Overview

The notebook trains a binary classification model to predict targets (`0` or `1`) for Quora questions using:

- **Text preprocessing** with NLTK (lemmatization, stopword removal)
- **GloVe embeddings** (300-dimensional vectors from `glove.42B.300d.txt`)
- A **LSTM-based neural network** architecture

Data Loading and Preprocessing

1. Mount Google Drive & Load Data

```
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/Datasets/Quora Text Classification Data.csv')
```

- Loads the dataset from Google Drive, assumed to have columns `question_text` (input text) and `target` (binary label).

2. Text Cleaning Pipeline

```
def cleaning(text):
    text = text.lower()
    words = word_tokenize(text)
    words = [w for w in words if w not in stop_words]
    words = [lem.lemmatize(w) for w in words]
    return ' '.join(words)
df['Clean Text'] = df['question_text'].progress_apply(cleaning)
```

- Converts text to lowercase, removes stopwords/punctuation, and lemmatizes tokens using WordNetLemmatizer[1][2].
- Outputs a cleaned text column for model input.

3. NLTK Setup

```
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

- Downloads required NLTK resources for tokenization and lemmatization[1].

GloVe Embeddings Setup

1. Load Pretrained Embeddings

```
!unzip '/content/drive/MyDrive/Word Embeddings/glove.42B.300d.zip'
embedding_values = {}
f = open('/content/glove.42B.300d.txt')
for line in tqdm(f):
    word = line.split(' ')[^0]
    coef = np.array(line.split(' ')[1:], dtype="float32")
    embedding_values[word] = coef
```

- Unzips and loads 300D GloVe vectors into a dictionary[3][2].

2. Tokenization & Sequence Padding

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['Clean Text'])
seq = tokenizer.texts_to_sequences(x)
pad_seq = pad_sequences(seq, maxlen=300)
```

- Converts text to integer sequences and pads them to a fixed length of 300[4][2].

3. Embedding Matrix Creation

```
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in tokenizer.word_index.items():
    embedding_matrix[i] = embedding_values.get(word, np.zeros(300))
```

- Maps tokenized words to their GloVe vectors. Unknown words are initialized as zero vectors[2].

Model Architecture

```
model = Sequential()
model.add(Embedding(vocab_size, 300, input_length=300,
                    weights=[embedding_matrix], trainable=False))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

1. **Embedding Layer**: Uses pre-trained GloVe vectors (frozen with `trainable=False`)[2].
2. **LSTM Layer**: 50-unit LSTM to capture sequential dependencies[4][3].
3. **Dense Layers**: 128-unit ReLU layer followed by a sigmoid output for binary classification.

Compilation:

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Training

```
history = model.fit(pad_seq, y, validation_split=0.2, epochs=5)
```

- Trains for 5 epochs with an 80-20 train-validation split.
- Uses Adam optimizer and binary cross-entropy loss[2].

Key Parameters and Design Choices

1. **Sequence Length**: `maxlen=300` ensures consistent input size, truncating/padding texts as needed[4].
2. **Embedding Freezing**: GloVe vectors are not updated during training to leverage pretrained semantics[2].
3. **LSTM Units**: 50 units balance complexity and computational cost for this task[3].

Potential Improvements

1. **Bidirectional LSTM**: Replace `LSTM` with `Bidirectional(LSTM)` to capture forward/backward context[5].
2. **Class Imbalance Handling**: Add class weights or oversampling if targets are skewed[2].
3. **Hyperparameter Tuning**: Experiment with LSTM units, sequence length, and optimizer settings.
4. **Advanced Tokenization**: Incorporate bigrams or custom tokenization rules[1].

---

1. https://www.kdnuggets.com/2018/03/simple-text-classifier-google-colaboratory.html
2. https://www.kaggle.com/code/sepidafs/text-classification-with-glove-lstm-gru-99-acc
3. https://www.youtube.com/watch?v=e0WW5w13V64
4. https://www.mathworks.com/help/releases/R2021a/deeplearning/ug/classify-text-data-using-deep-learning.html
5. https://github.com/patrikasvanagas/Bi-LSTM-Glove-Sentiment-Classification