Subject: Large Language Model
Name: Dipen Prajapati
Net ID: dp1435

Project Topic:

Create a Contextualized Word Embeddings using existing pretrained Word Embeddings

Introduction

This project report presents the context word embeddings implementation and comparison against the ELECTRA model. Also, the performance comparison between BERT and ELECTRA models is carried out to evaluate their speed and efficiency in generating embeddings.

Part 1: Contextual Word Embedding with Same Word, Different Meaning

Objective

Visualizing context word embeddings in defining the variation of the same word employed in alternate contexts based on a comparison of embeddings developed by the ELECTRA model.

Methodology

1. **Loading the Model and Tokenizer:**
   - The ELECTRA model and tokenizer are loaded from the transformer library.
2. **Input Sentences:**
   - Give two input sentences for which wants to check context of same word relation with other word embedding.
3. **Generating Embeddings:**
   - Tokenize and translate the sentences to tensors.
   - Model generates embeddings without computing gradients (torch.no_grad()).

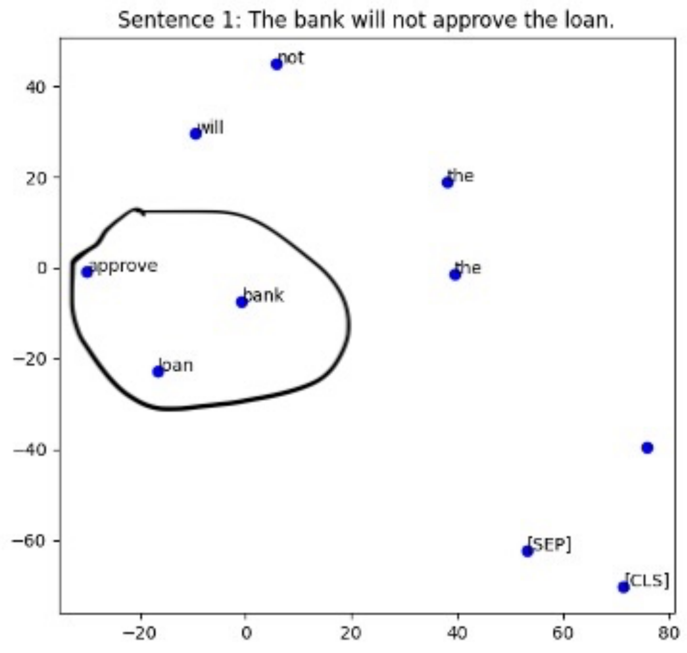
4. Token embeddings extraction and Conversion to Numpy:
   - Extracts the embeddings and converts them to NumPy arrays for visualization.
5. **Dimensionality Reduction (t-SNE):**
   - High-dimensional embeddings are mapped into 2D space for visualization.
6. **Visualization:**
   - Token embeddings of both sentences are plotted to examine how well the model can distinguish between different senses of a particular word.

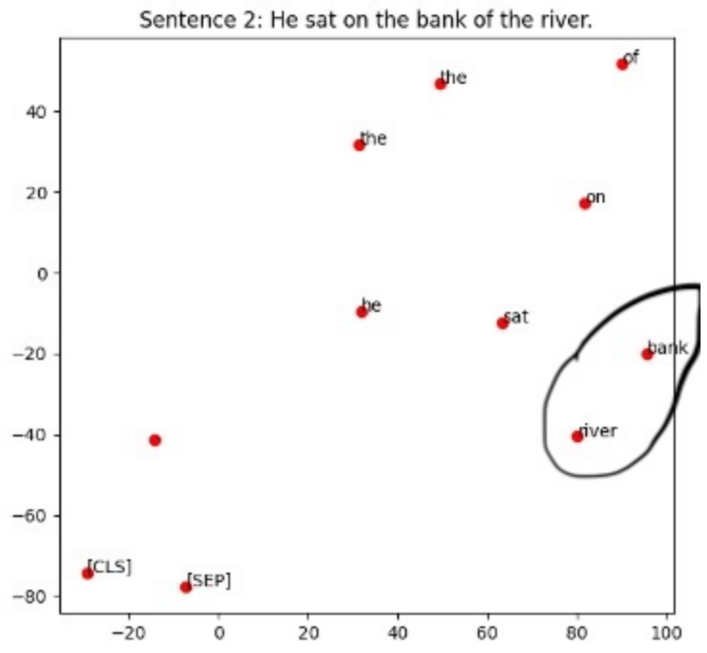
Results & Analysis

- **Input Sentences:**
  1. "The bank will not approve the loan."
  2. "He sat on the bank of the river."

Input sentences can be anything. To check model performance, use same word with different context or meaning in different sentences.

- The embeddings generated by ELECTRA clearly separate the word **'bank'** in two different contexts: financial institution (sentence 1) and riverbank (sentence 2).
- The visualization plot shows distinct clusters for the word 'bank' based on its meaning context.





Part 2: Paragraph Testing - BERT vs. ELECTRA

Objective

To compare the speed and effectiveness of BERT and ELECTRA models in generating contextual embeddings for a paragraph.

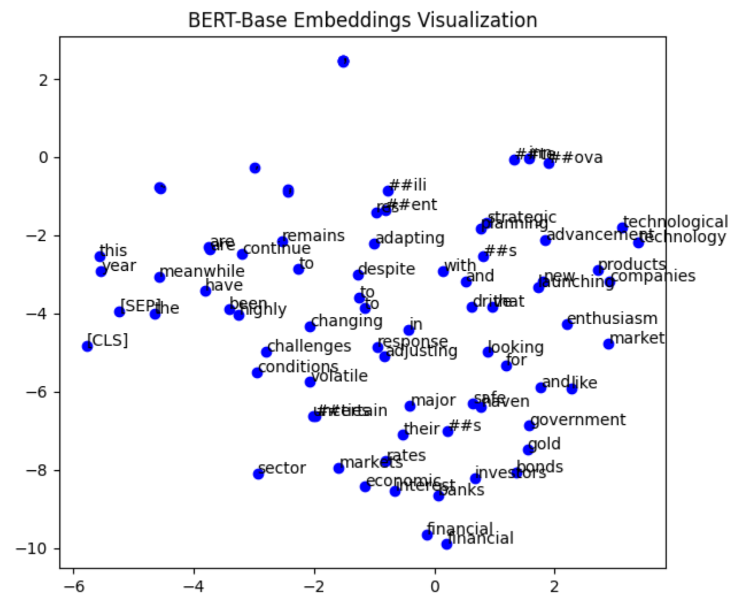
Methodology

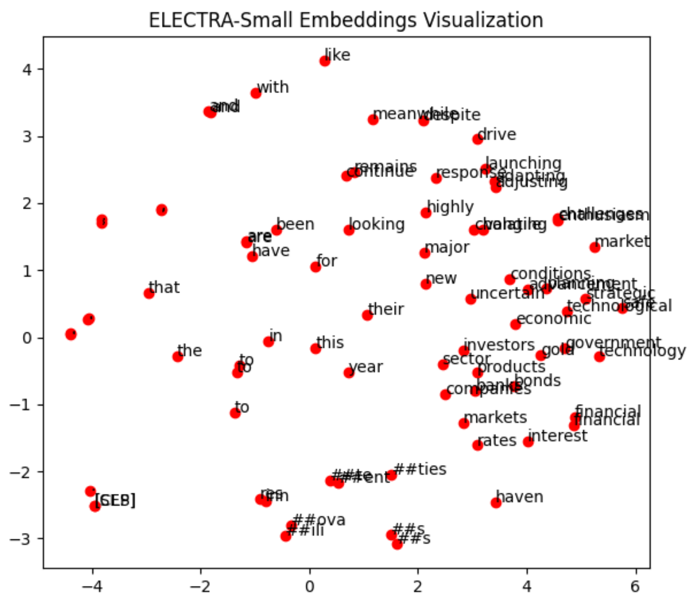
1. **Model Loading:**
   - BERT-base and ELECTRA-small models are loaded using the transformers library.
2. **Device Configuration:**
   - The models are run on GPU.
3. **Embedding Generation:**
   - Embeddings are generated for the same paragraph using both models and measuring time taken by each model is recorded.
4. **Dimensionality Reduction (t-SNE):**
   - Embeddings are reduced to 2D space for visualization and its comparison.
5. **Visualization:**
   - Comparison plots are generated to visualize the embeddings generated by BERT and ELECTRA.

Results & Analysis

- **Input Paragraph:** take any paragraph for embeddings.
- **Output Timing:**
  BERT-Base Time: 1.4768 seconds
  ELECTRA-Small Time: 0.2584 seconds
- **Observations:**
  - ELECTRA is faster than BERT in generating embeddings due to its more efficient architecture.

- Both models produce well embeddings, but ELECTRA produces equally good embeddings at a much faster rate.





Conclusion

When comparing the BERT and ELECTRA models for contextual word embeddings, it was shown that ELECTRA can effectively generate meaningful word embeddings in a significantly shorter amount of processing time. It was shown that the models could easily distinguish between multiple contextual meanings of a single word using t-SNE in embedding visualization.

For real-time applications where processing time is crucial, ELECTRA is faster in speed makes it an appealing option. However, further research is required to evaluate ELECTRA's performance in comparison to BERT on multilingual tasks. Future studies will compare these models' performance in this area by applying them to sentiment analysis of financial news.

The code and results are attached here below in this pdf.

LLMproj_1

March 15, 2025

[1]:
```
pip install transformers datasets torch
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-
packages (4.48.3)
Requirement already satisfied: datasets in /usr/local/lib/python3.11/dist-
packages (3.4.0)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages
(2.6.0+cu124)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-
packages (from transformers) (3.17.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.28.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-
packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-
packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-
packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.21.0)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-
packages (from transformers) (4.67.1)
Requirement already satisfied: pyarrow>=15.0.0 in
/usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in
/usr/local/lib/python3.11/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(from datasets) (2.2.2)
Requirement already satisfied: xxhash in /usr/local/lib/python3.11/dist-packages
(from datasets) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in
/usr/local/lib/python3.11/dist-packages (from datasets) (0.70.16)
```

```
Requirement already satisfied: fsspec<=2024.12.0,>=2023.1.0 in
/usr/local/lib/python3.11/dist-packages (from
fsspec[http]<=2024.12.0,>=2023.1.0->datasets) (2024.10.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-
packages (from datasets) (3.11.13)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.11/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-
packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages
(from torch) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in
/usr/local/lib/python3.11/dist-packages (from torch) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in
/usr/local/lib/python3.11/dist-packages (from torch) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in
/usr/local/lib/python3.11/dist-packages (from torch) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in
/usr/local/lib/python3.11/dist-packages (from torch) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in
/usr/local/lib/python3.11/dist-packages (from torch) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in
/usr/local/lib/python3.11/dist-packages (from torch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-
packages (from torch) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-
packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-
```

```
packages (from aiohttp->datasets) (25.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (0.3.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.18.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers)
(2025.1.31)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->torch) (3.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas->datasets) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas->datasets) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas->datasets) (1.17.0)
```

1 Importing Libraries

[2]:
```
import torch
from transformers import ElectraTokenizer, ElectraModel
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

2 1. Loading ELECTRA tokenizer and model

[3]:
```
model_name = "google/electra-small-discriminator"
tokenizer = ElectraTokenizer.from_pretrained(model_name)
model = ElectraModel.from_pretrained(model_name)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
```

```
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
```

3 2. Taking Input for Two Sentences:

[4]:
```
# Enter Input sentences here
sentence1 = input("Enter first sentence: ")
sentence2 = input("Enter second sentence: ")

sentences = [sentence1, sentence2]
```

```
Enter first sentence: The bank will not approve the loan.
Enter second sentence: He sat on the bank of the river.
```

4 3. Tokenization & Embedding Generation:

[5]:
```
input_1 = tokenizer(sentences[0], return_tensors="pt")
input_2 = tokenizer(sentences[1], return_tensors="pt")

with torch.no_grad():
    output_1 = model(**input_1)
    output_2 = model(**input_2)
```

5 4. Extracting Token Embeddings & Converting to NumPy:

[6]:
```
contx_embd_1 = output_1.last_hidden_state.squeeze(0)
contx_embd_2 = output_2.last_hidden_state.squeeze(0)

embd_1 = contx_embd_1.numpy()
embd_2 = contx_embd_2.numpy()
```

6 5. Dimensionality Reduction using t-SNE:

[7]:
```
# converting higher dimension to 2D using t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=5)
embd_2d_1 = tsne.fit_transform(embd_1)
embd_2d_2 = tsne.fit_transform(embd_2)

tokens_1 = tokenizer.convert_ids_to_tokens(input_1["input_ids"].squeeze(0))
tokens_2 = tokenizer.convert_ids_to_tokens(input_2["input_ids"].squeeze(0))
```

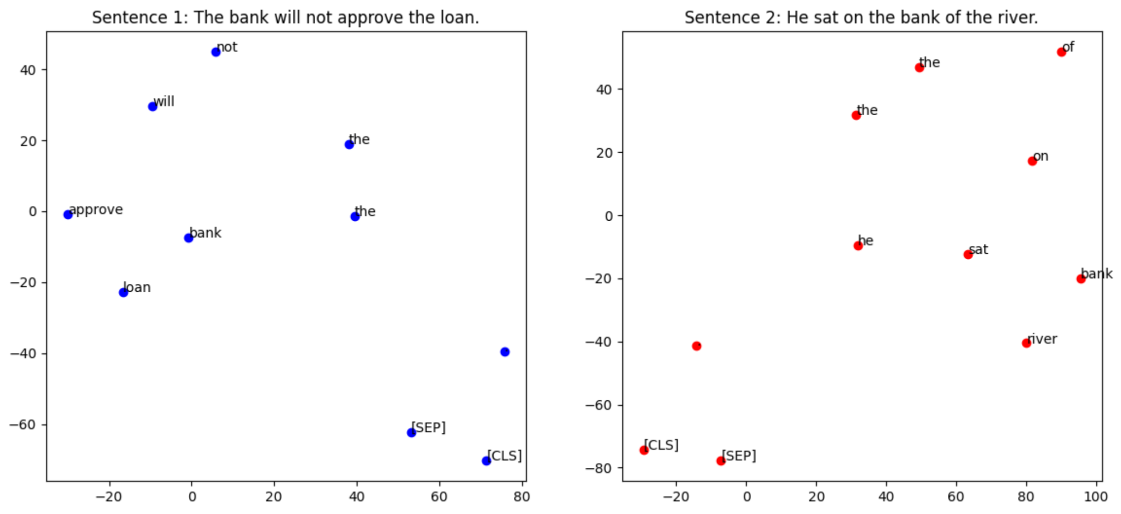
7 6. Plotting & Visualization:

[8]:
```python
# Graph visualization
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Plot for sentence 1
axes[0].scatter(embd_2d_1[:, 0], embd_2d_1[:, 1], c='blue')
for i, token in enumerate(tokens_1):
    axes[0].annotate(token, (embd_2d_1[i, 0], embd_2d_1[i, 1]))
axes[0].set_title(f"Sentence 1: {sentence1}")

# Plot for sentence 2
axes[1].scatter(embd_2d_2[:, 0], embd_2d_2[:, 1], c='red')
for i, token in enumerate(tokens_2):
    axes[1].annotate(token, (embd_2d_2[i, 0], embd_2d_2[i, 1]))
axes[1].set_title(f"Sentence 2: {sentence2}")

plt.show()
```



7.1 Part: 2

8 Importing Libraries

[9]:
```python
import torch
import time
from transformers import BertModel, BertTokenizer, ElectraModel,␣
 ↪ElectraTokenizer
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

9 1. Loading Models and Tokenizers:

[10]:
```
# Paragraph for testing
paragraph = """Financial markets have been highly volatile this year. Major␣
↪banks are adjusting their interest rates
in response to economic uncertainties. Investors are looking for safe havens␣
↪like gold and government bonds.
Meanwhile, technology companies continue to innovate, launching new products␣
↪that drive market enthusiasm.
Despite challenges, the financial sector remains resilient, adapting to␣
↪changing conditions with strategic planning
and technological advancements."""

# BERT-base Model and tokenizer
bert_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
bert_model = BertModel.from_pretrained("bert-base-uncased")

# ELECTRA-small Model and tokenizer
electra_tokenizer = ElectraTokenizer.from_pretrained("google/
↪electra-small-discriminator")
electra_model = ElectraModel.from_pretrained("google/
↪electra-small-discriminator")
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]
```

10 2. Checking Device Availability:

[11]:
```
# Move models to GPU for faster process
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
bert_model.to(device)
electra_model.to(device)
```

[11]:
```
ElectraModel(
  (embeddings): ElectraEmbeddings(
    (word_embeddings): Embedding(30522, 128, padding_idx=0)
    (position_embeddings): Embedding(512, 128)
    (token_type_embeddings): Embedding(2, 128)
    (LayerNorm): LayerNorm((128,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (embeddings_project): Linear(in_features=128, out_features=256, bias=True)
  (encoder): ElectraEncoder(
    (layer): ModuleList(
      (0-11): 12 x ElectraLayer(
        (attention): ElectraAttention(
          (self): ElectraSelfAttention(
            (query): Linear(in_features=256, out_features=256, bias=True)
            (key): Linear(in_features=256, out_features=256, bias=True)
            (value): Linear(in_features=256, out_features=256, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): ElectraSelfOutput(
            (dense): Linear(in_features=256, out_features=256, bias=True)
            (LayerNorm): LayerNorm((256,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): ElectraIntermediate(
          (dense): Linear(in_features=256, out_features=1024, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): ElectraOutput(
          (dense): Linear(in_features=1024, out_features=256, bias=True)
          (LayerNorm): LayerNorm((256,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
)
```

11 3. Generating Embeddings and Measuring Time:

[12]:
```
# function for embeddings and time measuremnet so that i can use for both models
def embedding(model, tokenizer, text):
    ip = tokenizer(text, return_tensors="pt", truncation=True, padding=True).
 ↪to(device)
    start_time = time.time()
    with torch.no_grad():
        output = model(**ip)
    end_time = time.time()
    embedding_time = end_time - start_time
    embeddings = output.last_hidden_state.squeeze(0).cpu().numpy()
    tokens = tokenizer.convert_ids_to_tokens(ip["input_ids"].squeeze(0))
    return embeddings, embedding_time, tokens
```

```
# embeddings and time for BERT-Base
bert_embd, bert_time, bert_tokens = embedding(bert_model, bert_tokenizer,␣
 ↪paragraph)

# embeddings and time for ELECTRA-Small
electra_embd, electra_time, electra_tokens = embedding(electra_model,␣
 ↪electra_tokenizer, paragraph)

# Print Time Taken
print(f"BERT-Base Time: {bert_time:.4f} seconds")
print(f"ELECTRA-Small Time: {electra_time:.4f} seconds")
```

```
BERT-Base Time: 1.4768 seconds
ELECTRA-Small Time: 0.2584 seconds
```

12 4. Dimensionality Reduction & Visualization:

[13]:
```
# Reduce dimensions to 2D using t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=35)
bert_embd_2d = tsne.fit_transform(bert_embd)
electra_embd_2d = tsne.fit_transform(electra_embd)
```

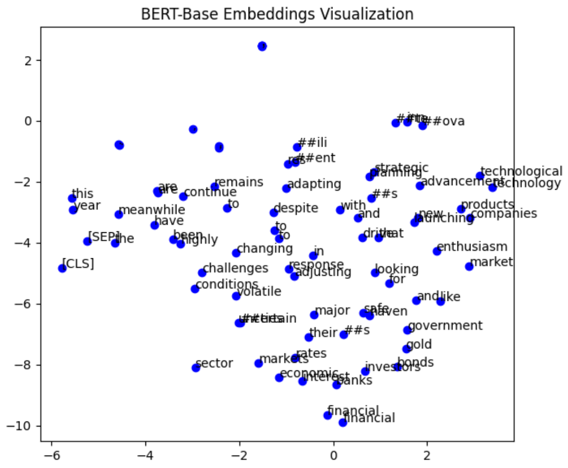
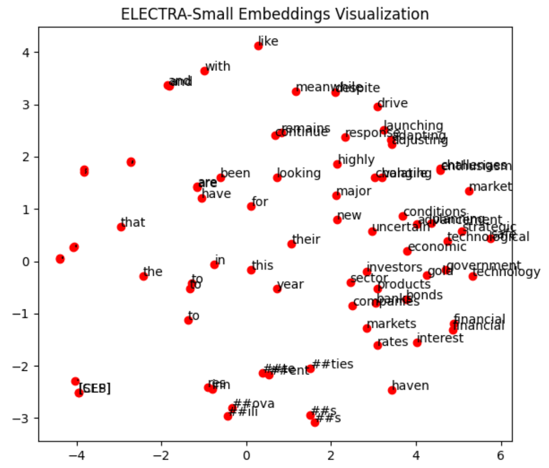
13 5. Plotting Results:

[14]:
```
# Plotation of embeddings
plt.figure(figsize=(15, 6))

# BERT Plot
plt.subplot(1, 2, 1)
plt.scatter(bert_embd_2d[:, 0], bert_embd_2d[:, 1], c='blue')
for i, token in enumerate(bert_tokens):
    plt.annotate(token, (bert_embd_2d[i, 0], bert_embd_2d[i, 1]))
plt.title("BERT-Base Embeddings Visualization")

# ELECTRA Plot
plt.subplot(1, 2, 2)
plt.scatter(electra_embd_2d[:, 0], electra_embd_2d[:, 1], c='red')
for i, token in enumerate(electra_tokens):
    plt.annotate(token, (electra_embd_2d[i, 0], electra_embd_2d[i, 1]))
plt.title("ELECTRA-Small Embeddings Visualization")

plt.show()
```


BERT-Base Embeddings Visualization
[CLS]
[SEP]
this
year
meanwhile
have
been
highly
the
are
continue
remains
to
adapting
despite
with
and
strategic
planning
advancements
technological
technology
products
companies
new
launching
drive
that
enthusiasm
market
changing
in
challenges
conditions
volatile
response
adjusting
looking
for
and
like
major
safe
haven
their
##s
government
gold
uncertain
##ties
rates
markets
sector
economic
interest
banks
investors
bonds
financial
##ili
##ent
##ova

ELECTRA-Small Embeddings Visualization
like
with
and
meanwhile
despite
drive
remains
continue
response
launching
adapting
adjusting
highly
changing
volatile
challenges
enthusiasm
been
are
have
looking
major
for
market
new
conditions
uncertain
advancement
strategic
technological
that
their
economic
in
this
the
to
year
investors
gold
government
technology
sector
products
banks
bonds
companies
markets
rates
interest
financial
##ties
##ent
haven
[SEP]
##ova
##ili
##s