check if saved optimizer or scheduler states exist
    if (
            args.model_name_or_path
            and os.path.isfile(os.path.join(args.model_name_or_path,
                "optimizer.pt"))
            and os.path.isfile(os.path.join(args.model_name_or_path,
                "scheduler.pt"))
    ):
        # Load in optimizer and scheduler states
        optimizer.load_state_dict(torch.load(os.path.join(args.model_name_or_path,
            "optimizer.pt")))
```

```
226         scheduler.load_state_dict(torch.load(os.path.join(args.model_name_or_path,
                "scheduler.pt")))
227
228     if args.fp16:
229         try:
230             from apex import amp
231         except ImportError:
232             raise ImportError("Please install apex from
                    https://www.github.com/nvidia/apex to use fp16 training.")
233         model, optimizer = amp.initialize(model, optimizer,
                opt_level=args.fp16_opt_level)
234
235     # multi-gpu training (should be after apex fp16 initialization)
236     if args.n_gpu > 1:
237         model = torch.nn.DataParallel(model)
238
239     # Distributed training (should be after apex fp16 initialization)
240     if args.local_rank != -1:
241         model = torch.nn.parallel.DistributedDataParallel(
242             model, device_ids=[args.local_rank],
                    output_device=args.local_rank, find_unused_parameters=True
243         )
244
245     # Train!
246     logger.info("***** Running training *****")
247     logger.info("  Num examples = %d", len(train_dataset))
248     logger.info("  Num Epochs = %d", args.num_train_epochs)
249     logger.info("  Instantaneous batch size per GPU = %d",
            args.per_gpu_train_batch_size)
250     logger.info(
251         "  Total train batch size (w. parallel, distributed & accumulation) =
                %d",
252         args.train_batch_size
253         * args.gradient_accumulation_steps
254         * (torch.distributed.get_world_size() if args.local_rank != -1 else
                1),
255     )
256     logger.info("  Gradient Accumulation steps = %d",
            args.gradient_accumulation_steps)
257     logger.info("  Total optimization steps = %d", t_total)
258
259     global_step = 0
260     epochs_trained = 0
261     steps_trained_in_current_epoch = 0
262     # Check if continuing training from a checkpoint
263     if args.model_name_or_path and os.path.exists(args.model_name_or_path):
```

```
264         try:
265             # set global_step to gobal_step of last saved checkpoint from
                    model path
266             checkpoint_suffix =
                    args.model_name_or_path.split("-")[-1].split("/")[0]
267             global_step = int(checkpoint_suffix)
268             epochs_trained = global_step // (len(train_dataloader) //
                    args.gradient_accumulation_steps)
269             steps_trained_in_current_epoch = global_step %
                    (len(train_dataloader) // args.gradient_accumulation_steps)
270
271             logger.info("  Continuing training from checkpoint, will skip to
                    saved global_step")
272             logger.info("  Continuing training from epoch %d", epochs_trained)
273             logger.info("  Continuing training from global step %d",
                    global_step)
274             logger.info("  Will skip the first %d steps in the first epoch",
                    steps_trained_in_current_epoch)
275         except ValueError:
276             logger.info("  Starting fine-tuning.")
277
278     tr_loss, logging_loss = 0.0, 0.0
279
280     model.zero_grad()
281     train_iterator = trange(
282         epochs_trained, int(args.num_train_epochs), desc="Epoch",
                disable=args.local_rank not in [-1, 0]
283     )
284     set_seed(args)  # Added here for reproducibility
285     for _ in train_iterator:
286         epoch_iterator = tqdm(train_dataloader, desc="Iteration",
                disable=args.local_rank not in [-1, 0])
287         for step, batch in enumerate(epoch_iterator):
288
289             # Skip past any already trained steps if resuming training
290             if steps_trained_in_current_epoch > 0:
291                 steps_trained_in_current_epoch -= 1
292                 continue
293
294             inputs, labels = mask_tokens(batch, tokenizer, args) if args.mlm
                    else (batch, batch)
295             inputs = inputs.to(args.device)
296             labels = labels.to(args.device)
297             model.train()
298             outputs = model(inputs, masked_lm_labels=labels) if args.mlm else
                    model(inputs, labels=labels)
```

```
299             loss = outputs[0]  # model outputs are always tuple in
                    transformers (see doc)
300
301             if args.n_gpu > 1:
302                 loss = loss.mean()  # mean() to average on multi-gpu parallel
                        training
303             if args.gradient_accumulation_steps > 1:
304                 loss = loss / args.gradient_accumulation_steps
305
306             if args.fp16:
307                 with amp.scale_loss(loss, optimizer) as scaled_loss:
308                     scaled_loss.backward()
309             else:
310                 loss.backward()
311
312             tr_loss += loss.item()
313             if (step + 1) % args.gradient_accumulation_steps == 0:
314                 if args.fp16:
315                     torch.nn.utils.clip_grad_norm_(amp.master_params(optimizer),
                            args.max_grad_norm)
316                 else:
317                     torch.nn.utils.clip_grad_norm_(model.parameters(),
                            args.max_grad_norm)
318                 optimizer.step()
319                 scheduler.step()  # Update learning rate schedule
320                 model.zero_grad()
321                 global_step += 1
322
323                 if args.local_rank in [-1, 0] and args.logging_steps > 0 and
                        global_step % args.logging_steps == 0:
324                     # Log metrics
325                     if (
326                             args.local_rank == -1 and
                                args.evaluate_during_training
327                     ):  # Only evaluate when single GPU otherwise metrics may
                            not average well
328                         results = evaluate(args, model, tokenizer)
329                         for key, value in results.items():
330                             tb_writer.add_scalar("eval_{}".format(key),
                                    value, global_step)
331                     tb_writer.add_scalar("lr", scheduler.get_lr()[0],
                            global_step)
332                     tb_writer.add_scalar("loss", (tr_loss - logging_loss) /
                            args.logging_steps, global_step)
333                     logging_loss = tr_loss
334
```

```
335                 if args.local_rank in [-1, 0] and args.save_steps > 0 and
                        global_step % args.save_steps == 0:
336                     checkpoint_prefix = "checkpoint"
337                     # Save model checkpoint
338                     output_dir = os.path.join(args.output_dir,
                            "{}-{}".format(checkpoint_prefix, global_step))
339                     os.makedirs(output_dir, exist_ok=True)
340                     model_to_save = (
341                         model.module if hasattr(model, "module") else model
342                     )  # Take care of distributed/parallel training
343                     model_to_save.save_pretrained(output_dir)
344                     tokenizer.save_pretrained(output_dir)
345
346                     torch.save(args, os.path.join(output_dir,
                            "training_args.bin"))
347                     logger.info("Saving model checkpoint to %s", output_dir)
348
349                     _rotate_checkpoints(args, checkpoint_prefix)
350
351                     torch.save(optimizer.state_dict(),
                            os.path.join(output_dir, "optimizer.pt"))
352                     torch.save(scheduler.state_dict(),
                            os.path.join(output_dir, "scheduler.pt"))
353                     logger.info("Saving optimizer and scheduler states to
                            %s", output_dir)
354
355             if 0 < args.max_steps < global_step:
356                 epoch_iterator.close()
357                 break
358         if 0 < args.max_steps < global_step:
359             train_iterator.close()
360             break
361
362     if args.local_rank in [-1, 0]:
363         tb_writer.close()
364
365     return global_step, tr_loss / global_step
366
367
368 def evaluate(args, model: PreTrainedModel, tokenizer: PreTrainedTokenizer,
        prefix="") -> Dict:
369     # Loop to handle MNLI double evaluation (matched, mis-matched)
370     eval_output_dir = args.output_dir
371
372     eval_dataset = load_and_cache_examples(args, tokenizer, evaluate=True)
373
```

```
374     if args.local_rank in [-1, 0]:
375         os.makedirs(eval_output_dir, exist_ok=True)
376
377     args.eval_batch_size = args.per_gpu_eval_batch_size * max(1, args.n_gpu)
378
379     # Note that DistributedSampler samples randomly
380
381     def collate(examples: List[torch.Tensor]):
382         if tokenizer._pad_token is None:
383             return pad_sequence(examples, batch_first=True)
384         return pad_sequence(examples, batch_first=True,
                padding_value=tokenizer.pad_token_id)
385
386     eval_sampler = SequentialSampler(eval_dataset)
387     eval_dataloader = DataLoader(
388         eval_dataset, sampler=eval_sampler, batch_size=args.eval_batch_size,
            collate_fn=collate
389     )
390
391     # multi-gpu evaluate
392     if args.n_gpu > 1:
393         model = torch.nn.DataParallel(model)
394
395     # Eval!
396     logger.info("***** Running evaluation {} *****".format(prefix))
397     logger.info("  Num examples = %d", len(eval_dataset))
398     logger.info("  Batch size = %d", args.eval_batch_size)
399     eval_loss = 0.0
400     nb_eval_steps = 0
401     model.eval()
402
403     for batch in tqdm(eval_dataloader, desc="Evaluating"):
404         inputs, labels = mask_tokens(batch, tokenizer, args) if args.mlm else
            (batch, batch)
405         inputs = inputs.to(args.device)
406         labels = labels.to(args.device)
407
408         with torch.no_grad():
409             outputs = model(inputs, masked_lm_labels=labels) if args.mlm else
                model(inputs, labels=labels)
410             lm_loss = outputs[0]
411             eval_loss += lm_loss.mean().item()
412         nb_eval_steps += 1
413
414     eval_loss = eval_loss / nb_eval_steps
415     perplexity = torch.exp(torch.tensor(eval_loss))
```

```
416
417     result = {"perplexity": perplexity}
418
419     output_eval_file = os.path.join(eval_output_dir, prefix,
            "eval_results.txt")
420     with open(output_eval_file, "w") as writer:
421         logger.info("***** Eval results {} *****".format(prefix))
422         for key in sorted(result.keys()):
423             logger.info("  %s = %s", key, str(result[key]))
424             writer.write("%s = %s\n" % (key, str(result[key])))
425
426     return result
427
428
429 def main():
430     parser = argparse.ArgumentParser()
431
432     # Required parameters
433     parser.add_argument(
434         "--train_data_file", default=None, type=str, required=True, help="The
                input training data file (a text file)."
435     )
436     parser.add_argument(
437         "--output_dir",
438         type=str,
439         required=True,
440         help="The output directory where the model predictions and
                checkpoints will be written.",
441     )
442     parser.add_argument(
443         "--model_type", type=str, required=True, help="The model architecture
                to be trained or fine-tuned.",
444     )
445
446     # Other parameters
447     parser.add_argument(
448         "--eval_data_file",
449         default=None,
450         type=str,
451         help="An optional input evaluation data file to evaluate the
                perplexity on (a text file).",
452     )
453     parser.add_argument(
454         "--line_by_line",
455         action="store_true",
```

```
456         help="Whether distinct lines of text in the dataset are to be handled
                as distinct sequences.",
457     )
458     parser.add_argument(
459         "--should_continue", action="store_true", help="Whether to continue
                from latest checkpoint in output_dir"
460     )
461     parser.add_argument(
462         "--model_name_or_path",
463         default=None,
464         type=str,
465         help="The model checkpoint for weights initialization. Leave None if
                you want to train a model from scratch.",
466     )
467
468     parser.add_argument(
469         "--mlm", action="store_true", help="Train with masked-language
                modeling loss instead of language modeling."
470     )
471     parser.add_argument(
472         "--mlm_probability", type=float, default=0.15, help="Ratio of tokens
                to mask for masked language modeling loss"
473     )
474
475     parser.add_argument(
476         "--config_name",
477         default=None,
478         type=str,
479         help="Optional pretrained config name or path if not the same as
                model_name_or_path. If both are None, "
480             "initialize a new config.",
481     )
482     parser.add_argument(
483         "--tokenizer_name",
484         default=None,
485         type=str,
486         help="Optional pretrained tokenizer name or path if not the same as
                model_name_or_path. If both are None, "
487             "initialize a new tokenizer.",
488     )
489     parser.add_argument(
490         "--cache_dir",
491         default=None,
492         type=str,
493         help="Optional directory to store the pre-trained models downloaded
                from s3 (instead of the default one)",
```

```
494     )
495     parser.add_argument(
496         "--block_size",
497         default=-1,
498         type=int,
499         help="Optional input sequence length after tokenization."
500             "The training dataset will be truncated in block of this size
                for training."
501             "Default to the model max input length for single sentence
                inputs (take into account special tokens).",
502     )
503     parser.add_argument("--do_train", action="store_true", help="Whether to
        run training.")
504     parser.add_argument("--do_eval", action="store_true", help="Whether to
        run eval on the dev set.")
505     parser.add_argument(
506         "--evaluate_during_training", action="store_true", help="Run
            evaluation during training at each logging step."
507     )
508
509     parser.add_argument("--per_gpu_train_batch_size", default=4, type=int,
        help="Batch size per GPU/CPU for training.")
510     parser.add_argument(
511         "--per_gpu_eval_batch_size", default=4, type=int, help="Batch size
            per GPU/CPU for evaluation."
512     )
513     parser.add_argument(
514         "--gradient_accumulation_steps",
515         type=int,
516         default=1,
517         help="Number of updates steps to accumulate before performing a
            backward/update pass.",
518     )
519     parser.add_argument("--learning_rate", default=5e-5, type=float,
        help="The initial learning rate for Adam.")
520     parser.add_argument("--weight_decay", default=0.01, type=float,
        help="Weight decay if we apply some.")
521     parser.add_argument("--adam_epsilon", default=1e-8, type=float,
        help="Epsilon for Adam optimizer.")
522     parser.add_argument("--max_grad_norm", default=1.0, type=float, help="Max
        gradient norm.")
523     parser.add_argument(
524         "--num_train_epochs", default=1.0, type=float, help="Total number of
            training epochs to perform."
525     )
526     parser.add_argument(
```

```
527         "--max_steps",
528         default=-1,
529         type=int,
530         help="If > 0: set total number of training steps to perform. Override
                num_train_epochs.",
531     )
532     parser.add_argument("--warmup_steps", default=0, type=int, help="Linear
            warmup over warmup_steps.")
533
534     parser.add_argument("--logging_steps", type=int, default=500, help="Log
            every X updates steps.")
535     parser.add_argument("--save_steps", type=int, default=500, help="Save
            checkpoint every X updates steps.")
536     parser.add_argument(
537         "--save_total_limit",
538         type=int,
539         default=None,
540         help="Limit the total amount of checkpoints, delete the older
                checkpoints in the output_dir, does not delete "
541             "by default",
542     )
543     parser.add_argument(
544         "--eval_all_checkpoints",
545         action="store_true",
546         help="Evaluate all checkpoints starting with the same prefix as
                model_name_or_path ending and ending with "
547             "step number",
548     )
549     parser.add_argument("--no_cuda", action="store_true", help="Avoid using
            CUDA when available")
550     parser.add_argument(
551         "--overwrite_output_dir", action="store_true", help="Overwrite the
                content of the output directory"
552     )
553     parser.add_argument(
554         "--overwrite_cache", action="store_true", help="Overwrite the cached
                training and evaluation sets"
555     )
556     parser.add_argument("--seed", type=int, default=42, help="random seed for
            initialization")
557
558     parser.add_argument(
559         "--fp16",
560         action="store_true",
561         help="Whether to use 16-bit (mixed) precision (through NVIDIA apex)
                instead of 32-bit",
```

```
562     )
563     parser.add_argument(
564         "--fp16_opt_level",
565         type=str,
566         default="O1",
567         help="For fp16: Apex AMP optimization level selected in ['O0', 'O1',
                'O2', and 'O3'].
568               "See details at https://nvidia.github.io/apex/amp.html",
569     )
570     parser.add_argument("--local_rank", type=int, default=-1, help="For
            distributed training: local_rank")
571     parser.add_argument("--server_ip", type=str, default="", help="For
            distant debugging.")
572     parser.add_argument("--server_port", type=str, default="", help="For
            distant debugging.")
573     args = parser.parse_args()
574
575     if args.model_type in ["bert", "roberta", "distilbert", "camembert"] and
            not args.mlm:
576         raise ValueError(
577             "BERT and RoBERTa-like models do not have LM heads but masked LM
                    heads. They must be run using the --mlm "
578             "flag (masked language modeling)."
579         )
580     if args.eval_data_file is None and args.do_eval:
581         raise ValueError(
582             "Cannot do evaluation without an evaluation data file. Either
                    supply a file to --eval_data_file "
583             "or remove the --do_eval argument."
584         )
585     if args.should_continue:
586         sorted_checkpoints = _sorted_checkpoints(args)
587         if len(sorted_checkpoints) == 0:
588             raise ValueError("Used --should_continue but no checkpoint was
                    found in --output_dir.")
589         else:
590             args.model_name_or_path = sorted_checkpoints[-1]
591
592     if (
593             os.path.exists(args.output_dir)
594             and os.listdir(args.output_dir)
595             and args.do_train
596             and not args.overwrite_output_dir
597             and not args.should_continue
598     ):
599         raise ValueError(
```

```
600             "Output directory ({}) already exists and is not empty. Use
                    --overwrite_output_dir to overcome.".format(
601                 args.output_dir
602             )
603         )
604
605     # Setup distant debugging if needed
606     if args.server_ip and args.server_port:
607         # Distant debugging - see
                https://code.visualstudio.com/docs/python/debugging#_attach-to-a-local-scr
608         import ptvsd
609
610         print("Waiting for debugger attach")
611         ptvsd.enable_attach(address=(args.server_ip, args.server_port),
                redirect_output=True)
612         ptvsd.wait_for_attach()
613
614     # Setup CUDA, GPU & distributed training
615     if args.local_rank == -1 or args.no_cuda:
616         device = torch.device("cuda" if torch.cuda.is_available() and not
                args.no_cuda else "cpu")
617         args.n_gpu = 0 if args.no_cuda else torch.cuda.device_count()
618     else:  # Initializes the distributed backend which will take care of
            sychronizing nodes/GPUs
619         torch.cuda.set_device(args.local_rank)
620         device = torch.device("cuda", args.local_rank)
621         torch.distributed.init_process_group(backend="nccl")
622         args.n_gpu = 1
623     args.device = device
624
625     # Setup logging
626     logging.basicConfig(
627         format="%(asctime)s - %(levelname)s - %(name)s -   %(message)s",
628         datefmt="%m/%d/%Y %H:%M:%S",
629         level=logging.INFO if args.local_rank in [-1, 0] else logging.WARN,
630     )
631     logger.warning(
632         "Process rank: %s, device: %s, n_gpu: %s, distributed training: %s,
                16-bits training: %s",
633         args.local_rank,
634         device,
635         args.n_gpu,
636         bool(args.local_rank != -1),
637         args.fp16,
638     )
639
```

```
640     # Set seed
641     set_seed(args)
642
643     # Load pretrained model and tokenizer
644     if args.local_rank not in [-1, 0]:
645         torch.distributed.barrier()
646         # Barrier to make sure only the first process in distributed training
                download model & vocab
647
648     if args.config_name:
649         config = AutoConfig.from_pretrained(args.config_name,
                cache_dir=args.cache_dir)
650     elif args.model_name_or_path:
651         config = AutoConfig.from_pretrained(args.model_name_or_path,
                cache_dir=args.cache_dir)
652     else:
653         # When we release a pip version exposing CONFIG_MAPPING,
654         # we can do `config = CONFIG_MAPPING[args.model_type]()`.
655         raise ValueError(
656             "You are instantiating a new config instance from scratch. This
                    is not supported, but you can do it from "
657             "another script, save it, "
658             "and load it from here, using --config_name"
659         )
660
661     if args.tokenizer_name:
662         tokenizer = AutoTokenizer.from_pretrained(args.tokenizer_name,
                cache_dir=args.cache_dir)
663     elif args.model_name_or_path:
664         tokenizer = AutoTokenizer.from_pretrained(args.model_name_or_path,
                cache_dir=args.cache_dir)
665     else:
666         raise ValueError(
667             "You are instantiating a new tokenizer from scratch. This is not
                    supported, but you can do it from "
668             "another script, save it, "
669             "and load it from here, using --tokenizer_name"
670         )
671
672     if args.block_size <= 0:
673         args.block_size = tokenizer.max_len
674         # Our input block size will be the max possible for the model
675     else:
676         args.block_size = min(args.block_size, tokenizer.max_len)
677
678     if args.model_name_or_path:
```

```
679             model = AutoModelWithLMHead.from_pretrained(
680                 args.model_name_or_path,
681                 from_tf=bool(".ckpt" in args.model_name_or_path),
682                 config=config,
683                 cache_dir=args.cache_dir,
684             )
685         else:
686             logger.info("Training new model from scratch")
687             model = AutoModelWithLMHead.from_config(config)
688 
689         model.to(args.device)
690 
691         if args.local_rank == 0:
692             torch.distributed.barrier()
693             # End of barrier to make sure only the first process in distributed
                training download model & vocab
694 
695         logger.info("Training/evaluation parameters %s", args)
696 
697         # Training
698         if args.do_train:
699             if args.local_rank not in [-1, 0]:
700                 torch.distributed.barrier()
701                 # Barrier to make sure only the first process in distributed
702                 # training process the dataset, and the others will use the cache
703 
704             train_dataset = load_and_cache_examples(args, tokenizer,
                    evaluate=False)
705 
706             if args.local_rank == 0:
707                 torch.distributed.barrier()
708 
709             global_step, tr_loss = train(args, train_dataset, model, tokenizer)
710             logger.info(" global_step = %s, average loss = %s", global_step,
                    tr_loss)
711 
712         # Saving best-practices: if you use save_pretrained for the model and
                tokenizer,
713         # you can reload them using from_pretrained()
714         if args.do_train and (args.local_rank == -1 or
                torch.distributed.get_rank() == 0):
715             # Create output directory if needed
716             if args.local_rank in [-1, 0]:
717                 os.makedirs(args.output_dir, exist_ok=True)
718 
719             logger.info("Saving model checkpoint to %s", args.output_dir)
```

```
720         # Save a trained model, configuration and tokenizer using
                'save_pretrained()'.
721         # They can then be reloaded using 'from_pretrained()'
722         model_to_save = (
723             model.module if hasattr(model, "module") else model
724         )  # Take care of distributed/parallel training
725         model_to_save.save_pretrained(args.output_dir)
726         tokenizer.save_pretrained(args.output_dir)
727
728         # Good practice: save your training arguments together with the
                trained model
729         torch.save(args, os.path.join(args.output_dir, "training_args.bin"))
730
731         # Load a trained model and vocabulary that you have fine-tuned
732         model = AutoModelWithLMHead.from_pretrained(args.output_dir)
733         tokenizer = AutoTokenizer.from_pretrained(args.output_dir)
734         model.to(args.device)
735
736     # Evaluation
737     results = {}
738     if args.do_eval and args.local_rank in [-1, 0]:
739         checkpoints = [args.output_dir]
740         if args.eval_all_checkpoints:
741             checkpoints = list(
742                 os.path.dirname(c) for c in sorted(glob.glob(args.output_dir
                        + "/**/" + WEIGHTS_NAME, recursive=True))
743             )
744             logging.getLogger("transformers.modeling_utils").setLevel(logging.WARN)
                    # Reduce logging
745         logger.info("Evaluate the following checkpoints: %s", checkpoints)
746         for checkpoint in checkpoints:
747             global_step = checkpoint.split("-")[-1] if len(checkpoints) > 1
                    else ""
748             prefix = checkpoint.split("/")[-1] if
                    checkpoint.find("checkpoint") != -1 else ""
749
750             model = AutoModelWithLMHead.from_pretrained(checkpoint)
751             model.to(args.device)
752             result = evaluate(args, model, tokenizer, prefix=prefix)
753             result = dict((k + "_{}".format(global_step), v) for k, v in
                    result.items())
754             results.update(result)
755
756     return results
757
758
```

```
759 if __name__ == "__main__":
760     main()
```

ПРИЛОЖЕНИЕ Б

ЕЩЕ КАРТИНКИ

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Рисунок Б.1 — Еще одна картинка, ничем не лучше предыдущей. Но надо же как-то заполнить место.