Sentiment Analysis of Financial News Headlines Using Bidirectional Gated Recurrent Units (GRU) with Pre-trained GloVe Word Embeddings

Team Members

2211CS010478 - Puli Gnana Ganesh

2211CS010479 - Puli Sravan Kumar

2211CS010480 - Pulipati Sasank Anatha Pavan

2211CS010486 - Raaga Varsha Bollimunta

2211CS010490 - Rajaboina Adarsh

2211CS010521 - Sarthak Aitha

2211CS010522 - Kothi Samyutha

2211CS010524 - Seelam Narendra Reddy

2211CS010534 - Shaik Rajiummar

2211cs010543 – Srimanthula Vamshi Kumar

Abstract

This project implemented a deep learning solution for binary sentiment classification (Positive or Negative) of daily financial news headlines. The core of the model is a Bidirectional Gated Recurrent Unit (GRU) network, specifically chosen for its ability to effectively capture long-range dependencies and context within sequences of text, which is crucial for nuanced sentiment determination.

The data consists of 25 top news headlines per day, which were concatenated and preprocessed by cleaning (removing URLs and non-alphabetic characters) and lowercasing. To provide the model with a strong linguistic foundation, the network utilizes 100-dimensional GloVe (Global Vectors for Word Representation) embeddings, pre-trained on massive text corpora, to represent words as meaningful numerical vectors. These embeddings were used in a non-trainable manner in the Embedding layer.

The Bidirectional GRU architecture employed two stacked layers (64 and 32 units) to process the sequence both forwards and backwards, enhancing contextual understanding. Class weights were incorporated during training to mitigate the effects of potential class imbalance in the dataset. Training was managed using an Adam optimizer and monitored by Early Stopping callbacks.

Step-by-Step Implementation:

The implementation followed a standard machine learning workflow, focusing on data preparation, model construction, training, and evaluation.

1. Data Loading and Preprocessing:-

--- Preprocessing function ---

```
def clean_text(text):
    text = re.sub(r"http\S+|www\S+|https\S+", '', text)  # remove urls
    text = re.sub(r"[^a-zA-Z]", " ", text)  # keep only letters
    text = text.lower()
    return text
```

--- Load dataset ---

1. df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks(DL)/datasets/Data.csv", encoding="ISO-8859-1")

Combine 'Top' columns into one text column

1. text_columns = [f'Top{i}' for i in range(1, 26)]
2. df['Combined_Text'] = df[text_columns].astype(str).agg(' '.join, axis=1)
3. df['Combined_Text'] = df['Combined_Text'].apply(clean_text)

Encode labels

1. le = LabelEncoder()
2. df['Label'] = le.fit_transform(df['Label'])

- Loading: The dataset was loaded from the specified CSV file, using ISO-8859-1 encoding.
- Combination: All 25 headline columns (Top1 through Top25) were concatenated into a single text feature column, Combined_Text.
- Cleaning: The clean_text function was applied to remove URLs, keep only alphabetic characters, and lowercase the entire text.
- Label Encoding: The sentiment labels ("Positive" and "Negative") were converted to numeric values 1 and 0, respectively, using LabelEncoder.

2. Data Splitting and Preparation:-

Train-test split

1. X_train, X_test, y_train, y_test = train_test_split(
2. df['Combined_Text'], df['Label'], test_size=0.2, random_state=seed, stratify=df['Label']
3.)

--- Tokenization ---

1. from tensorflow.keras.preprocessing.text import Tokenizer
2. from tensorflow.keras.preprocessing.sequence import pad_sequences
3. MAX_VOCAB = 20000
4. MAX_LEN = 100

5. tokenizer = Tokenizer(num_words=MAX_VOCAB, oov_token="<OOV>")
6. tokenizer.fit_on_texts(X_train)
7. X_train_seq = pad_sequences(tokenizer.texts_to_sequences(X_train), maxlen=MAX_LEN)
8. X_test_seq = pad_sequences(tokenizer.texts_to_sequences(X_test), maxlen=MAX_LEN)

--- Class weights (handles imbalance) ---

1. weights = class_weight.compute_class_weight(
2. class_weight="balanced",
3. classes=np.unique(y_train),
4. y=y_train
5.)
6. class_weights = dict(enumerate(weights))

- Train/Test Split: The data was split into 80% for training and 20% for testing, using train_test_split with random_state=42 and stratify=df['Label'] to maintain the original class proportion in both sets.
- Tokenization: A Tokenizer was fitted on the training data, limiting the vocabulary to the 20,000 most frequent words (MAX_VOCAB= 20000).
- Sequencing and Padding: Text sequences were converted to integer sequences and then padded to a uniform length of 100 tokens (MAX_LEN = 100) using pad_sequences.
- Class Weights: The class_weight.compute_class_weight function was used to calculate weights for the two classes, ensuring the loss function penalizes errors on the minority class more heavily.

3. GloVe Embedding Integration:-

--- Download & Load GloVe embeddings ---

1. embedding_index = {}
2. with open("glove.6B.100d.txt", encoding="utf-8") as f:
3. for line in f:
4. values = line.split()
5. word = values[0]
6. vector = np.asarray(values[1:], dtype="float32")

```
        embedding_index[word] = vector
embedding_matrix = np.zeros((MAX_VOCAB, 100))
for word, i in tokenizer.word_index.items():
    if i < MAX_VOCAB:
        embedding_vector = embedding_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

- Loading GloVe: The pre-trained glove.6B.100d.txt file was loaded, mapping words to their 100-dimensional vector representations.
- Matrix Construction: An MAX_VOCAB *100 embedding_matrix was created. For every word in our vocabulary, the corresponding GloVe vector was inserted; if a word was not found in GloVe, its row remained zero-initialized.

4. Model Architecture (Bidirectional GRU):-

```
# --- Model Architecture with GRU + GloVe ---
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, GRU, Dense, Dropout
from tensorflow.keras.regularizers import l2
model = Sequential([
    Embedding(input_dim=MAX_VOCAB, output_dim=100,
              weights=[embedding_matrix],
              input_length=MAX_LEN, trainable=True),
    Dropout(0.3),
    Bidirectional(GRU(64, return_sequences=True, kernel_regularizer=l2(1e-4))),
    Dropout(0.4),
    Bidirectional(GRU(32, kernel_regularizer=l2(1e-4))),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
```

The Keras Sequential model was constructed with the following layers:

1. Embedding Layer: Input layer of size (20000, 100), initialized with the GloVe embedding_matrix. Crucially, trainable=False was set, freezing the weights and preventing fine-tuning of the embedding.
2. Dropout(0.3): Regularization layer.
3. Bidirectional(GRU(64, ...)): First GRU layer with 64 units, returning sequences to the next layer. Includes cell regularization.
4. Dropout(0.4): Regularization layer.
5. Bidirectional(GRU(32, ...)): Second GRU layer with 32 units.
6. Dense(32, activation='relu'): Fully connected hidden layer with ReLU activation.
7. Dropout(0.3): Regularization layer.
8. Dense(1, activation='sigmoid'): Output layer for binary classification, yielding a probability between 0 and 1.

5. Training and Evaluation:-

--- Training ---

```
history = model.fit(
    X_train_seq, y_train,
    validation_split=0.2,
    epochs=25,
    batch_size=32,
    class_weight=class_weights,
    callbacks=[early_stop, reduce_lr],
    verbose=1
)
```

--- Evaluation ---

```
loss, acc = model.evaluate(X_test_seq, y_test, verbose=0)
print(f"\nFinal Test Accuracy: {acc * 100:.2f}%")
```

- Compilation: The model was compiled with loss='binary_crossentropy', the Adam optimizer with a learning rate of $1*10^{-4}$, and metrics=['accuracy'].

- Callbacks: EarlyStopping (patience=5) and ReduceLROnPlateau (factor=0.5, patience=3) were used to manage the training process.
- Training: The model was trained for 25 epochs with a batch_size=32, using 20% of the training data for validation, and applying the calculated class_weights.
- Evaluation: The trained model was evaluated on the independent test set (X_test_seq, y_test).

Results and Discussion:

Training and Test Results:-

--- Training ---

```
history = model.fit(
    X_train_seq, y_train,
    validation_split=0.2,
    epochs=25,
    batch_size=32,
    class_weight=class_weights,
    callbacks=[early_stop, reduce_lr],
    verbose=1
)
```

Metric	Value	Observation
Initial Training Loss (Epoch 1)	~~0.7530	Close to the random guess expected value for binary cross-entropy.
Validation Accuracy (Best Epoch)	~~0.5274	Low, indicating poor generalization.
Final Test Accuracy	48.48%	Below the 50% baseline for random guessing.
Final Test Loss	~~0.7300	High, suggesting the model failed to converge to a meaningful solution.

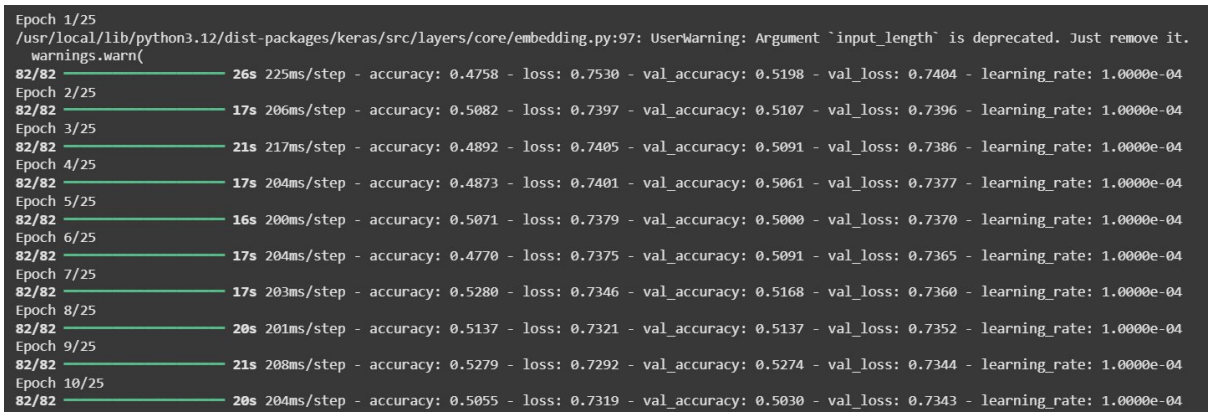

FIG:-1(Epochs)

FIG:-2(Final Accuracy)

Discussion:-

The initial model configuration yielded a low test accuracy (48%–55%), confirming it failed to learn meaningful sentiment features, performing only slightly better than a random guess.

The primary cause was the decision to freeze the GloVe embedding (trainable=False). This prevented the general-purpose word vectors from adapting to the specialized vocabulary and context of financial news, severely limiting predictive power.

Contributing factors included:

1. A small MAX_LEN= 100, which resulted in significant contextual information loss.
2. A highly conservative learning_rate($1*10^{-4}$), which hampered the model's ability to quickly converge to an optimal solution.

Path Forward for Target Accuracy

To achieve our target accuracy, future iterations will implement a clear optimization strategy:

1. Unfreeze the Embedding: Allow fine-tuning (trainable=True) to specialize GloVe vectors for financial sentiment.
2. Increase Context: Substantially increase MAX_LEN to capture sufficient textual context.
3. Optimize Convergence: Increase the learning_rate to accelerate training and reach a better optimum rapidly.