PDL Challenge 4:
Understanding Word Embedidng – BERT

SR+ITL

July 6, 2020

1 Introduction

Last week, we explored natural language processing (NLP) using recurrent neural networks (RNNs) and gated recurrent units (GRUs), emphasizing their strengths in handling sequential data. This week, we delve into transformers and assess their performance on the same dataset. At the core of models like BERT, transformers rely on a self-attention mechanism to analyze the relationships between all words in a sequence simultaneously. This attention mechanism assigns a weight to each word pair, effectively understanding context across long distances without sequential data processing. In BERT, input sequences are tokenized, and a [CLS] token signals the start, while [SEP] indicates boundaries between sentences. BERT uses multi-head attention to capture different aspects of word relationships and positional encodings to retain word order. The model is pre-trained on large text corpora with tasks like masked language modeling and next-sentence prediction to learn nuanced language representations, which can then be fine-tuned for specific tasks like sentiment analysis. This parallel processing and context-aware architecture often yield superior results compared to traditional RNNs and GRUs.

2 Dataset

In our previous session, we introduced the IMDb Movie Reviews dataset for sentiment analysis, focusing on traditional recurrent neural networks (RNNs) and gated recurrent units (GRUs). This week, we continue working with this dataset, shifting to advanced word embeddings provided by the BERT model. The IMDb dataset contains over 50,000 movie reviews categorized as positive or negative, offering a balanced dataset ideal for binary classification tasks.

In this week's challenge, we aim to use BERT's powerful bidirectional context understanding to classify movie reviews into positive and negative sentiments, enhancing accuracy and reducing training time compared to traditional methods. By leveraging word embeddings, BERT effectively maps tokens to high-

dimensional representations, enabling a deeper understanding of sentiment from the given textual data.

3 Understanding BERT and Its Architecture

BERT, or Bidirectional Encoder Representations from Transformers, introduced by Google in 2018, is a groundbreaking model in the field of natural language processing. Its architecture is based on the transformer, a type of model that eschews sequence-aligned recurrent processing in favor of attention mechanisms. This allows BERT to process text bidirectionally, i.e., considering both left-to-right and right-to-left contexts simultaneously, thereby gaining a more comprehensive understanding of language context.

3.1 Transformer Architecture

The transformer architecture, the backbone of BERT, is primarily built on the self-attention mechanism that computes the input and output without relying on sequence-based RNNs or convolutions. This design allows BERT to manage long-range dependencies with ease and makes the model highly efficient for tasks involving large texts.

3.2 Capabilities of BERT

BERT excels in various NLP tasks, including but not limited to:

- Question answering
- Sentence prediction
- Abstract summarization
- Conversational response generation

It achieves this versatility through fine-tuning, where the pre-trained model is slightly adjusted to cater to specific tasks.

3.3 Applying BERT to Sentiment Analysis

Using BERT for sentiment analysis on the IMDb Movie Reviews dataset involves classifying reviews as positive or negative. This binary sentiment classification leverages BERT's deep understanding of contextual relationships within text.

3.4 Loading the Pre-Trained BERT Model

To begin with sentiment analysis, we first load the pre-trained BERT model. The following PyTorch code snippet demonstrates how to load a BERT model pre-trained on the 'bert-base-uncased' dataset, suitable for English text sentiment analysis tasks:

```
from transformers import BertModel, BertTokenizer

# Load pre-trained BERT base model
model = BertModel.from_pretrained('bert-base-uncased')

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

print("Pre-trained␣BERT␣model␣and␣tokenizer␣have␣been␣loaded␣
    ↪ successfully.")
```

This model can now be used to tokenize the reviews, process them into the appropriate format, and then fine-tune the classification layers to suit our specific sentiment analysis task.

4 Visualizing BERT Embeddings

After loading the pre-trained BERT model and processing the IMDb movie reviews, the next step is to visualize the embeddings to understand how BERT represents the semantic relationships between words. This section explores how to visualize these embeddings using dimensionality reduction techniques.

4.1 Methodology for Embedding Visualization

- **Extract Embeddings**: First, extract the embeddings for a subset of the data. These embeddings can be derived from the output of the last hidden layer of the BERT model, which contains the richest representations of the input text.
- **Dimensionality Reduction**: Use dimensionality reduction techniques to reduce the complexity of the embeddings:
  - **Principal Component Analysis (PCA)**: A linear dimensionality reduction technique that reduces embeddings to the two or three most important principal components for visualization on a 2D or 3D plot.
  - **t-Distributed Stochastic Neighbor Embedding (t-SNE)**: A non-linear technique particularly suited for visualizing high-dimensional datasets, preserving local similarities well.
- **Visualization**: Plot the reduced embeddings using a scatter plot. Color-code the points based on their labels (positive or negative reviews) to see if the embeddings form distinct clusters.

4.2 Code Snippet for Visualization

```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import torch

# Assuming 'embeddings' is a matrix of extracted embeddings
# and 'labels' is a list of corresponding labels

# Dimensionality reduction using t-SNE
tsne = TSNE(n_components=2)
reduced_embeddings = tsne.fit_transform(embeddings)

# Plotting the embeddings
plt.figure(figsize=(10, 8))
for label in set(labels):
    indices = [i for i, l in enumerate(labels) if l == label]
    plt.scatter(reduced_embeddings[indices, 0], reduced_embeddings
        ↪ [indices, 1], label=label)

plt.legend()
plt.xlabel("Component␣1")
plt.ylabel("Component␣2")
plt.title("t-SNE␣Visualization␣of␣BERT␣Embeddings")
plt.show()
```

The above code provides a basic example of how to visualize BERT embeddings using t-SNE, with distinct colors for each label. Depending on your analysis, you can also experiment with PCA or other techniques for visualization.

5 Fine-Tuning BERT for IMDb Sentiment Analysis

In this section, we fine-tune the BERT model for sentiment analysis on the IMDb dataset. We follow these steps:

1. Install the required libraries, including the Transformers library.
2. Load the pre-trained BERT model and tokenizer.
3. Prepare the IMDb reviews dataset and preprocess it accordingly.
4. Fine-tune BERT on this dataset and use it to make predictions.

5.1 Code Snippet for Fine-Tuning BERT

Consider the following code snippet.

```
# Install Transformers library if necessary
# !pip install transformers

from transformers import BertTokenizer,
    ↪ TFBertForSequenceClassification
from transformers import InputExample, InputFeatures
import tensorflow as tf

# Load pre-trained BERT model and tokenizer
model = TFBertForSequenceClassification.from_pretrained("bert-
    ↪ base-uncased")
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Convert categories to numeric values
def cat2num(value):
    return 1 if value == 'positive' else 0

df['sentiment'] = df['sentiment'].apply(cat2num)
train = df[:45000]
test = df[45000:]

# Data Preprocessing
def convert_data_to_examples(train, test, review, sentiment):
    train_InputExamples = train.apply(lambda x: InputExample(guid=
        ↪ None, text_a=x[review], label=x[sentiment]), axis=1)
    validation_InputExamples = test.apply(lambda x: InputExample(
        ↪ guid=None, text_a=x[review], label=x[sentiment]), axis
        ↪ =1)
    return train_InputExamples, validation_InputExamples

def convert_examples_to_tf_dataset(examples, tokenizer,
    ↪ max_length=128):
    features = []
    for e in examples:
        input_dict = tokenizer.encode_plus(
            e.text_a, add_special_tokens=True, max_length=
                ↪ max_length, return_token_type_ids=True,
            return_attention_mask=True, pad_to_max_length=True,
                ↪ truncation=True
        )
        features.append(InputFeatures(
            input_ids=input_dict["input_ids"], attention_mask=
                ↪ input_dict["attention_mask"],
```

```
            token_type_ids=input_dict["token_type_ids"], label=e.
                ↪ label))

    def gen():
        for f in features:
            yield {"input_ids": f.input_ids, "attention_mask": f.
                ↪ attention_mask, "token_type_ids": f.
                ↪ token_type_ids}, f.label

    return tf.data.Dataset.from_generator(
        gen, ({"input_ids": tf.int32, "attention_mask": tf.int32,
            ↪ "token_type_ids": tf.int32}, tf.int64),
        ({"input_ids": tf.TensorShape([None]), "attention_mask":
            ↪ tf.TensorShape([None]), "token_type_ids": tf.
            ↪ TensorShape([None])}, tf.TensorShape([])))

# Convert data to examples and TF dataset
train_InputExamples, validation_InputExamples =
    ↪ convert_data_to_examples(train, test, 'review', 'sentiment
    ↪ ')
train_data = convert_examples_to_tf_dataset(list(
    ↪ train_InputExamples), tokenizer).shuffle(100).batch(32).
    ↪ repeat(2)
validation_data = convert_examples_to_tf_dataset(list(
    ↪ validation_InputExamples), tokenizer).batch(32)

# Model Training
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=3e
    ↪ -5),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(
                  ↪ from_logits=True),
              metrics=[tf.keras.metrics.SparseCategoricalAccuracy('
                  ↪ accuracy')])

model.fit(train_data, epochs=2, validation_data=validation_data)

# Example Predictions
pred_sentences = [
    'worst␣movie␣of␣my␣life,␣will␣never␣watch␣movies␣from␣this␣
        ↪ series',
    'Wow,␣blew␣my␣mind,␣what␣a␣movie␣by␣Marvel,␣animation␣and␣
        ↪ story␣is␣amazing'
]
tf_batch = tokenizer(pred_sentences, max_length=128, padding=True
    ↪ , truncation=True, return_tensors='tf')
tf_outputs = model(tf_batch)
```

```
tf_predictions = tf.nn.softmax(tf_outputs[0], axis=-1)

labels = ['Negative', 'Positive']
label = tf.argmax(tf_predictions, axis=1).numpy()
for i in range(len(pred_sentences)):
    print(pred_sentences[i], ": ", labels[label[i]])
```

The above code provides an approach to load, preprocess, and fine-tune BERT on the IMDb dataset for sentiment analysis. It then makes predictions on new movie reviews.

6 Investigating the Performance of the Fine-Tuned Model

After fine-tuning BERT on the IMDb dataset, it is essential to assess the model's effectiveness in classifying sentiments. This section explores the performance of the trained model and visualizes the outcomes for the most positive and most negative reviews.

6.1 Evaluating Model Performance

Evaluate the model's accuracy, precision, recall, and F1-score on the test set to provide a comprehensive view of its strengths and weaknesses in sentiment classification.

6.2 Visualizing Extreme Sentiments

Identify and visualize the reviews predicted as the most positive and most negative by the fine-tuned model. This involves sorting predictions by their sentiment scores and extracting a few samples from each extreme. Present the following:

1. **Most Positive Reviews:** Samples that the model predicts to be overwhelmingly positive.
2. **Most Negative Reviews:** Samples that the model predicts to be strongly negative.

Analyze these examples to provide qualitative insights into the model's behavior and whether it aligns with human sentiment analysis.

7 Exploring Reviewer Sentiments with K-Means Clustering

To gain a more nuanced understanding of the sentiments expressed in movie reviews, this task will involve using K-means clustering on the 64-dimensional

word embeddings extracted from the fine-tuned BERT model. This approach aims to identify distinct clusters of reviews that could reflect various aspects of movie enjoyment beyond simple positive or negative sentiments.

7.1 Clustering Word Embeddings

1. **Extract Embeddings:** Utilize the fine-tuned BERT model to extract 64-dimensional embeddings for each review in the dataset. These embeddings should capture the essence of the reviewers' sentiments.

2. **Apply K-Means Clustering:** Implement K-means clustering on these embeddings to group similar reviews together. Start with a range of cluster numbers to explore how reviews aggregate and determine the optimal number of clusters using metrics such as the elbow method or silhouette score.

3. **Visualize Clusters:** Use dimensionality reduction techniques such as PCA or t-SNE to visualize the clusters in two dimensions. This visualization will help identify how closely related different reviews are and how they group together.

7.2 Interpreting Clusters

1. **Analyze Cluster Content:** Examine the content of each cluster to understand the predominant sentiments or themes. For instance, identify if clusters correspond to different types of enjoyment, such as visual (cinematography, special effects), emotional (storyline, character depth), or other aspects like soundtrack quality.

2. **Label Clusters:** Attempt to label each cluster based on the discovered themes or sentiments. This involves a qualitative analysis of a sample of reviews from each cluster to infer the common thread that binds them.

3. **Report Findings:** Document the findings from the clustering analysis, detailing the different types of movie review sentiments discovered beyond positive and negative. Discuss how these insights could be used to enhance sentiment analysis models or improve content recommendation systems.

This task will not only enhance understanding of the complex nature of sentiment expressed in text but also showcase how clustering can be used as a tool for in-depth data exploration and analysis.