Model B: Imbalanced Review Rating Predictor

Purpose

Model A is a deep learning-based text classification model built to predict review ratings (1 to 5 stars) from raw product review text. It was trained on a **balanced dataset**, where each rating class (1 through 5) has an equal number of samples. This design helps ensure fairness by preventing the model from being biased toward dominant classes.

Full code:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout
from tensorflow.keras.callbacks import EarlyStopping


model = Sequential()

model.add(Embedding(input_dim=vocab_size,
                output_dim=embedding_dim,
                weights=[embedding_matrix],
                input_length=max_len,
                trainable=True))
model.add(Bidirectional(LSTM(128, return_sequences=False)))
model.add(Dropout(0.5))  # optional but helps reduce overfitting
model.add(Dense(32, activation='relu'))
model.add(Dense(5, activation='softmax'))  # 5 classes for ratings

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

# Train
early_stop = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
history = model.fit(X_train_pad, y_train, epochs=20, batch_size=64, validation_data=(X_test_pad,
y_test),callbacks=[early_stop])
```

Model Architecture

The model is constructed using the **Keras Sequential API** and follows a typical embedding + RNN + dense classifier pipeline. Here's a breakdown of each component:

1. Embedding Layer

```
Embedding(input_dim=vocab_size,
          output_dim=300,
          weights=[embedding_matrix],
          input_length=max_len,
          trainable=True)
```

- **Purpose**: Converts integer token sequences into dense vector representations using pretrained FastText embeddings (`wiki.simple.vec`, 300D).
- **Trainable**: Set to `True` to allow fine-tuning during training.
- **Weights**: Initialized using an embedding matrix built from FastText vectors.
- **Input Length**: Fixed using `max_len` (set during preprocessing and padding).

2. Bidirectional LSTM Layer

```
Bidirectional(LSTM(128, return_sequences=False))
```

- **Purpose**: Extracts contextual information from both the past and future (forward and backward) using 128 LSTM units.
- **Bidirectionality**: Helps capture sentiment and meaning more robustly than a unidirectional LSTM.
- **`return_sequences=False`**: Only the final hidden state is used for classification (suitable for sentence-level prediction).

3. Dropout Layer

```
Dropout(0.5)
```

- **Purpose**: Prevents overfitting by randomly disabling 50% of the neurons during each training pass.

4. Dense Layer

```
Dense(32, activation='relu')
```

- **Purpose**: Acts as a non-linear transformation layer that learns intermediate features.
- **ReLU Activation**: Adds non-linearity and helps in learning complex relationships in data.

5. Output Layer

```
Dense(5, activation='softmax')
```

- **Purpose**: Predicts the probability distribution over 5 review classes (0 through 4), which are mapped to ratings 1 through 5 after prediction.
- **Softmax**: Ensures outputs represent class probabilities that sum to 1.

Model Compilation

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

- **Loss Function**: `sparse_categorical_crossentropy` — used for multi-class classification with integer labels (instead of one-hot encoding).
- **Optimizer**: `Adam` — adaptive learning rate optimization suitable for NLP tasks.

- **Metric**: `accuracy` standard classification accuracy.

Regularization: Early Stopping

```
EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
```

- **Why**: To prevent overfitting and long training time.
- **Mechanism**: Monitors the validation loss and stops training if it doesn't improve after 2 consecutive epochs.
- **Restores**: The model weights from the best epoch (lowest validation loss).

Summary Table

Layer Type	Details
Embedding	FastText 300D, trainable, input length = `max_len`
BiLSTM	128 units, bidirectional
Dropout	50%
Dense (hidden)	32 units, ReLU activation
Output Dense	5 units, Softmax activation
Loss Function	Sparse Categorical Crossentropy
Optimizer	Adam
Early Stopping	Patience = 2, monitors validation loss

Word Embeddings: FastText (wiki.simple.vec)

What Are Word Embeddings?

Word embeddings are dense vector representations of words where similar words have similar representations in vector space. Instead of treating words as discrete symbols, embeddings allow the model to **learn semantic relationships** between words.

Which Embedding Is Used?

- **Type**: Pretrained **FastText** vectors
- **File Used**: `wiki.simple.vec`
- **Vector Dimension**: 300
- **Source**: Trained on **Simple English Wikipedia**
- **Format**: `.vec` file with lines like:

  ```
  good  0.1234 -0.5432 ... 300 values
  ```

Why FastText?

FastText is chosen over other embeddings like GloVe or Word2Vec for the following reasons:

Feature	FastText vs Others
Subword Information	FastText captures word parts (prefixes/suffixes), which improves handling of **rare**, **misspelled**, or **compound words**. Neither GloVe nor Word2Vec do this.
Generalization	FastText can generate vectors for **OOV (Out-of-Vocabulary)** words by combining character n-grams — others return `null`.
Domain Relevance	The use of **Simple English Wikipedia** as the training corpus aligns well with product review language: clear, non-technical, and consumer-oriented.
Fast to Load	`.vec` format is human-readable and easy to parse.
Compatibility	Easily converted into an embedding matrix compatible with TensorFlow/Keras models.

Why LSTM? Why Bidirectional LSTM?

What is an LSTM?

LSTM (**Long Short-Term Memory**) is a type of Recurrent Neural Network (RNN) that is designed to **learn patterns in sequential data**, such as text, time series, or speech.

It's particularly good at **remembering long-term dependencies** and **avoiding vanishing gradients**, which is a common issue with standard RNNs.

How Does LSTM Work?

LSTM processes sequences **word by word** (or time step by time step) using a **memory cell** and three key gates:

Gate	Function
Forget Gate	Decides what information to discard from the cell state.
Input Gate	Decides which new information to add to the cell state.
Output Gate	Decides what to output (i.e., what part of memory to pass to the next time step).

Internal Workflow at Each Step:

1. Takes a word/token as input.
2. Updates its **cell state** (memory) based on current input and past memory.
3. Uses the gates to control **information flow**.
4. Moves to the next token and repeats.

This ability to “remember” long-term dependencies helps LSTM capture **context** like:

- “I loved this product although delivery was late.”
  (final label = 4 or 5, despite late delivery)

Why LSTM for This Task?

Reason	Explanation
Text is Sequential	Reviews are sentences or paragraphs — order of words matters. LSTMs are well-suited for this.
Contextual Understanding Needed	To correctly predict a rating, the model needs to understand context (e.g., "not good" vs "good").
Better than CNNs for Long Texts	LSTM captures longer dependencies; CNNs are better for fixed n-gram patterns.
Robust to Varying Lengths	Can handle both short and long reviews effectively.

Why *Bidirectional* LSTM?

A **Bidirectional LSTM** processes input in both directions:

- One LSTM processes text **left to right** (past → future)
- Another processes text **right to left** (future → past)

Then it **concatenates** both outputs to form a **full-context representation**.

Benefits:

- Captures **dependencies from both ends** of the sentence.
- Improves performance in tasks like sentiment analysis, where key clues can appear at any point in the text.
- Handles patterns like:
  - "Not only was it slow, but it also broke" (need forward and backward context)

Summary

Component	Reason for Selection
FastText	Handles OOV words, uses subword units, Simple English Wiki trained
LSTM	Ideal for sequential data, avoids vanishing gradients
Bidirectional	Learns full context, improves accuracy in language tasks

Strengths of LSTM

Strength	Why it Matters
Captures long-term dependencies	Understands meaning from distant words
Avoids vanishing gradient	Learns more effectively over long text
Works well with noisy data	Handles variations in text (spelling, style)
Effective in NLP	State-of-the-art performance in sentiment, translation, etc.
Supports variable input length	Works with both short and long reviews

When to Use LSTM

Use Case	Example
Sentiment analysis	Predicting positive/negative reviews
Text classification	Assigning topics or ratings to text
Sequence prediction	Predicting next word or sequence pattern
Speech and language modeling	Translating sentences, voice recognition
Time series forecasting	Stock market trends, temperature over time

When *Not* to Use LSTM

Scenario	Better Alternative
Very large datasets with long training time	Transformer-based models (e.g., BERT, RoBERTa) are faster and scale better
Highly structured, tabular data	Use tree-based models like XGBoost or Random Forest
Real-time deployment with low latency constraints	LSTMs are slower than CNNs or smaller models
Non-sequential tasks (e.g., image classification)	Use CNNs instead

Limitations of LSTM

Limitation	Impact
Training can be slow	Especially with long sequences
High memory usage	Due to recurrent operations and multiple gates
Hard to parallelize	Each step depends on the previous one
Struggles with extremely long sequences	Even LSTM has limits; Transformers may perform better

Training Progress (Epoch-wise):

Epoch	Accuracy	Loss	Val Accuracy	Val Loss
1	43.26%	1.2804	53.43%	1.0782
2	56.02%	1.0246	54.51%	1.0714
3	59.41%	0.9499	54.67%	1.0857
4	**63.17%**	**0.8698**	**53.93%**	**1.0938** *(Early Stopping)*

Interpretation

- **Model A** shows **progressive improvement** in training accuracy up to Epoch 4.
- Early stopping was triggered due to **no improvement in validation loss** beyond this point.
- **Overfitting is minimal**, as validation accuracy is consistently in the ~54% range.
- Despite being trained on a **perfectly balanced dataset**, the model generalizes **moderately well** even on validation samples