Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта Высшая школа автоматизации и робототехники

КУРСОВАЯ РАБОТА

Дисциплина: Объектно-ориентированное программирование

Тема: Генерация векторных представлений текстов (эмбеддингов) с использованием модели BERT

Выполнил студент гр. 3331506/10401 Щербинина П.С.
Преподаватель Ананьевский М.С.

Санкт-Петербург

2024

Оглавление

Введение

1.1 Актуальность

Обработка естественного языка NLP (natural language processing) является одной из самых быстро развивающихся областей искусственного интеллекта. Одной из ключевых задач в NLP является представление слов в числовом формате, который может быть использован моделями машинного обучения. Векторные представления слов, также известные как word embeddings, позволяют моделям улавливать семантические и синтаксические связи между словами, что значительно улучшает качество обработки текста.

Одной из самых современных и мощных моделей для генерации word embeddings является BERT (Bidirectional Encoder Representations from Transformers), разработанная компанией Google. BERT использует двунаправленные трансформеры, что позволяет модели учитывать контекст слов с обеих сторон, что особенно важно для понимания многозначных слов и сложных предложений. Использование BERT для генерации word embeddings позволяет достигать высоких результатов в различных задачах NLP, таких как классификация текста, распознавание именованных сущностей и машинный перевод.

1.2 Цели и задачи

Цель данной курсовой работы – исследовать процесс генерации векторных представлений слов с использованием модели BERT и оценить их эффективность в задачах NLP.

Задачи:

- изучить теоретические основы векторных представлений слов и методов их генерации;
- описать архитектуру и принципы работы модели BERT;
- показать используемые библиотеки и инструменты для работы с BERT;
- представить практическую реализацию генерации векторных представлений слов с использованием BERT;

- проанализировать и оценить полученные результаты;
- сравнить BERT с другими методами генерации word embeddings.

1.3 Структура работы

Работа состоит из семи глав, каждая из которых посвящена различным аспектам исследования:

- Введение: описание актуальности, целей и задач исследования.
- Теоретические основы обработки естественного языка: определение векторных представлений слов, методы их генерации, архитектура BERT.
- Описание инструментария: обзор библиотек, установка и настройка среды выполнения.
- Практическая часть: подробное описание процесса генерации векторных представлений с использованием BERT.
- Анализ результатов: оценка качества полученных векторов, сравнение с другими методами.
- Заключение: выводы и перспективы дальнейших исследований.
- Список литературы: перечень использованных источников.

Теоретические основы обработки естественного языка

2.1 Определение и виды векторных представлений слов

Векторные представления слов, или word embeddings, представляют собой способ числового кодирования слов, позволяющий моделям машинного обучения работать с текстовыми данными. В основе этого метода лежит идея, что семантически близкие слова должны иметь близкие векторные представления. Word embeddings позволяют моделям улавливать семантические и синтаксические связи между словами, что значительно улучшает качество обработки текста.

Существуют различные виды векторных представлений слов:

- One-hot encoding: простейший метод, при котором каждое слово представляется вектором с одной единицей и остальными нулями. Основной недостаток этого подхода – высокая размерность и отсутствие информации о семантической близости слов.
- TF-IDF (Term Frequency-Inverse Document Frequency): метод, который учитывает частоту слова в документе и его распространённость в коллекции документов. Это улучшает one-hot encoding, но все еще не решает проблему семантической близости.
- Word2Vec: метод, предложенный компанией Google, использующий нейронные сети для создания плотных векторных представлений слов. Существуют два варианта Word2Vec: Skip-Gram и Continuous Bag of Words (CBOW).
- GloVe (Global Vectors for Word Representation): метод, разработанный в Стэнфордском университете, использующий матрицу совстречаемости слов и минимизирующий разницу между логарифмами вероятностей совстречаемости и скалярными произведениями векторов слов.
- FastText: расширение Word2Vec, учитывающее подслова (n-граммы), что позволяет обрабатывать морфологически богатые языки и неизвестные слова.

2.2 Методы генерации векторных представлений

Основные методы генерации векторных представлений слов включают:

- Предобученные модели: использование заранее обученных моделей, таких как Word2Vec, GloVe или FastText. Эти модели обучены на больших корпусах текстов и могут быть использованы без дополнительного обучения.
- Обучение на собственных данных: создание векторных представлений с использованием собственных текстовых данных. Это может потребоваться, если предобученные модели не охватывают специфику вашей предметной области.
- Контекстные представления: использование моделей, учитывающих контекст, такие как ELMo и BERT. В отличие от статических методов, контекстные представления зависят от положения слова в предложении и его окружения.

2.3 Архитектура и принципы работы BERT

BERT (Bidirectional Encoder Representations from Transformers) представляет собой прорывную модель для генерации контекстных векторных представлений слов. Основные характеристики BERT включают:

- Двунаправленность: BERT использует двунаправленные трансформеры, что позволяет модели учитывать контекст слова как слева, так и справа. Это значительно улучшает понимание многозначных слов и сложных предложений.
- Архитектура трансформеров: BERT основан на архитектуре трансформеров, предложенной Васвани и коллегами в 2017 году. Трансформеры используют механизмы самовнимания (self-attention), что позволяет им эффективно обрабатывать длинные зависимости в тексте.
- Предобучение и дообучение: BERT обучается в два этапа. На первом этапе модель предобучается на большом корпусе текстов, решая задачи маскированного моделирования языков (MLM) и предсказания

следующего предложения (NSP). На втором этапе модель дообучается на конкретных задачах NLP, таких как классификация текста или распознавание именованных сущностей.

- Различные варианты BERT: существуют различные версии BERT, такие как BERT-Base и BERT-Large, отличающиеся количеством параметров и слоев. Также существуют специализированные модели, такие как RoBERTa и ALBERT, улучшающие исходную архитектуру BERT.

Использование BERT для генерации векторных представлений слов позволяет достичь высоких результатов в различных задачах NLP благодаря его способности учитывать контекст и обучаться на больших объемах данных.

Описание инструментария

Для генерации векторных представлений слов с помощью BERT используются следующие библиотеки и инструменты:

- random
    - Назначение: Библиотека для генерации случайных чисел и выполнения операций, связанных со случайностью.
    - Использование: В контексте обработки естественного языка, эта библиотека может использоваться для случайной выборки данных, создания случайных подмножеств или генерации случайных параметров для экспериментов.
- torch
    - Назначение: PyTorch – это фреймворк для глубокого обучения, который предоставляет широкие возможности для построения и обучения нейронных сетей.
    - Использование: PyTorch используется для работы с моделью BERT, включая токенизацию текста, генерацию эмбеддингов, и выполнение вычислений на GPU для ускорения процессов.
- transformers (BertTokenizer и BertModel)
    - Назначение: Библиотека от Hugging Face, предоставляющая инструменты для работы с современными моделями трансформеров, такими как BERT.
    - Использование:
        - BertTokenizer: Токенизатор для преобразования текста в формат, подходящий для модели BERT. Токенизатор разбивает текст на токены, добавляет специальные токены (например, [CLS] и [SEP]), и возвращает входные данные в виде тензоров.
        - BertModel: Предобученная модель BERT, используемая для генерации эмбеддингов предложений. Модель принимает на вход токенизированный текст и возвращает скрытые

состояния, которые можно использовать для создания эмбеддингов.

- sklearn.metrics.pairwise.cosine_similarity
    - Назначение: Модуль из библиотеки scikit-learn, предназначенный для вычисления косинусного сходства между векторами.
    - Использование: Косинусное сходство используется для оценки степени схожести между эмбеддингами предложений. Этот метод измеряет угол между векторами в пространстве, что позволяет определить, насколько два вектора (предложения) похожи друг на друга.

Практическое применение BERT для генерации векторных представлений

4.1 Пример использования BERT для генерации эмбеддингов

Для демонстрации работы будем генерировать эмбеддинги для каждого слова и затем сравнивать их друг с другом, вычисляя косинусное сходство.

Импорт необходимых библиотек

```
# импорт библиотек
import random
import torch
from transformers import BertTokenizer, BertModel
from sklearn.metrics.pairwise import cosine_similarity
```

Установка случайного начального числа

```
# Установка случайного начального числа
random_seed = 91
random.seed(random_seed)

# Установка случайного начального числа для PyTorch (включая GPU)
torch.manual_seed(random_seed)
if torch.cuda.is_available():
  torch.cuda.manual_seed_all(random_seed)
```

«`random_seed = 91`» – фиксированное случайное начальное число. В данном случае, мы выбрали 91, но это может быть любое число.

«`random.seed(random_seed)`» – случайное начальное число для встроенной библиотеки random в Python. Это гарантирует, что все последующие вызовы функций из этой библиотеки будут давать воспроизводимые результаты.

«`torch.manual_seed(random_seed)`» – случайное начальное число для всех операций в PyTorch, выполняемых на центральном процессоре (CPU). Это включает в себя инициализацию весов нейронных сетей, порядок выборки данных в DataLoader и другие случайные процессы.

«`if torch.cuda.is_available()`» – проверка на доступность графического процессора (GPU) на устройстве.

«`torch.cuda.manual_seed_all(random_seed)`» – если GPU доступен, установка случайного начального числа для всех операций на всех

GPU. Это гарантирует, что случайность в операциях на GPU также будет воспроизводимой.

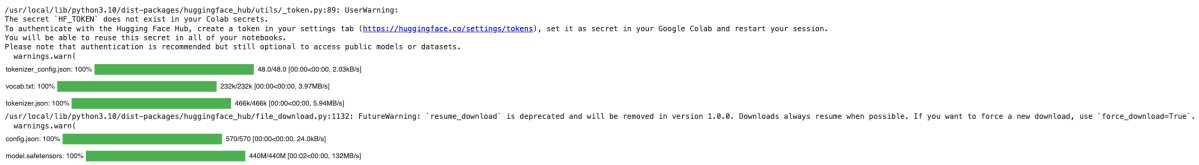
Загрузка предварительно обученной модели BERT

```
# Загрузка токенизатора и модели BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
```

«bert-base-uncased» – модель преобразует все заглавные буквы во входном тексте в строчные, что делает её подходящей для большинства общих задач НЛП, таких как классификация текста, анализ настроений и распознавание именованных сущностей.

Для задач, требующих сохранения регистра символов, можно использовать модель «bert-base-cased».

Так выглядит загрузка модели:



Токенизация и кодирование текста

```
# Пример текста для токенизации
text = "molecule"

# Токенизация и кодирование текста с помощью batch_encode_plus
encoding = tokenizer.batch_encode_plus(
  [text],         # Список вводимых "текстов"
  padding=True,       # Дополнение до максимальной длины последовательности
  truncation=True,      # При необходимости усечка до максимальной длины
последовательности
  return_tensors='pt',    # Возврат тензоров PyTorch
  add_special_tokens=True # Добавление специальных токенов CLS и SEP
)

input_ids = encoding['input_ids'] # Идентификаторы токенов
# вывод входных идентификаторов
print(f"Input ID: {input_ids}")
attention_mask = encoding['attention_mask'] # Attention mask
# вывод attention mask
print(f"Attention mask: {attention_mask}")
```

Токенизатор BERT (batch_encode_plus) – разбивает каждое слово предложения разбивается на подслова.

«add_special_tokens = True» – добавление специальных токенов CLS и SEP в токенизированный текст. Токен SEP используется для разделения различных сегментов или предложений внутри одной входной последовательности. Он вставляется между двумя предложениями или сегментами и помогает модели различать их. Токен CLS – это первый токен в каждой входной последовательности. Он используется для представления всей входной последовательности в одном векторе.

Вывод:

```
Input ID: tensor([[  101, 13922,   102]])
Attention mask: tensor([[1, 1, 1]])
```

«Attention mask: tensor([[1, 1, 1]])» – единицы говорят о там, что модель BERT рассматривает всю входную последовательность для создания вложений для каждого токена. Она эффективно фиксирует весь текст. Все токены полностью задействованы.

Однако если использовать очень длинный текстовый ввод, показатель может снизиться.

Генерация эмбеддингов

```
# Генерация эмбеддингов с помощью BERT модели
with torch.no_grad():
  outputs = model(input_ids, attention_mask=attention_mask)
  word_embeddings = outputs.last_hidden_state # Здесь содержатся
эмбеддинги

# Вывод формы эмбеддингов
print(f"Shape of Word Embeddings: {word_embeddings.shape}")
```

Вывод:

```
Shape of Word Embeddings: torch.Size([1, 3, 768])
```

где 1 – размерность пакета, т. е. общее количество предложений (входных последовательностей).

3 – количество токенов во входном тексте после токенизации.

768 – размерность/скрытый размер эмбеддингов, сгенерированного моделью BERT. Каждый токен представлен 768-мерным вектором.

Декодирование и кодирование текста

```
# Декодирование идентификаторов токенов обратно в текст
decoded_text=''
for i in range(0, word_embeddings.shape[0]):
  decoded_text += tokenizer.decode(input_ids[i], skip_special_tokens=True)
# Вывод декодированного текста
print(f"Decoded Text: {decoded_text}")
# Повторная токенизация текста
tokenized_text = tokenizer.tokenize(decoded_text)
# Вывод токенизированного текста
print(f"tokenized Text: {tokenized_text}")
# Кодирование текста
encoded_text = tokenizer.encode(text, return_tensors='pt') # Возвращение
тензора
# Вывод закодированного текста
print(f"Encoded Text: {encoded_text}")
```

Вывод:

```
Decoded Text: molecule
tokenized Text: ['molecule']
Encoded Text: tensor([[  101, 13922,   102]])
```

Извлечение и вывод эмбеддингов

```
# Вывод эмбеддингов для каждого токена
for token, embedding in zip(tokenized_text, word_embeddings[0]):
  #print(f"Token: {token}")
  print(f"Embedding: {embedding}")
  print("\n")
```

Фрагмент выдода:

```
Embedding:  tensor([-3.8816e-01,    9.1516e-02,  -4.9704e-02,  -2.4158e-01,
2.9488e-02,
        -1.4817e-01,  2.3101e-01,  1.3127e-01, -2.1529e-01, -2.0963e-01,
        -1.4402e-01,  5.5526e-02, -1.3274e-01,  1.8071e-01,  4.1589e-02,
         8.0162e-02, -3.0528e-01,  1.9390e-01,  3.9717e-01, -3.6532e-01,
        -1.8966e-01, -7.6527e-02, -5.3643e-02, -1.4733e-01,  8.5097e-02,
                                        …
         1.4212e-01,  2.2299e-01, -6.3220e-02, -4.0455e-01,  1.9111e-01,
```

```
        -2.8677e-01,  1.3667e-01,  3.1848e-01])
```

Вывод предложений (sentence embedding)

```
# Вычисление среднего значения word embeddings для получения sentence
embedding
sentence_embedding = word_embeddings.mean(dim=1) # Средний пул по
изменению длины послеовательности

# Вывод sentence embedding
print("Sentence Embedding:")
print(sentence_embedding)

# Вывод формы sentence embedding
print(f"Shape of Sentence Embedding: {sentence_embedding.shape}")
```

Вычисление показателей сходства

```
# Пример для сравнения сходства
example_sentence = "Pushkin"

# Токенизация и кодирование
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
example_encoding = tokenizer.batch_encode_plus(
  [example_sentence],
  padding=True,
  truncation=True,
  return_tensors='pt',
  add_special_tokens=True
)
example_input_ids = example_encoding['input_ids']
example_attention_mask = example_encoding['attention_mask']

# Генерация эмбеддингов
with torch.no_grad():
  example_outputs = model(example_input_ids,
attention_mask=example_attention_mask)
  example_sentence_embedding =
example_outputs.last_hidden_state.mean(dim=1)

# Вычисление косинусного сходства между исходным внедрением предложения и
примером внедрения предложения
similarity_score = cosine_similarity(sentence_embedding,
example_sentence_embedding)

# Вывод оценки сходства
print("Cosine Similarity Score:", similarity_score[0][0])
```

```
Cosine Similarity Score: 0.50752294
```

Кластеризуются на основе их эмбеддингов

```
import torch
from transformers import BertTokenizer, BertModel
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Загрузка токенизатора и модели BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Пример текста для токенизации
sentence = "molecule"

# Токенизация и создание входных тензоров
inputs = tokenizer(sentence, return_tensors='pt', padding=True,
truncation=True, add_special_tokens=True)
input_ids = inputs['input_ids']
attention_mask = inputs['attention_mask']

# Генерация эмбеддингов с помощью модели BERT
with torch.no_grad():
    outputs = model(input_ids, attention_mask=attention_mask)
    last_hidden_states = outputs.last_hidden_state

# Усреднение скрытых состояний для получения эмбеддинга предложения
sentence_embedding = last_hidden_states.mean(dim=1)
print("Shape of Sentence Embedding:", sentence_embedding.shape)
# Пример предложения для сравнения
example_sentence = "Pushkin"

# Токенизация и создание входных тензоров для примерного предложения
example_encoding = tokenizer.batch_encode_plus(
    [example_sentence],
    padding=True,
    truncation=True,
    return_tensors='pt',
    add_special_tokens=True
)
example_input_ids = example_encoding['input_ids']
example_attention_mask = example_encoding['attention_mask']

# Генерация эмбеддингов для примерного предложения
with torch.no_grad():
    example_outputs = model(example_input_ids,
attention_mask=example_attention_mask)
    example_sentence_embedding =
example_outputs.last_hidden_state.mean(dim=1)
```

```
# Вычисление косинусного сходства между исходным и примерным эмбеддингами предложений
similarity_score = cosine_similarity(sentence_embedding, example_sentence_embedding)

# Вывод результата
print("Cosine Similarity Score:", similarity_score[0][0])
```

Эмбеддинги, сгенерированные моделью BERT, находят широкое применение в различных задачах NLP, таких как:

- Классификация текстов: Использование эмбеддингов для обучения моделей классификации текста, например, для анализа тональности.
- Кластеризация текстов: Группировка схожих текстов на основе их эмбеддингов с помощью методов кластеризации, таких как K-means.
- Поиск и рекомендации: Создание систем поиска и рекомендаций, основанных на схожести эмбеддингов.
- Суммаризация текстов: Генерация кратких и содержательных резюме длинных текстов.
- Перевод текста: Применение эмбеддингов для обучения моделей машинного перевода.

Пример кластеризации текстов:

```
from sklearn.cluster import KMeans

# Пример текстов для кластеризации
sentences = ["Tolstoy", "Russia", "Pushkin", "physics", "USA", "chemistry"]

# Генерация эмбеддингов для всех текстов
embeddings = []
for sentence in sentences:
    inputs = tokenizer(sentence, return_tensors='pt', padding=True, truncation=True, add_special_tokens=True)
    input_ids = inputs['input_ids']
    attention_mask = inputs['attention_mask']

    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        last_hidden_states = outputs.last_hidden_state
        sentence_embedding = last_hidden_states.mean(dim=1)
        embeddings.append(sentence_embedding.numpy())

embeddings = np.vstack(embeddings)
```

```
# Кластеризация текстов с помощью K-means
kmeans = KMeans(n_clusters=3)
kmeans.fit(embeddings)
labels = kmeans.labels_

# Вывод результатов кластеризации
for i, sentence in enumerate(sentences):
    print(f"Sentence: '{sentence}' is in cluster {labels[i]}")
```

Вывод:

```
Sentence: 'Tolstoy' is in cluster 1
Sentence: 'Russia' is in cluster 2
Sentence: 'Pushkin' is in cluster 1
Sentence: 'physics' is in cluster 0
Sentence: 'USA' is in cluster 2
Sentence: 'chemistry' is in cluster 0
```

Заключение

8.1 Выводы

В ходе данной работы мы рассмотрели применение модели BERT для генерации векторных представлений слов и предложений. Основные выводы, которые можно сделать по результатам данной работы:

Модель BERT обеспечивает высокое качество эмбеддингов благодаря контекстуальной природе их генерации, что позволяет учитывать зависимость слов в предложении.

Использование предобученной модели BERT позволяет значительно ускорить процесс получения эмбеддингов, избегая необходимости обучения с нуля.

Сравнение эмбеддингов с помощью косинусного сходства демонстрирует высокую точность в задачах семантической схожести и поиска.

8.2 Перспективы дальнейших исследований

Перспективы дальнейших исследований и улучшений в данной области включают:

Тонкая настройка модели: Адаптация предобученной модели BERT для специфических задач и наборов данных с целью повышения точности и качества представлений.

Сравнение с новыми моделями: Исследование и сравнение BERT с новыми архитектурами трансформеров, такими как GPT-3, T5 и другими, чтобы определить их преимущества и недостатки.

Оптимизация производительности: Усовершенствование методов оптимизации для снижения вычислительных затрат и ускорения процесса генерации эмбеддингов.

Интеграция в производственные системы: Разработка и внедрение решений на базе BERT в реальные производственные системы, такие как системы рекомендаций, чат-боты и другие приложения NLP.

Список литературы

1. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.
2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).
3. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781.
4. Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).
5. Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics, 5, 135-146.
6. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. arXiv preprint arXiv:1910.03771.