

Report on Text Classification with RNN and Pretrained GloVe Embeddings

1. Introduction

The goal of this assignment is to compare the performance of Recurrent Neural Networks (RNNs) with random embeddings and pretrained GloVe embeddings for text classification tasks. Using the IMDB dataset, which contains movie reviews labeled as either positive or negative, I explored two approaches:

1. Standard RNN: An RNN model trained with randomly initialized word embeddings.
2. Pretrained GloVe RNN: An RNN model with pretrained GloVe word embeddings to leverage the semantic relationships between words.

By training and evaluating both models on different sample sizes, I compared their accuracy and loss values to determine which model performs better.

2. Dataset

IMDB Dataset Overview

The IMDB dataset is a well-known dataset used for sentiment analysis. It contains 25,000 labeled movie reviews for training and 25,000 for testing. Each review is a sequence of words, and the label corresponds to whether the review is positive or negative. The dataset is preprocessed by converting the words into integer sequences based on the frequency of words in the dataset, where the most common 10,000 words are retained.

Data Preprocessing

- Vocabulary Size: The maximum number of words considered is 10,000, ensuring that only the most frequently occurring words are used.
- Sequence Length: Each sequence (review) is padded or truncated to a maximum length of 150 words to maintain consistency in input size for the models.
- Training Subsets: For comparison, three subsets of the training data are considered:
 - 100 samples
 - 500 samples
 - 1000 samples

The data is split into training and testing sets, with the training sets containing the subsets mentioned above, and the test set containing the remaining 25,000 samples.

3. Parameters

Key Parameters for Models:

- Embedding Dimension: 100 (using GloVe embeddings with 100-dimensional word vectors).
- Vocabulary Size: 10,000 words.
- Maximum Sequence Length: 150 words per review.
- Batch Size: 32
- Epochs: 20 (with early stopping to prevent overfitting).
- Dropout Rates: Varying from 0.3 to 0.5 to regularize the models.

Evaluation Metrics:

- Accuracy: The percentage of correct predictions on the test data.
- Loss: The binary cross-entropy loss, which measures how ill the model predicts the sentiment of each review.

4. Models

4.1. Standard RNN with Random Embeddings

The standard RNN model uses randomly initialized word embeddings and consists of the following architecture:

- Embedding Layer: Converts integer sequences into dense word embeddings of dimension 100.
- Bidirectional LSTM Layers: Three bidirectional LSTM layers are used to capture both forward and backward dependencies in the text.
- Dense Layers: After the LSTM layers, dense layers with ReLU activations are used to process the features extracted by the LSTM layers.
- Output Layer: A single neuron with a sigmoid activation function to classify the sentiment (positive/negative).

The architecture is designed to include multiple LSTM layers with dropout layers to prevent overfitting. The dropout rate is set to 0.4 and 0.3, and the LSTM units are set to 32 and 16 for different layers.

4.2. Pretrained GloVe RNN

The GloVe embeddings are pre-trained on a large corpus (6B tokens), capturing semantic relationships between words. The pretrained GloVe model uses the following architecture:

- Embedding Layer: Initialized with the pretrained GloVe embeddings, the embedding layer is either trainable or non-trainable depending on the configuration.
- Bidirectional LSTM Layers: Similar to the standard RNN model, three bidirectional LSTM layers are used.
- Dense Layers: Following the LSTM layers, dense layers with ReLU activation are included.
- Output Layer: A sigmoid activation for binary classification.

This model takes advantage of the semantic properties of words, improving the model's ability to understand and process the relationships between words more effectively than random embeddings.

5. Embedding Layers (GloVe.6B.100d.txt)

GloVe Embeddings

GloVe (Global Vectors for Word Representation) is an unsupervised learning algorithm for obtaining word vectors. The pretrained embeddings, GloVe.6B.100d.txt, contain 100-dimensional vectors for 400,000 words. These embeddings are trained on a large corpus (6 billion tokens) and capture the semantic meaning of words based on their co-occurrence in the corpus. The pretrained embeddings are used to initialize the embedding layer in the pretrained GloVe RNN model.

Embedding Matrix Preparation

The embedding matrix is constructed by mapping each word in the IMDB dataset to its corresponding vector in the GloVe embeddings. If a word is not present in the GloVe vocabulary, its embedding is initialized with a zero vector. This matrix is then used as the inputs for the embedding layer in the model.

6. Training

Training Procedure

- Early Stopping: To prevent overfitting, an early stopping callback is used to halt training if the validation loss doesn't improve for 5 consecutive epochs.
- Optimizers: The Adam optimizer is used for both models, as it adapts the learning rate during training.
- Training Process: Models are trained for 20 epochs, but training is halted early based on the validation performance.

Training Data: The models are trained on subsets of the IMDB training set (100, 500, and 1000 samples) to evaluate how sample size affects model performance.

7. Results and Discussion

Comparative Results:

Sample Size	Standard RNN Accuracy	Pretrained GloVe RNN Accuracy	Standard RNN Loss	Pretrained GloVe RNN Loss
100	0.51104	0.5	0.69298	0.72147
500	0.67636	0.51236	0.63626	0.69311
1000	0.65656	0.51208	0.60727	0.69252

Key Observations:

- Accuracy:
 - The Standard RNN model outperforms the Pretrained GloVe RNN in terms of accuracy for all sample sizes.
 - The accuracy of the Standard RNN increases with the sample size, while the GloVe RNN shows little improvement, remaining around 50% for most of the sample sizes.
- Loss:
 - The Standard RNN consistently has a lower loss compared to the Pretrained GloVe RNN, suggesting that the model is better at minimizing the binary cross-entropy loss.

Discussion:

- Model Performance: Despite using pretrained GloVe embeddings, the Pretrained GloVe RNN did not show a significant improvement in accuracy over the Standard RNN. This could be due to the relatively small dataset sizes (100, 500, 1000 samples), where the pretrained embeddings did not have enough data to shine. The Standard RNN seems to learn effectively even with random embeddings in this case.
- Embedding Contributions: The pretrained embeddings help capture the semantic meaning of words, but with smaller sample sizes, the model may not fully benefit from this. Further training with a larger dataset may show more significant improvements.

8. Conclusion

In this assignment, I trained two RNN models for sentiment analysis using the IMDB dataset: one with random embeddings and the other with pretrained GloVe embeddings. The Standard RNN performed better in terms of accuracy and loss, likely due to the small sample sizes. The pretrained GloVe RNN did not show significant improvements, possibly because the model did not have enough data to fully exploit the semantic relationships in the pretrained embeddings.

Importing Libraries

```
In [1]: from keras.datasets import imdb
        from keras.preprocessing.sequence import pad_sequences
        from keras.models import Sequential
        from keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
        from keras.optimizers import Adam
        from keras.callbacks import EarlyStopping
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.metrics import classification_report
```

Loading IMDB Dataset

```
In [2]: # Load IMDB dataset
vocab_size = 10000
(train_sequences, train_targets), (test_sequences, test_targets) = imdb.load_data(num_

# Set maximum sequence length
sequence_length = 150
train_sequences = pad_sequences(train_sequences, maxlen=sequence_length)
test_sequences = pad_sequences(test_sequences, maxlen=sequence_length)
```

```
In [3]: # Subsets of data for training (100, 500, 1000 samples)
training_sizes = [100, 500, 1000]
train_data_samples = [train_sequences[:size] for size in training_sizes]
train_labels_samples = [train_targets[:size] for size in training_sizes]
```

Loading pretrained GloVe embeddings

```
In [4]: # Load pretrained GloVe embeddings
embedding_index = {}
embedding_dimension = 100
with open('glove.6B.100d.txt', 'r') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefficients = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefficients
```

```
In [5]: # Prepare the GloVe embedding matrix
embedding_matrix = np.zeros((vocab_size, embedding_dimension))
word_index = imdb.get_word_index()
for word, i in word_index.items():
    if i < vocab_size:
        embedding_vector = embedding_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

RNN Model with Random Embedding

```
In [6]: # Build complex RNN model with random embeddings
def build_standard_rnn(input_length):
    model = Sequential()
    model.add(Embedding(vocab_size, embedding_dimension, input_shape=(input_length,)))
    model.add(Bidirectional(LSTM(32, activation='tanh', return_sequences=True)))
    model.add(Dropout(0.4))
    model.add(Bidirectional(LSTM(32, activation='tanh', return_sequences=True)))
    model.add(Dropout(0.4))
    model.add(Bidirectional(LSTM(16)))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(1, activation='sigmoid'))
    model.build(input_shape=(None, input_length)) # Specify input shape here
    model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
    return model
```

RNN Model with Pre trained GloVe embeddings

```
In [7]: # Build enhanced RNN model with pretrained GloVe embeddings
def build_pretrained_rnn(input_length, embedding_matrix=None):
    model = Sequential()
    model.add(Embedding(vocab_size, embedding_dimension, input_shape=(input_length,),
                        weights=[embedding_matrix] if embedding_matrix is not None else
                        trainable=True))
    model.add(Bidirectional(LSTM(64, activation='tanh', return_sequences=True)))
    model.add(Dropout(0.5))
    model.add(Bidirectional(LSTM(64, activation='tanh', return_sequences=True)))
    model.add(Dropout(0.5))
    model.add(Bidirectional(LSTM(32)))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))
    model.build(input_shape=(None, input_length)) # Specify input shape here
    model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
    return model
```

Plotting Function

```
In [8]: # Plotting function to compare accuracy and loss
def plot_metrics(history, model_name, sample_size):
    fig, ax = plt.subplots(1, 2, figsize=(14, 6))

    # Plot accuracy
    ax[0].plot(history.history['accuracy'], label='Training Accuracy', color='teal')
    ax[0].plot(history.history['val_accuracy'], label='Validation Accuracy', color='coral')
    ax[0].set_title(f'{model_name} Accuracy - {sample_size} Samples')
    ax[0].set_xlabel('Epochs')
    ax[0].set_ylabel('Accuracy')
```

```

ax[0].legend()

# Plot Loss
ax[1].plot(history.history['loss'], label='Training Loss', color='teal')
ax[1].plot(history.history['val_loss'], label='Validation Loss', color='coral')
ax[1].set_title(f'{model_name} Loss - {sample_size} Samples')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend()

plt.tight_layout()
plt.show()

```

Model Summary

```

In [9]: model_rnn = build_standard_rnn(sequence_length)
print("Standard RNN Model :")
print(model_rnn.summary())
model_pretrained_rnn = build_pretrained_rnn(sequence_length, embedding_matrix)
print("Pretrained GloVe RNN Model :")
print(model_pretrained_rnn.summary())

```

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(**kwargs)
```

Standard RNN Model :

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 150, 100)	1,000
bidirectional (Bidirectional)	(None, 150, 64)	1,088
dropout (Dropout)	(None, 150, 64)	0
bidirectional_1 (Bidirectional)	(None, 150, 64)	1,088
dropout_1 (Dropout)	(None, 150, 64)	0
bidirectional_2 (Bidirectional)	(None, 32)	1,088
dense (Dense)	(None, 32)	1,024
dropout_2 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 1)	32

Total params: 1,070,337 (4.08 MB)

Trainable params: 1,070,337 (4.08 MB)

Non-trainable params: 0 (0.00 B)

None
Pretrained GloVe RNN Model :
Model: "sequential_1"

Layer (type)	Output Shape	Pa
embedding_1 (Embedding)	(None, 150, 100)	1,00
bidirectional_3 (Bidirectional)	(None, 150, 128)	8
dropout_3 (Dropout)	(None, 150, 128)	
bidirectional_4 (Bidirectional)	(None, 150, 128)	9
dropout_4 (Dropout)	(None, 150, 128)	
bidirectional_5 (Bidirectional)	(None, 64)	4
dense_2 (Dense)	(None, 64)	
dropout_5 (Dropout)	(None, 64)	
dense_3 (Dense)	(None, 1)	

Total params: 1,228,737 (4.69 MB)
Trainable params: 1,228,737 (4.69 MB)
Non-trainable params: 0 (0.00 B)

Model Training

```
In [10]: # Train and evaluate models for different sample sizes
results_comparison = []
for size, data, labels in zip(training_sizes, train_data_samples, train_labels_samples):
    print(f"\nTraining with {size} samples...")

    # Standard RNN with random embeddings
    model_rnn = build_standard_rnn(sequence_length)
    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    rnn_history = model_rnn.fit(data, labels, epochs=20, batch_size=32,
                               validation_data=(test_sequences, test_targets),
                               callbacks=[early_stopping])

    # Evaluate standard RNN
    rnn_loss, rnn_accuracy = model_rnn.evaluate(test_sequences, test_targets)

    # Pretrained GloVe RNN
    model_pretrained_rnn = build_pretrained_rnn(sequence_length, embedding_matrix)
    pretrained_rnn_history = model_pretrained_rnn.fit(data, labels, epochs=20, batch_size=32,
                                                       validation_data=(test_sequences, test_targets),
                                                       callbacks=[early_stopping])

    # Evaluate pretrained GloVe RNN
    pretrained_rnn_loss, pretrained_rnn_accuracy = model_pretrained_rnn.evaluate(test_sequences, test_targets)
```



```


# Store comparison results
results_comparison.append({
    'Sample Size': size,
    'Standard RNN Accuracy': rnn_accuracy,
    'Pretrained GloVe RNN Accuracy': pretrained_rnn_accuracy,
    'Standard RNN Loss': rnn_loss,
    'Pretrained GloVe RNN Loss': pretrained_rnn_loss
})

# Plot metrics for each model
plot_metrics(rnn_history, 'Standard RNN', size)
plot_metrics(pretrained_rnn_history, 'Pretrained GloVe RNN', size)


```

Training with 100 samples...


Epoch 1/20

4/4  22s 4s/step - accuracy: 0.5047 - loss: 0.6942 - val_accuracy: 0.5110 - val_loss: 0.6930


Epoch 2/20

4/4  21s 7s/step - accuracy: 0.5620 - loss: 0.6898 - val_accuracy: 0.5000 - val_loss: 0.6935


Epoch 3/20

4/4  32s 4s/step - accuracy: 0.5808 - loss: 0.6845 - val_accuracy: 0.5000 - val_loss: 0.6955


Epoch 4/20


4/4  21s 4s/step - accuracy: 0.5832 - loss: 0.6735 - val_accuracy: 0.5000 - val_loss: 0.7001

Epoch 5/20


4/4  29s 7s/step - accuracy: 0.5853 - loss: 0.6612 - val_accuracy: 0.5000 - val_loss: 0.7089

Epoch 6/20


4/4  32s 4s/step - accuracy: 0.5987 - loss: 0.6458 - val_accuracy: 0.5000 - val_loss: 0.7280

782/782  12s 16ms/step - accuracy: 0.5165 - loss: 0.6929


Epoch 1/20

4/4  16s 4s/step - accuracy: 0.5499 - loss: 0.6925 - val_accuracy: 0.5000 - val_loss: 0.7215


Epoch 2/20

4/4  19s 3s/step - accuracy: 0.5818 - loss: 0.6736 - val_accuracy: 0.5000 - val_loss: 0.7329


Epoch 3/20


4/4  11s 4s/step - accuracy: 0.6077 - loss: 0.6893 - val_accuracy: 0.5000 - val_loss: 0.7286

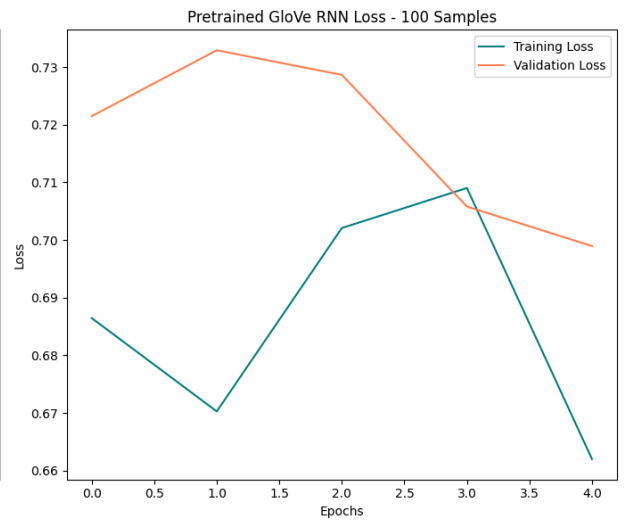
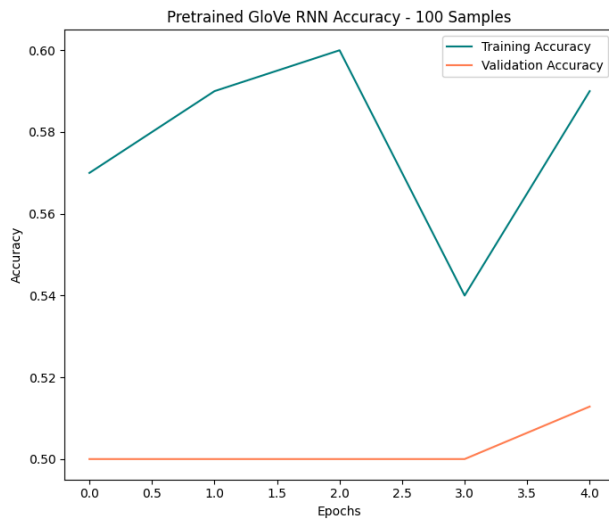
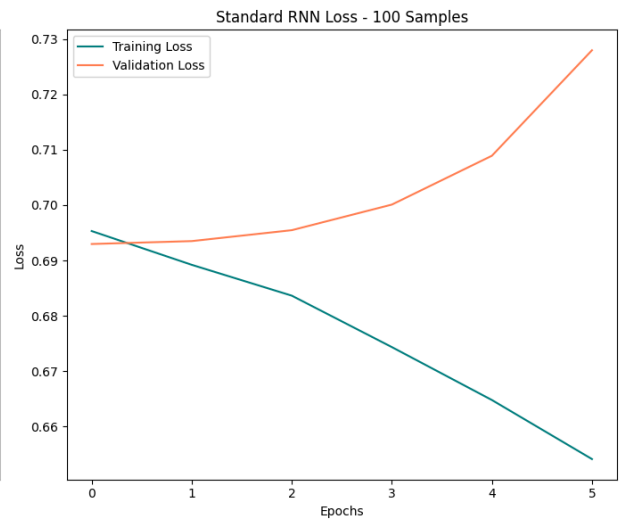
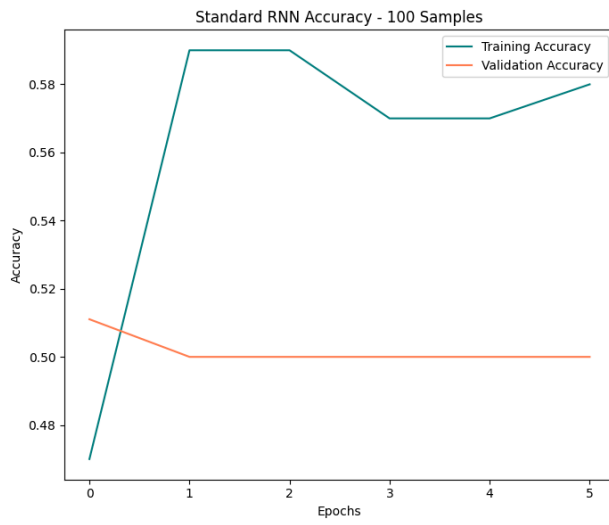
Epoch 4/20

4/4  21s 4s/step - accuracy: 0.5681 - loss: 0.7043 - val_accuracy: 0.5000 - val_loss: 0.7058

Epoch 5/20


4/4  20s 3s/step - accuracy: 0.5745 - loss: 0.6655 - val_accuracy: 0.5128 - val_loss: 0.6989

782/782  13s 16ms/step - accuracy: 0.5073 - loss: 0.7177




Training with 500 samples...


Epoch 1/20

16/16  **17s** 763ms/step - accuracy: 0.4803 - loss: 0.6936 - val_accuracy: 0.5130 - val_loss: 0.6928


Epoch 2/20

16/16  **12s** 780ms/step - accuracy: 0.5570 - loss: 0.6913 - val_accuracy: 0.5191 - val_loss: 0.6923


Epoch 3/20

16/16  **20s** 778ms/step - accuracy: 0.6510 - loss: 0.6829 - val_accuracy: 0.6312 - val_loss: 0.6671


Epoch 4/20

16/16  **11s** 699ms/step - accuracy: 0.8803 - loss: 0.5093 - val_accuracy: 0.6764 - val_loss: 0.6363


Epoch 5/20

16/16  **21s** 1s/step - accuracy: 0.9756 - loss: 0.1864 - val_accuracy: 0.6724 - val_loss: 0.8979


Epoch 6/20

16/16  **30s** 696ms/step - accuracy: 0.9907 - loss: 0.0710 - val_accuracy: 0.6761 - val_loss: 1.1621


Epoch 7/20


16/16  **22s** 774ms/step - accuracy: 1.0000 - loss: 0.0176 - val_accuracy: 0.6772 - val_loss: 1.4369

Epoch 8/20


16/16  **12s** 765ms/step - accuracy: 1.0000 - loss: 0.0091 - val_accuracy: 0.6776 - val_loss: 1.6305

Epoch 9/20


16/16  **21s** 1s/step - accuracy: 1.0000 - loss: 0.0052 - val_accuracy: 0.6787 - val_loss: 1.7716

782/782  **13s** 17ms/step - accuracy: 0.6749 - loss: 0.6382


Epoch 1/20

16/16  **17s** 759ms/step - accuracy: 0.5472 - loss: 0.6927 - val_accuracy: 0.5124 - val_loss: 0.6931


Epoch 2/20

16/16  **21s** 793ms/step - accuracy: 0.5141 - loss: 0.6990 - val_accuracy: 0.5020 - val_loss: 0.6933


Epoch 3/20


16/16  **21s** 1s/step - accuracy: 0.5324 - loss: 0.6951 - val_accuracy: 0.5114 - val_loss: 0.6942

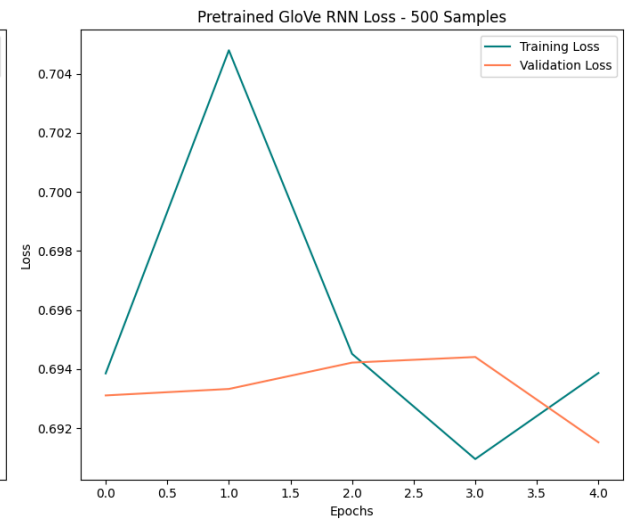
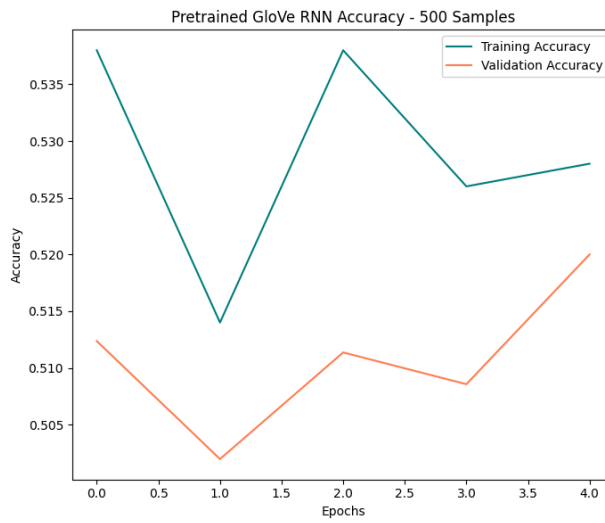
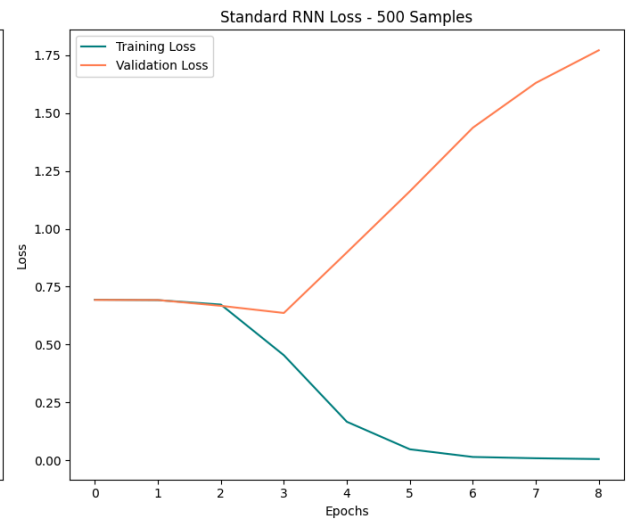
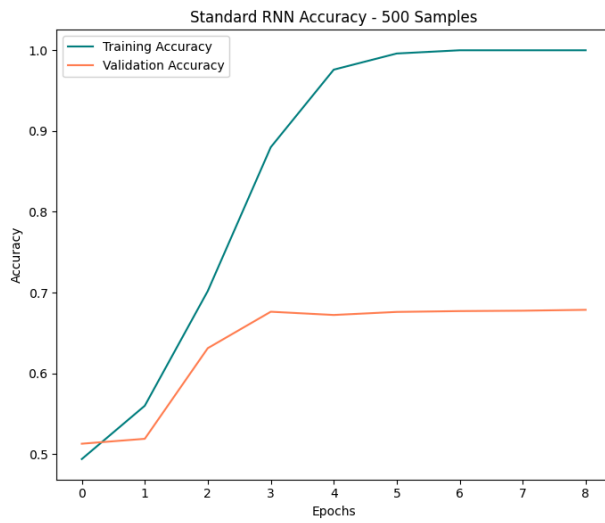
Epoch 4/20

16/16  **32s** 790ms/step - accuracy: 0.5139 - loss: 0.6964 - val_accuracy: 0.5086 - val_loss: 0.6944

Epoch 5/20

16/16  **20s** 728ms/step - accuracy: 0.5228 - loss: 0.6923 - val_accuracy: 0.5200 - val_loss: 0.6915

782/782  **14s** 17ms/step - accuracy: 0.5172 - loss: 0.6932



Training with 1000 samples...

Epoch 1/20

32/32 ————— 18s 398ms/step - accuracy: 0.4975 - loss: 0.6929 - val_accuracy: 0.5505 - val_loss: 0.6891

Epoch 2/20

32/32 ————— 21s 403ms/step - accuracy: 0.7226 - loss: 0.6369 - val_accuracy: 0.6566 - val_loss: 0.6073

Epoch 3/20

32/32 ————— 19s 370ms/step - accuracy: 0.8999 - loss: 0.3141 - val_accuracy: 0.7072 - val_loss: 0.7570

Epoch 4/20

32/32 ————— 21s 401ms/step - accuracy: 0.9784 - loss: 0.0939 - val_accuracy: 0.7224 - val_loss: 1.0543

Epoch 5/20

32/32 ————— 19s 367ms/step - accuracy: 0.9901 - loss: 0.0442 - val_accuracy: 0.7071 - val_loss: 1.1913

Epoch 6/20

32/32 ————— 12s 385ms/step - accuracy: 0.9982 - loss: 0.0115 - val_accuracy: 0.7205 - val_loss: 1.3715

Epoch 7/20

32/32 ————— 21s 397ms/step - accuracy: 1.0000 - loss: 0.0043 - val_accuracy: 0.7216 - val_loss: 1.5753

782/782 ————— 12s 16ms/step - accuracy: 0.6520 - loss: 0.6094

Epoch 1/20

32/32 ————— 19s 417ms/step - accuracy: 0.5151 - loss: 0.7012 - val_accuracy: 0.5121 - val_loss: 0.6925

Epoch 2/20

32/32 ————— 20s 395ms/step - accuracy: 0.5265 - loss: 0.6924 - val_accuracy: 0.5214 - val_loss: 0.6926

Epoch 3/20

32/32 ————— 20s 385ms/step - accuracy: 0.4749 - loss: 0.6973 - val_accuracy: 0.5130 - val_loss: 0.6923

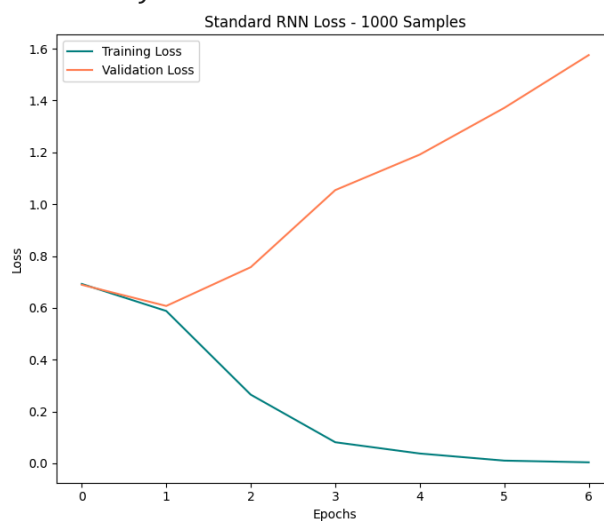
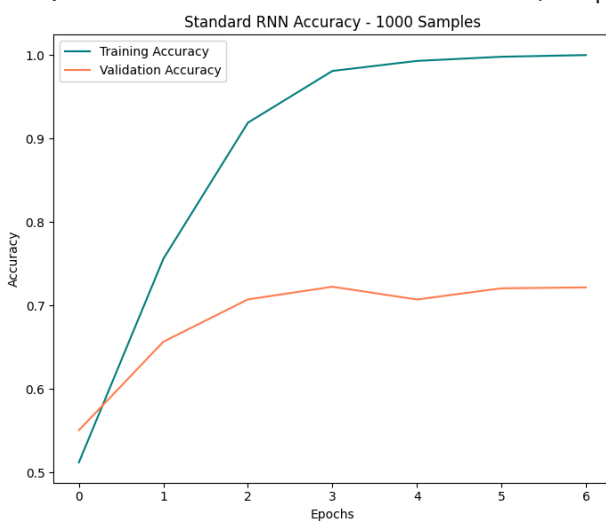
Epoch 4/20

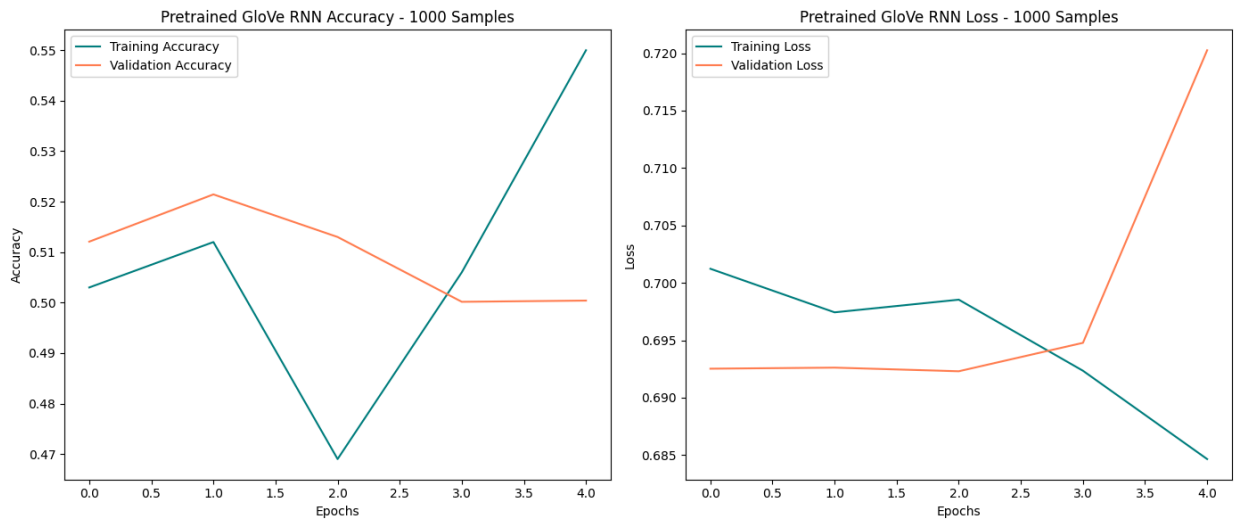
32/32 ————— 21s 396ms/step - accuracy: 0.4931 - loss: 0.6921 - val_accuracy: 0.5002 - val_loss: 0.6948

Epoch 5/20

32/32 ————— 30s 698ms/step - accuracy: 0.5583 - loss: 0.6851 - val_accuracy: 0.5004 - val_loss: 0.7203

782/782 ————— 13s 17ms/step - accuracy: 0.5165 - loss: 0.6923





Model Comparison

```
In [11]: # Create DataFrame for results and visualize
df_results = pd.DataFrame(results_comparison)
print("\nComparative Results:")
print(df_results)
```

Comparative Results:

	Sample Size	Standard RNN Accuracy	Pretrained GloVe RNN Accuracy \
0	100	0.51104	0.50000
1	500	0.67636	0.51236
2	1000	0.65656	0.51208

	Standard RNN Loss	Pretrained GloVe RNN Loss
0	0.692982	0.721475
1	0.636255	0.693111
2	0.607266	0.692524

```
In [12]: # Plot accuracy and loss comparison
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Accuracy comparison
df_results.plot(x='Sample Size', y=['Standard RNN Accuracy', 'Pretrained GloVe RNN Accuracy'],
                kind='bar', ax=axes[0], color=['skyblue', 'salmon'])
axes[0].set_title('Test Accuracy Comparison')
axes[0].set_ylabel('Accuracy')

# Loss comparison
df_results.plot(x='Sample Size', y=['Standard RNN Loss', 'Pretrained GloVe RNN Loss'],
                kind='bar', ax=axes[1], color=['skyblue', 'salmon'])
axes[1].set_title('Test Loss Comparison')
axes[1].set_ylabel('Loss')

plt.tight_layout()
plt.show()
```

