Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления

Домашнее задание
по дисциплине «Методы машинного обучения»

Выполнила: Кудрявцев С.Д.

Группа: ИУ5-22М

Проверил: Гапанюк Ю.Е.

Москва, 2022 г.

ОГЛАВЛЕНИЕ

1. ЗАДАНИЕ

Домашнее задание по дисциплине направлено на анализ современных методов машинного обучения и их применение для решения практических задач. Домашнее задание включает три основных этапа:

1. выбор задачи;
2. теоретический этап;
3. практический этап.

Этап выбора задачи предполагает анализ ресурса paperswithcode [**Ошибка! Источник ссылки не найден.**]. Данный ресурс включает описание нескольких тысяч современных задач в области машинного обучения. Каждое описание задачи содержит ссылки на наиболее современные и актуальные научные статьи, предназначенные для решения задачи (список статей регулярно обновляется авторами ресурса). Каждое описание статьи содержит ссылку на репозиторий с открытым исходным кодом, реализующим представленные в статье эксперименты. На этапе выбора задачи обучающийся выбирает одну из задач машинного обучения, описание которой содержит ссылки на статьи и репозитории с исходным кодом.

Теоретический этап включает проработку как минимум двух статей, относящихся к выбранной задаче. Результаты проработки обучающийся излагает в теоретической части отчета по домашнему заданию, которая может включать:

- описание общих подходов к решению задачи;
- конкретные топологии нейронных сетей, нейросетевых ансамблей или других моделей машинного обучения, предназначенных для решения задачи;
- математическое описание, алгоритмы функционирования, особенности обучения используемых для решения задачи нейронных сетей, нейросетевых ансамблей или других моделей машинного обучения;
- описание наборов данных, используемых для обучения моделей;
- оценка качества решения задачи, описание метрик качества и их значений;
- предложения обучающегося по улучшению качества решения задачи.

Практический этап включает повторение экспериментов авторов статей на основе представленных авторами репозиториев с исходным кодом и возможное улучшение обучающимися полученных результатов. Результаты проработки обучающийся излагает в практической части отчета по домашнему заданию, которая может включать:

- исходные коды программ, представленные авторами статей, результаты документирования программ обучающимися с использованием диаграмм UML, путем визуализации топологий нейронных сетей и другими способами;
- результаты выполнения программ, вычисление значений для описанных в статьях метрик качества, выводы обучающегося о воспроизводимости экспериментов авторов статей и соответствии практических экспериментов теоретическим материалам статей;
- предложения обучающегося по возможным улучшениям решения задачи, результаты практических экспериментов (исходные коды, документация) по возможному улучшению решения задачи.

Отчет по домашнему заданию должен содержать:

1. Титульный лист.
2. Постановку выбранной задачи машинного обучения, соответствующую этапу выбора задачи.
3. Теоретическую часть отчета.
4. Практическую часть отчета.
5. Выводы обучающегося по результатам выполнения теоретической и практической частей.
6. Список использованных источников.

2. ПОСТАНОВКА ЗАДАЧИ

В результате анализа содержимого ресурса «Papers with code» была выбрана область обработки естественной речи. В данной области было решено изучить решения задачи ответов на вопросы.

Благодаря возможности моделирования двунаправленных контекстов предварительное обучение на основе шумоподавления на основе автоматического кодирования, такое как BERT, обеспечивает более высокую производительность, чем подходы к предварительному обучению, основанные на авторегрессивном моделировании языка. Однако, полагаясь на искажение ввода с помощью масок, BERT игнорирует зависимость между замаскированными позициями и страдает от несоответствия предварительной настройки и точной настройки. В свете этих плюсов и минусов мы предлагаем XLNet, обобщенный авторегрессионный метод предварительного обучения, который (1) позволяет изучать двунаправленные контексты за счет максимизации ожидаемой вероятности для всех перестановок порядка факторизации и (2) преодолевает ограничения BERT благодаря его авторегрессивному методу. формулировка. Кроме того, XLNet объединяет идеи Transformer-XL, современной авторегрессионной модели, в предварительную подготовку. Эмпирически при сопоставимых условиях эксперимента XLNet превосходит BERT по 20 задачам, часто с большим отрывом, включая ответы на вопросы, вывод на естественном языке, анализ настроений и ранжирование документов.

Повышение древовидности - это высокоэффективный и широко используемый метод машинного обучения. В этой статье мы описываем масштабируемую сквозную систему повышения древовидности под названием XGBoost, которая широко используется специалистами по обработке данных для достижения самых современных результатов во многих задачах машинного обучения. Мы предлагаем новый алгоритм с учетом разреженности для разреженных данных и взвешенный квантильный эскиз для приближенного обучения дереву. Что еще более важно, мы предоставляем информацию о шаблонах доступа к кэшу, сжатии данных и сегментировании для создания масштабируемой системы повышения древовидности. Объединяя эти идеи, XGBoost масштабируется за пределы миллиардов примеров, используя гораздо меньше ресурсов, чем существующие системы.

Благодаря возможности моделирования двунаправленных контекстов предварительное обучение на основе автокодирования с шумоподавлением, такое как BERT, обеспечивает лучшую производительность, чем подходы предварительного

обучения, основанные на авторегрессионном языковом моделировании. Однако, полагаясь на искажение входных данных с помощью масок, БЕРТ пренебрегает зависимостью между замаскированными позициями и страдает от несоответствия предварительной подготовки и точной настройки. В свете этих плюсов и минусов мы предлагаем XLNet, обобщенный метод предварительной подготовки с авторегрессией, который (1) позволяет изучать двунаправленные контексты путем максимизации ожидаемой вероятности по всем перестановкам порядка факторизации и (2) преодолевает ограничения BERT благодаря его авторегрессионной формулировке. Кроме того, XLNet интегрирует идеи Transformer-XL, современной модели авторегрессии, в предварительное обучение. Эмпирически, при сопоставимых условиях эксперимента, XLNet превосходит BERT в 20 задачах, часто с большим отрывом, включая ответы на вопросы, вывод на естественном языке, анализ настроений и ранжирование документов.

3. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

3.1. XLNET

Неконтролируемое обучение представлению оказалось весьма успешным в области обработки естественного языка
[7, 22, 27, 28, 10]. Как правило, эти методы сначала предварительно обучают нейронные сети на крупномасштабных
немаркированных текстовых массивах, а затем настраивают модели или представления для последующих задач. В соответствии
с этой общей идеей высокого уровня в литературе были рассмотрены различные неконтролируемые цели
предварительной подготовки. Среди них авторегрессионное (AR) языковое моделирование и автокодирование (AE) были
двумя наиболее успешными целями предварительной подготовки.
Моделирование языка AR направлено на оценку распределения вероятностей текстового корпуса с помощью авторегрессионной модели [7, 27, 28]. В частности, учитывая текстовую последовательность x = (x1, · · · , xT),
моделирование языка AR разлагает вероятность на прямое произведение p(x) = QT
t=1 p(xt | x<t) или обратное
произведение p(x) = Q1
t=T
p(xt | x>t). Параметрическая модель (например, нейронная сеть) обучается моделированию каждого
условное распределение. Поскольку языковая модель AR обучена кодировать только однонаправленный контекст (прямой или обратный), она неэффективна при моделировании глубоких двунаправленных контекстов.
Напротив, последующие задачи понимания языка часто требуют двунаправленной контекстной информации.
Это приводит к разрыву между моделированием языка AR и эффективным предварительным обучением.
Для сравнения, предварительное обучение на основе АЭ не выполняет явную оценку плотности, а вместо этого нацелено на
восстановление исходных данных из поврежденных входных данных. Примечательным примером является BERT [10], который был
современный подход к предварительной подготовке. Учитывая последовательность входных токенов, определенная часть токенов
заменяется специальным символом [МАСКА], и модель обучается восстановлению исходных токенов из
поврежденной версии. Поскольку оценка плотности не является частью задачи, BERT разрешается использовать двунаправленные контексты для реконструкции. В качестве непосредственного преимущества это устраняет вышеупомянутый
двунаправленный информационный пробел в моделировании языка AR, что приводит к повышению производительности. Однако
искусственные символы, такие как [МАСКА], используемые BERT во время предварительной подготовки, отсутствуют в реальных данных во
время точной настройки, что приводит к несоответствию предварительной подготовки и точной настройки. Более того, поскольку предсказанные токены
замаскированы во входных данных, BERT не может смоделировать совместную вероятность, используя правило произведения, как при
моделировании на языке AR. Другими словами, БЕРТ предполагает, что предсказанные токены независимы друг от друга

другое, учитывая немаскированные лексемы, которые упрощаются как высокоуровневые, дальнодействующие зависимости
, распространенные в естественном языке [9].
Столкнувшись с плюсами и минусами существующих целей предварительной языковой подготовки, в этой работе мы предлагаем
XLNet, обобщенный метод авторегрессии, который использует лучшее из моделирования на языке AR
и AE, избегая при этом их ограничений.
• Во-первых, вместо использования фиксированного прямого или обратного порядка факторизации, как в обычных моделях AR, XLNet максимизирует ожидаемую логарифмическую вероятность последовательности без учета всех возможных перестановок
порядка факторизации. Благодаря операции перестановки контекст для каждой позиции может
состоять из лексем как слева, так и справа. В ожидании каждая позиция учится использовать контекстуальную
информацию из всех позиций, т.е. захватывать двунаправленный контекст.
• Во-вторых, как обобщенная модель языка AR, XLNet не полагается на повреждение данных. Следовательно,
XLNet не страдает от несоответствия между предварительным обучением и настройкой, которому подвержен BERT. Между тем,
цель авторегрессии также обеспечивает естественный способ использования правила произведения для факторизации
совместная вероятность предсказанных токенов, устраняющая предположение о независимости, сделанное в BERT.
В дополнение к новой цели предварительного обучения, XLNet улучшает архитектурные проекты для предварительного обучения.
• Вдохновленный последними достижениями в области моделирования языка AR, XLNet интегрирует
механизм повторения сегментов и схему относительного кодирования Transformer-XL [9] в предварительное обучение, что
эмпирически повышает производительность, особенно для задач, связанных с более длинной текстовой последовательностью.
• Наивное применение архитектуры Transformer(-XL) к языковому моделированию на основе перестановок делает
не работает, потому что порядок факторизации произволен, а цель неоднозначна. В качестве решения мы
предлагаем изменить параметры сети Transformer(-XL), чтобы устранить двусмысленность.
Эмпирически, в сопоставимых условиях эксперимента, XLNet последовательно превосходит BERT [10] по
широкому спектру задач, включая задачи на понимание языка, задачи на понимание прочитанного
, такие как SQuAD и RACE, задачи классификации текста, такие как Yelp и IMDB, и задачу ранжирования документов ClueWeb09-B.
Связанная с этим работа Идея AR-моделирования на основе перестановок была исследована в [32, 12], но там
есть несколько ключевых отличий. Во-первых, предыдущие модели направлены на улучшение оценки плотности за счет
включения в модель “неупорядоченного” индуктивного смещения, в то время как XLNet мотивирован тем, что позволяет языковым
моделям AR изучать двунаправленные контексты. Технически, чтобы построить действительное

распределение прогноза с учетом цели, XLNet включает целевую позицию в скрытое состояние с помощью двухпотокового внимания, в
то время как предыдущие AR-модели, основанные на перестановках, полагались на неявную осведомленность о местоположении, присущую их
Архитектуры MLP. Наконец, как для orderless NADE, так и для XLNet, мы хотели бы подчеркнуть, что
“без порядка” не означает, что входная последовательность может быть произвольно переставлена, но что модель
допускает различные порядки факторизации распределения.
Другая связанная с этим идея заключается в выполнении авторегрессионного шумоподавления в контексте генерации текста [11],
который учитывает только фиксированный порядок
XL Net - это обобщенный метод предварительной подготовки AR, который использует цель моделирования языка перестановок
, чтобы объединить преимущества методов AR и AE. Нейронная архитектура XLNet разработана для
бесперебойной работы с AR objective, включая интеграцию Transformer-XL и тщательный дизайн
механизма двухпотокового внимания. XLNet достигает существенного улучшения по сравнению с предыдущими
целями предварительной подготовки по различным задачам.

3.2. XGBOOST

Машинное обучение и подходы, основанные на данных, становятся очень важными во многих областях. Интеллектуальные классификаторы спама
защищают нашу электронную почту, изучая огромное количество
данных о спаме и отзывов пользователей; рекламные системы учатся подбирать
правильные объявления в нужном контексте; системы обнаружения мошенничества
защищают банки от злоумышленников; системы обнаружения аномальных событий помогают физикам-экспериментаторам находить события
, которые приводят к новой физике. Есть два важных фактора
, которые стимулируют эти успешные приложения: использование эффективных
(статистические) модели, которые фиксируют сложные зависимости данных, и масштабируемые обучающие системы, которые изучают
интересующую модель из больших наборов данных.
Среди методов машинного обучения, используемых на практике,
повышение градиентного дерева [10]

1 - это один из методов, который используется во многих приложениях.
Было показано, что повышение
древовидности дает самые современные результаты по многим стандартным классификационным
критериям [16]. LambdaMart [5], вариант повышения дерева для ранжирования, обеспечивает самый современный результат для ранжирования проблемы. Помимо того, что он используется в качестве автономного предиктора, он
также включен в производственные конвейеры реального мира
для прогнозирования частоты кликов по рекламе [15]. Наконец, это метод выбора ансамбля по умолчанию и используется в
таких конкурсах, как премия Netflix [3].
В этой статье мы описываем XGBoost, масштабируемую
систему машинного обучения для повышения древовидности. Система доступна в
виде пакета с открытым исходным кодом2

. Влияние системы было
широко признано в ряде областей машинного обучения и обработки данных
проблемы добычи полезных ископаемых. Возьмем, к примеру, задания, размещенные на сайте соревнований по машинному обучению Kaggle. Среди
29 решений, выигравших challenge 3, опубликованных в
блоге Kaggle в течение 2015 года, 17 решений использовали XGBoost. Среди этих
решений восемь использовали исключительно XGBoost для обучения модели,
в то время как большинство других комбинировали XGBoost с нейронными сетями в ансамблях. Для сравнения, второй по популярности метод,
глубокие нейронные сети, был использован в 11 решениях. Успех
системы был также засвидетельствован на KDDCup 2015, где
XGBoost использовался каждой командой-победителем в топ-10.
Более того, команды-победители сообщили, что методы ансамбля превосходят хорошо настроенный XGBoost лишь на небольшую
величину [1].
Эти результаты демонстрируют, что наша система дает самые современные результаты по широкому кругу проблем. Примеры
проблем в этих выигрышных решениях включают:
прогнозирование продаж в магазинах; классификация событий физики высоких энергий; классификация веб-текста; прогнозирование поведения клиентов; обнаружение движения; прогнозирование количества кликов по рекламе; классификация вредоносных программ;
категоризация продуктов; прогнозирование рисков опасности; прогнозирование массового отсева из онлайн-курсов. Хотя анализ данных, зависящий от предметной области, и разработка функций играют важную
роль в этих решениях, тот факт, что XGBoost является консенсусным выбором учащихся, показывает влияние и важность
нашей системы и древовидного бустинга.
Наиболее важным фактором успеха XGBoost
является его масштабируемость во всех сценариях. Система работает более чем
в десять раз быстрее, чем существующие популярные решения на одной
машине, и масштабируется до миллиардов примеров в распределенных или
настройки с ограниченным объемом памяти. Масштабируемость XGBoost обусловлена несколькими важными системами и алгоритмической оптимизацией.
Эти инновации включают в себя: новый алгоритм обучения по дереву
предназначен для обработки разреженных данных; теоретически обоснованная процедура взвешенного
квантильного эскиза позволяет обрабатывать веса экземпляров
при приближенном обучении по дереву. Параллельные и распределенные вычисления ускоряют обучение, что позволяет быстрее исследовать модели. Что еще более важно, XGBoost использует неосновные вычислений и позволяет специалистам по обработке данных обрабатывать сотни
миллионов примеров на рабочем столе. Наконец, еще более
интересно объединить эти методы, чтобы создать сквозную
систему, которая масштабируется до еще больших объемов данных с наименьшим количеством
ресурсов кластера. Основные материалы этого документа
перечислены следующим образом:
• Мы разрабатываем и создаем масштабируемую сквозную
систему повышения древовидности.
• Мы предлагаем теоретически обоснованный
эскиз взвешенного квантиля для эффективного расчета предложения.

• Мы представляем новый алгоритм с учетом разреженности для параллельного обучения по дереву.
• Мы предлагаем эффективную блочную структуру с поддержкой кэша
для обучения вне ядра дерева.
Хотя существуют некоторые существующие работы по параллельному наращиванию дерева [22, 23, 19], такие направления, как вычисления вне ядра, обучение с учетом кэша и разреженности, не
были изучены. Что еще более важно, комплексная система
, которая сочетает в себе все эти аспекты, дает новое решение для
реальных случаев использования. Это позволяет специалистам по обработке данных, а также
исследователям создавать мощные варианты алгоритмов повышения древовидности [7, 8]. Помимо этих важных вкладов, мы также
внести дополнительные улучшения в предложение упорядоченного
цель обучения, которую мы включим для полноты картины.
Остальная часть статьи организована следующим образом.
Сначала мы рассмотрим повышение дерева и введем регуляризованную
цель в разделе 2. Затем мы опишем методы поиска разделения в разделе 3, а также проектирование системы в разделе 4, включая
экспериментальные результаты, когда это уместно, чтобы обеспечить количественную поддержку для каждой оптимизации, которую мы описываем. Соответствующая работа обсуждается в разделе 5. Подробные сквозные оценки
включены в раздел 6.

4. ПРАКТИЧЕСКАЯ ЧАСТЬ

4.1. XLNET

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

import collections
import os
import re
import numpy as np
import six
from os.path import join
from six.moves import zip

from absl import flags

import tensorflow as tf


def configure_tpu(FLAGS):
  if FLAGS.use_tpu:
    tpu_cluster = tf.contrib.cluster_resolver.TPUClusterResolver(
        FLAGS.tpu, zone=FLAGS.tpu_zone, project=FLAGS.gcp_project)
    master = tpu_cluster.get_master()
  else:
    tpu_cluster = None
    master = FLAGS.master

  session_config = tf.ConfigProto(allow_soft_placement=True)
  # Uncomment the following line if you hope to monitor GPU RAM growth
  # session_config.gpu_options.allow_growth = True

  if FLAGS.use_tpu:
    strategy = None
    tf.logging.info('Use TPU without distribute strategy.')
  elif FLAGS.num_core_per_host == 1:
    strategy = None
    tf.logging.info('Single device mode.')
  else:
    strategy = tf.contrib.distribute.MirroredStrategy(
        num_gpus=FLAGS.num_core_per_host)
    tf.logging.info('Use MirroredStrategy with %d devices.',
                    strategy.num_replicas_in_sync)

  per_host_input = tf.contrib.tpu.InputPipelineConfig.PER_HOST_V2
  run_config = tf.contrib.tpu.RunConfig(
      master=master,
      model_dir=FLAGS.model_dir,
      session_config=session_config,
      tpu_config=tf.contrib.tpu.TPUConfig(
          iterations_per_loop=FLAGS.iterations,
          num_shards=FLAGS.num_hosts * FLAGS.num_core_per_host,
          per_host_input_for_training=per_host_input),
      keep_checkpoint_max=FLAGS.max_save,
      save_checkpoints_secs=None,
      save_checkpoints_steps=FLAGS.save_steps,
      train_distribute=strategy
  )
  return run_config
```

```
def init_from_checkpoint(FLAGS, global_vars=False):
  tvars = tf.global_variables() if global_vars else
tf.trainable_variables()
  initialized_variable_names = {}
  scaffold_fn = None
  if FLAGS.init_checkpoint is not None:
    if FLAGS.init_checkpoint.endswith("latest"):
      ckpt_dir = os.path.dirname(FLAGS.init_checkpoint)
      init_checkpoint = tf.train.latest_checkpoint(ckpt_dir)
    else:
      init_checkpoint = FLAGS.init_checkpoint

    tf.logging.info("Initialize from the ckpt {}".format(init_checkpoint))

    (assignment_map, initialized_variable_names
    ) = get_assignment_map_from_checkpoint(tvars, init_checkpoint)
    if FLAGS.use_tpu:
      def tpu_scaffold():
        tf.train.init_from_checkpoint(init_checkpoint, assignment_map)
        return tf.train.Scaffold()

      scaffold_fn = tpu_scaffold
    else:
      tf.train.init_from_checkpoint(init_checkpoint, assignment_map)

    # Log customized initialization
    tf.logging.info("**** Global Variables ****")
    for var in tvars:
      init_string = ""
      if var.name in initialized_variable_names:
        init_string = ", *INIT_FROM_CKPT*"
      tf.logging.info("  name = %s, shape = %s%s", var.name, var.shape,
                      init_string)
  return scaffold_fn


def get_train_op(FLAGS, total_loss, grads_and_vars=None):
  global_step = tf.train.get_or_create_global_step()

  # increase the learning rate linearly
  if FLAGS.warmup_steps > 0:
    warmup_lr = (tf.cast(global_step, tf.float32)
                 / tf.cast(FLAGS.warmup_steps, tf.float32)
                 * FLAGS.learning_rate)
  else:
    warmup_lr = 0.0

  # decay the learning rate
  if FLAGS.decay_method == "poly":
    decay_lr = tf.train.polynomial_decay(
        FLAGS.learning_rate,
        global_step=global_step - FLAGS.warmup_steps,
        decay_steps=FLAGS.train_steps - FLAGS.warmup_steps,
        end_learning_rate=FLAGS.learning_rate * FLAGS.min_lr_ratio)
  elif FLAGS.decay_method == "cos":
    decay_lr = tf.train.cosine_decay(
        FLAGS.learning_rate,
        global_step=global_step - FLAGS.warmup_steps,
        decay_steps=FLAGS.train_steps - FLAGS.warmup_steps,
        alpha=FLAGS.min_lr_ratio)
  else:
    raise ValueError(FLAGS.decay_method)
```

```
  learning_rate = tf.where(global_step < FLAGS.warmup_steps,
                           warmup_lr, decay_lr)

  if (FLAGS.weight_decay > 0 and not FLAGS.use_tpu and
      FLAGS.num_core_per_host > 1):
    raise ValueError("Do not support `weight_decay > 0` with multi-gpu "
                     "training so far.")

  if FLAGS.weight_decay == 0:
    optimizer = tf.train.AdamOptimizer(
        learning_rate=learning_rate,
        epsilon=FLAGS.adam_epsilon)
  else:
    optimizer = AdamWeightDecayOptimizer(
        learning_rate=learning_rate,
        epsilon=FLAGS.adam_epsilon,
        exclude_from_weight_decay=["LayerNorm", "layer_norm", "bias"],
        weight_decay_rate=FLAGS.weight_decay)

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

  if grads_and_vars is None:
    grads_and_vars = optimizer.compute_gradients(total_loss)
  gradients, variables = zip(*grads_and_vars)
  clipped, gnorm = tf.clip_by_global_norm(gradients, FLAGS.clip)

  if getattr(FLAGS, "lr_layer_decay_rate", 1.0) != 1.0:
    n_layer = 0
    for i in range(len(clipped)):
      m = re.search(r"model/transformer/layer_(\d+?)/", variables[i].name)
      if not m: continue
      n_layer = max(n_layer, int(m.group(1)) + 1)

    for i in range(len(clipped)):
      for l in range(n_layer):
        if "model/transformer/layer_{}/".format(l) in variables[i].name:
          abs_rate = FLAGS.lr_layer_decay_rate ** (n_layer - 1 - l)
          clipped[i] *= abs_rate
          tf.logging.info("Apply mult {:.4f} to layer-{} grad of
{}".format(
              abs_rate, l, variables[i].name))
          break

  train_op = optimizer.apply_gradients(
      zip(clipped, variables), global_step=global_step)

  # Manually increment `global_step` for AdamWeightDecayOptimizer
  if FLAGS.weight_decay > 0:
    new_global_step = global_step + 1
    train_op = tf.group(train_op, [global_step.assign(new_global_step)])

  return train_op, learning_rate, gnorm


def clean_ckpt(_):
  input_ckpt = FLAGS.clean_input_ckpt
  output_model_dir = FLAGS.clean_output_model_dir

  tf.reset_default_graph()

  var_list = tf.contrib.framework.list_variables(input_ckpt)
  var_values, var_dtypes = {}, {}
  for (name, shape) in var_list:
```

```
    if not name.startswith("global_step") and "adam" not in name.lower():
      var_values[name] = None
      tf.logging.info("Include {}".format(name))
    else:
      tf.logging.info("Exclude {}".format(name))

  tf.logging.info("Loading from {}".format(input_ckpt))
  reader = tf.contrib.framework.load_checkpoint(input_ckpt)
  for name in var_values:
    tensor = reader.get_tensor(name)
    var_dtypes[name] = tensor.dtype
    var_values[name] = tensor

  with tf.variable_scope(tf.get_variable_scope(), reuse=tf.AUTO_REUSE):
    tf_vars = [
        tf.get_variable(v, shape=var_values[v].shape, dtype=var_dtypes[v])
        for v in var_values
    ]
  placeholders = [tf.placeholder(v.dtype, shape=v.shape) for v in tf_vars]
  assign_ops = [tf.assign(v, p) for (v, p) in zip(tf_vars, placeholders)]
  global_step = tf.Variable(
      0, name="global_step", trainable=False, dtype=tf.int64)
  saver = tf.train.Saver(tf.all_variables())

  if not tf.gfile.Exists(output_model_dir):
    tf.gfile.MakeDirs(output_model_dir)

  # Build a model consisting only of variables, set them to the average
values.
  with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    for p, assign_op, (name, value) in zip(placeholders, assign_ops,
                                           six.iteritems(var_values)):
      sess.run(assign_op, {p: value})

    # Use the built saver to save the averaged checkpoint.
    saver.save(sess, join(output_model_dir, "model.ckpt"),
               global_step=global_step)


def avg_checkpoints(model_dir, output_model_dir, last_k):
  tf.reset_default_graph()

  checkpoint_state = tf.train.get_checkpoint_state(model_dir)
  checkpoints = checkpoint_state.all_model_checkpoint_paths[- last_k:]
  var_list = tf.contrib.framework.list_variables(checkpoints[0])
  var_values, var_dtypes = {}, {}
  for (name, shape) in var_list:
    if not name.startswith("global_step"):
      var_values[name] = np.zeros(shape)
  for checkpoint in checkpoints:
    reader = tf.contrib.framework.load_checkpoint(checkpoint)
    for name in var_values:
      tensor = reader.get_tensor(name)
      var_dtypes[name] = tensor.dtype
      var_values[name] += tensor
    tf.logging.info("Read from checkpoint %s", checkpoint)
  for name in var_values:  # Average.
    var_values[name] /= len(checkpoints)

  with tf.variable_scope(tf.get_variable_scope(), reuse=tf.AUTO_REUSE):
    tf_vars = [
        tf.get_variable(v, shape=var_values[v].shape, dtype=var_dtypes[v])
        for v in var_values
```

```
    ]
  placeholders = [tf.placeholder(v.dtype, shape=v.shape) for v in tf_vars]
  assign_ops = [tf.assign(v, p) for (v, p) in zip(tf_vars, placeholders)]
  global_step = tf.Variable(
      0, name="global_step", trainable=False, dtype=tf.int64)
  saver = tf.train.Saver(tf.all_variables())

  # Build a model consisting only of variables, set them to the average
values.
  with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    for p, assign_op, (name, value) in zip(placeholders, assign_ops,
                                           six.iteritems(var_values)):
      sess.run(assign_op, {p: value})
    # Use the built saver to save the averaged checkpoint.
    saver.save(sess, join(output_model_dir, "model.ckpt"),
        global_step=global_step)


def get_assignment_map_from_checkpoint(tvars, init_checkpoint):
  """Compute the union of the current variables and checkpoint
variables."""
  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])
    # tf.logging.info('original name: %s', name)
    if name not in name_to_variable:
      continue
    # assignment_map[name] = name
    assignment_map[name] = name_to_variable[name]
    initialized_variable_names[name] = 1
    initialized_variable_names[name + ":0"] = 1

  return (assignment_map, initialized_variable_names)


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,
               beta_1=0.9,
               beta_2=0.999,
               epsilon=1e-6,
               exclude_from_weight_decay=None,
               include_in_weight_decay=["r_s_bias", "r_r_bias",
"r_w_bias"],
               name="AdamWeightDecayOptimizer"):
    """Constructs a AdamWeightDecayOptimizer."""
    super(AdamWeightDecayOptimizer, self).__init__(False, 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_weight_decay
    self.include_in_weight_decay = include_in_weight_decay

  def apply_gradients(self, grads_and_vars, global_step=None, name=None):
    """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_1,
grad))
      next_v = (
          tf.multiply(self.beta_2, v) + tf.multiply(1.0 - self.beta_2,
                                                    tf.square(grad)))

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

      # Just adding the square of the weights to the loss function is *not*
      # the correct way of using L2 regularization/weight decay with Adam,
      # since that will interact with the m and v parameters in strange
ways.
      #
      # Instead we want ot decay the weights in a manner that doesn't
interact
      # with the m/v parameters. This is equivalent to adding the square
      # of the weights to the loss with plain (non-momentum) SGD.
      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)
```

```
  def _do_use_weight_decay(self, param_name):
    """Whether to use L2 weight decay for `param_name`."""
    if not self.weight_decay_rate:
      return False
    for r in self.include_in_weight_decay:
      if re.search(r, param_name) is not None:
        return True

    if self.exclude_from_weight_decay:
      for r in self.exclude_from_weight_decay:
        if re.search(r, param_name) is not None:
          tf.logging.info('Adam WD excludes {}'.format(param_name))
          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


if __name__ == "__main__":
  flags.DEFINE_string("clean_input_ckpt", "", "input ckpt for cleaning")
  flags.DEFINE_string("clean_output_model_dir", "", "output dir for cleaned
ckpt")

  FLAGS = flags.FLAGS

  tf.app.run(clean_ckpt)
```

5. ВЫВОДЫ

По состоянию на 19 июня 2019 года XLNet превосходит BERTon в 20 задачах и достигает самых современных результатов в 18 задачах. Ниже приведены некоторые сравнения между XLNet-Large и BERT-Large, которые имеют схожие размеры моделей:
Результаты по пониманию прочитанного

Model	RACE accuracy	SQuAD1.1 EM
BERT-Large	72.0	84.1
XLNet-Base		
XLNet-Large	81.75	88.95

Мы используем результаты разработки команды в таблице, чтобы исключить другие факторы, такие как использование дополнительных данных обучения или других методов увеличения данных. Номера тестов приведены в таблице лидеров команд.
Результаты по классификации текста

Model	IMDB	Yelp-2	Yelp-5	DBpedia
BERT-Large	4.51	1.89	29.32	0.64
XLNet-Large	3.79	1.55	27.80	0.62

Приведенные выше цифры - это частота ошибок.

Результаты по GLUE

Model	MNLI	QNLI	QQP	RTE	SST-2
BERT-Large	86.6	92.3	91.3	70.4	93.2
XLNet-Base	86.8	91.7	91.4	74.0	94.7
XLNet-Large	89.8	93.9	91.8	83.8	95.6

Мы используем результаты разработки с одной задачей в таблице, чтобы исключить другие факторы, такие как многозадачное обучение или использование ансамблей.

В рамках домашнего задания был выполнен обзор теоретических и практических материалов, связанных с алгоритмами XLNET и XGBOOST.

В практической части был выполнен обзор содержимого репозитория авторов статьи на GitHub. В результате тестирования можно подтвердить, что модели действительно были обучены.