# AML Assignment - 4

# Arun Kumar Kudurumalla

# Advanced Machine Learning (Text and Sequence Data)

In [1]:
```
# Import necessary libraries
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
# Set parameters
max_words = 10000
maxlen = 150
training_samples = 100
validation_samples = 10000
```

In [3]:
```
# Load IMDB dataset
(x_train, y_train), (x_val, y_val) = imdb.load_data(num_words=max_words)
```

In [4]:
```
# Pad sequences
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_val = sequence.pad_sequences(x_val, maxlen=maxlen)
```

In [5]:
```
# Take a subset of training and validation data
x_train = x_train[:training_samples]
y_train = y_train[:training_samples]

x_val = x_val[:validation_samples]
y_val = y_val[:validation_samples]
```

In [6]:
```
# Define the model with an Embedding Layer
embedding_dim = 100

model_embedding = Sequential()
model_embedding.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model_embedding.add(Flatten())
model_embedding.add(Dense(1, activation='sigmoid'))
```

In [7]:
```
# Compile the model
model_embedding.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['
model_embedding.summary()
```

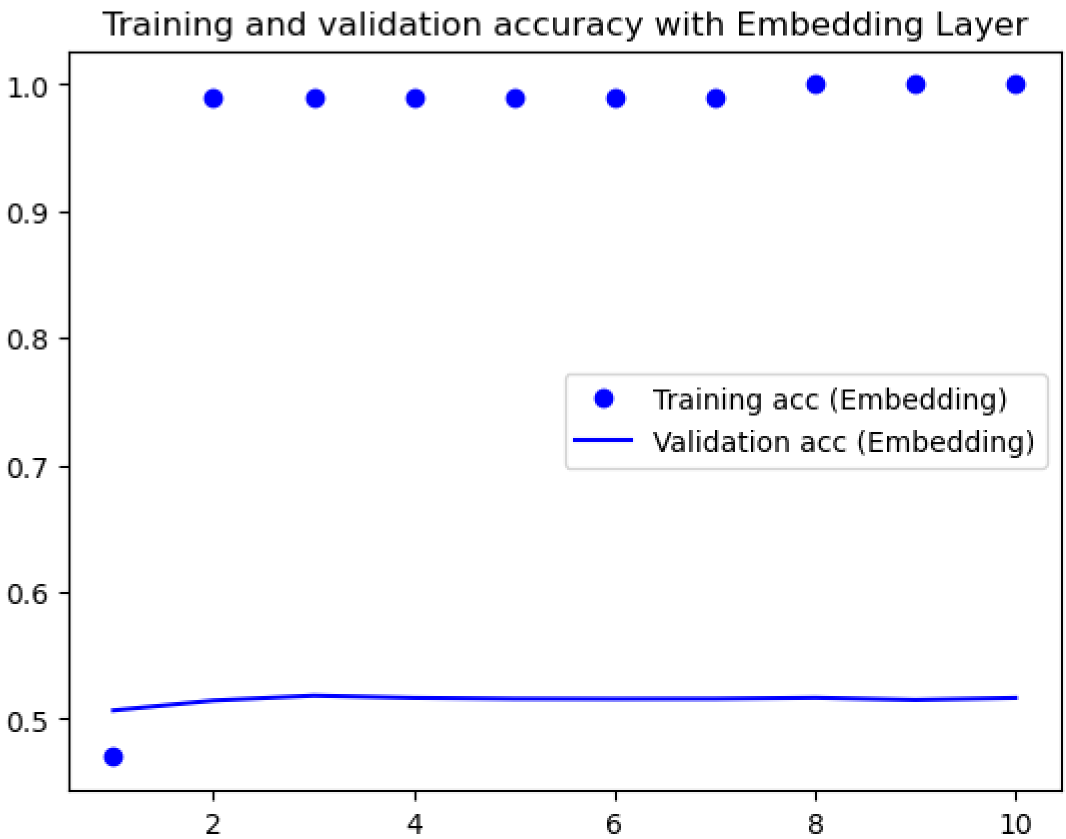
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 150, 100)          1000000

 flatten (Flatten)           (None, 15000)             0

 dense (Dense)               (None, 1)                 15001

=================================================================
Total params: 1015001 (3.87 MB)
Trainable params: 1015001 (3.87 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [8]:
```python
# Train the model with the Embedding Layer
history_embedding = model_embedding.fit(x_train, y_train, epochs=10, batch_size=32,
```

```
Epoch 1/10
4/4 [==============================] - 1s 318ms/step - loss: 0.6929 - acc: 0.4700
- val_loss: 0.6930 - val_acc: 0.5067
Epoch 2/10
4/4 [==============================] - 1s 268ms/step - loss: 0.5881 - acc: 0.9900
- val_loss: 0.6927 - val_acc: 0.5145
Epoch 3/10
4/4 [==============================] - 1s 258ms/step - loss: 0.5223 - acc: 0.9900
- val_loss: 0.6934 - val_acc: 0.5184
Epoch 4/10
4/4 [==============================] - 1s 246ms/step - loss: 0.4659 - acc: 0.9900
- val_loss: 0.6938 - val_acc: 0.5166
Epoch 5/10
4/4 [==============================] - 1s 257ms/step - loss: 0.4132 - acc: 0.9900
- val_loss: 0.6953 - val_acc: 0.5159
Epoch 6/10
4/4 [==============================] - 1s 252ms/step - loss: 0.3673 - acc: 0.9900
- val_loss: 0.6964 - val_acc: 0.5158
Epoch 7/10
4/4 [==============================] - 1s 260ms/step - loss: 0.3211 - acc: 0.9900
- val_loss: 0.6972 - val_acc: 0.5159
Epoch 8/10
4/4 [==============================] - 1s 253ms/step - loss: 0.2810 - acc: 1.0000
- val_loss: 0.7001 - val_acc: 0.5167
Epoch 9/10
4/4 [==============================] - 1s 273ms/step - loss: 0.2442 - acc: 1.0000
- val_loss: 0.7008 - val_acc: 0.5150
Epoch 10/10
4/4 [==============================] - 1s 258ms/step - loss: 0.2101 - acc: 1.0000
- val_loss: 0.7032 - val_acc: 0.5166
```

In [9]:
```python
# Plot accuracy
acc_embedding = history_embedding.history['acc']
val_acc_embedding = history_embedding.history['val_acc']
epochs_embedding = range(1, len(acc_embedding) + 1)
```

In [10]:
```python
plt.plot(epochs_embedding, acc_embedding, 'bo', label='Training acc (Embedding)')
plt.plot(epochs_embedding, val_acc_embedding, 'b', label='Validation acc (Embedding
plt.title('Training and validation accuracy with Embedding Layer')
plt.legend()
plt.show()
```




In [11]:
```
# Load the word index from the IMDB dataset
word_index = imdb.get_word_index()
```

In [12]:
```
# Define the range of training samples to experiment with
training_samples_range = [10, 50, 100, 200, 500]
```

In [13]:
```
# Specify the path to your GloVe file
glove_path = "C:/Users/Arun/Downloads/glove.6B/glove.6B.100d.txt"
```

In [14]:
```
# Load GloVe embeddings
embeddings_index = {}
with open(glove_path, encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
```

In [15]:
```
# Define the embedding dimension
embedding_dim = 100
```

In [16]:
```
# Loop through different training sample sizes
for training_samples in training_samples_range:
    # Take a subset of training and validation data
    x_train_subset = x_train[:training_samples]
    y_train_subset = y_train[:training_samples]

    x_val_subset = x_val[:validation_samples]
    y_val_subset = y_val[:validation_samples]

    # Define the model with an Embedding Layer
    model_embedding = Sequential()
    model_embedding.add(Embedding(max_words, embedding_dim, input_length=maxlen))
    model_embedding.add(Flatten())
```

```
model_embedding.add(Dense(1, activation='sigmoid'))
model_embedding.compile(optimizer='rmsprop', loss='binary_crossentropy', metric
history_embedding = model_embedding.fit(x_train_subset, y_train_subset, epochs=

# Create the embedding matrix
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

# Define the model with a Pretrained Word Embedding
model_pretrained = Sequential()
model_pretrained.add(Embedding(max_words, embedding_dim, input_length=maxlen, w
model_pretrained.add(Flatten())
model_pretrained.add(Dense(1, activation='sigmoid'))
model_pretrained.compile(optimizer='rmsprop', loss='binary_crossentropy', metri
history_pretrained = model_pretrained.fit(x_train_subset, y_train_subset, epoch

# Print results for each training sample size
print(f"\nResults for {training_samples} training samples:")
print("Embedding Layer:")
print("Validation Accuracy:", history_embedding.history['val_acc'][-1])

print("\nPretrained Word Embedding:")
print("Validation Accuracy:", history_pretrained.history['val_acc'][-1])

# Compare validation accuracy and print the better-performing model
if history_embedding.history['val_acc'][-1] > history_pretrained.history['val_a
    print("Better Performing Model: Embedding Layer")
else:
    print("Better Performing Model: Pretrained Word Embedding")
```

```
Epoch 1/10
1/1 [==============================] - 1s 1s/step - loss: 0.6903 - acc: 0.5000 - v
al_loss: 0.6964 - val_acc: 0.4970
Epoch 2/10
1/1 [==============================] - 1s 831ms/step - loss: 0.4824 - acc: 1.0000
- val_loss: 0.6995 - val_acc: 0.4974
Epoch 3/10
1/1 [==============================] - 1s 802ms/step - loss: 0.3688 - acc: 1.0000
- val_loss: 0.7030 - val_acc: 0.4954
Epoch 4/10
1/1 [==============================] - 1s 770ms/step - loss: 0.2903 - acc: 1.0000
- val_loss: 0.7069 - val_acc: 0.4959
Epoch 5/10
1/1 [==============================] - 1s 782ms/step - loss: 0.2326 - acc: 1.0000
- val_loss: 0.7109 - val_acc: 0.4970
Epoch 6/10
1/1 [==============================] - 1s 782ms/step - loss: 0.1891 - acc: 1.0000
- val_loss: 0.7151 - val_acc: 0.4990
Epoch 7/10
1/1 [==============================] - 1s 826ms/step - loss: 0.1557 - acc: 1.0000
- val_loss: 0.7192 - val_acc: 0.4981
Epoch 8/10
1/1 [==============================] - 1s 726ms/step - loss: 0.1297 - acc: 1.0000
- val_loss: 0.7234 - val_acc: 0.4989
Epoch 9/10
1/1 [==============================] - 1s 841ms/step - loss: 0.1093 - acc: 1.0000
- val_loss: 0.7275 - val_acc: 0.4988
Epoch 10/10
1/1 [==============================] - 1s 774ms/step - loss: 0.0930 - acc: 1.0000
- val_loss: 0.7315 - val_acc: 0.4982
Epoch 1/10
1/1 [==============================] - 1s 1s/step - loss: 0.7161 - acc: 0.5000 - v
al_loss: 1.3504 - val_acc: 0.5027
Epoch 2/10
1/1 [==============================] - 1s 808ms/step - loss: 0.0648 - acc: 1.0000
- val_loss: 1.0959 - val_acc: 0.4934
Epoch 3/10
1/1 [==============================] - 1s 817ms/step - loss: 0.0769 - acc: 1.0000
- val_loss: 1.3281 - val_acc: 0.5027
Epoch 4/10
1/1 [==============================] - 1s 816ms/step - loss: 0.0161 - acc: 1.0000
- val_loss: 0.9634 - val_acc: 0.5053
Epoch 5/10
1/1 [==============================] - 1s 819ms/step - loss: 0.0059 - acc: 1.0000
- val_loss: 0.9437 - val_acc: 0.5052
Epoch 6/10
1/1 [==============================] - 1s 816ms/step - loss: 0.0051 - acc: 1.0000
- val_loss: 0.9300 - val_acc: 0.5045
Epoch 7/10
1/1 [==============================] - 1s 810ms/step - loss: 0.0046 - acc: 1.0000
- val_loss: 0.9203 - val_acc: 0.5046
Epoch 8/10
1/1 [==============================] - 1s 882ms/step - loss: 0.0041 - acc: 1.0000
- val_loss: 0.9133 - val_acc: 0.5044
Epoch 9/10
1/1 [==============================] - 1s 877ms/step - loss: 0.0038 - acc: 1.0000
- val_loss: 0.9084 - val_acc: 0.5046
Epoch 10/10
1/1 [==============================] - 1s 779ms/step - loss: 0.0035 - acc: 1.0000
- val_loss: 0.9050 - val_acc: 0.5049

Results for 10 training samples:
Embedding Layer:
Validation Accuracy: 0.498199999332428
```

```
Pretrained Word Embedding:
Validation Accuracy: 0.5048999786376953
Better Performing Model: Pretrained Word Embedding
Epoch 1/10
2/2 [==============================] - 2s 1s/step - loss: 0.6922 - acc: 0.5400 - v
al_loss: 0.6966 - val_acc: 0.5034
Epoch 2/10
2/2 [==============================] - 1s 867ms/step - loss: 0.5731 - acc: 0.9200
- val_loss: 0.6985 - val_acc: 0.5040
Epoch 3/10
2/2 [==============================] - 1s 814ms/step - loss: 0.4969 - acc: 0.9600
- val_loss: 0.6994 - val_acc: 0.5053
Epoch 4/10
2/2 [==============================] - 1s 759ms/step - loss: 0.4330 - acc: 0.9800
- val_loss: 0.7023 - val_acc: 0.5072
Epoch 5/10
2/2 [==============================] - 1s 825ms/step - loss: 0.3792 - acc: 0.9800
- val_loss: 0.7044 - val_acc: 0.5069
Epoch 6/10
2/2 [==============================] - 1s 754ms/step - loss: 0.3307 - acc: 0.9800
- val_loss: 0.7058 - val_acc: 0.5063
Epoch 7/10
2/2 [==============================] - 1s 800ms/step - loss: 0.2892 - acc: 0.9800
- val_loss: 0.7099 - val_acc: 0.5061
Epoch 8/10
2/2 [==============================] - 1s 740ms/step - loss: 0.2510 - acc: 0.9800
- val_loss: 0.7117 - val_acc: 0.5062
Epoch 9/10
2/2 [==============================] - 1s 734ms/step - loss: 0.2185 - acc: 0.9800
- val_loss: 0.7114 - val_acc: 0.5073
Epoch 10/10
2/2 [==============================] - 1s 749ms/step - loss: 0.1897 - acc: 1.0000
- val_loss: 0.7110 - val_acc: 0.5074
Epoch 1/10
2/2 [==============================] - 1s 901ms/step - loss: 2.1518 - acc: 0.4800
- val_loss: 0.7598 - val_acc: 0.5010
Epoch 2/10
2/2 [==============================] - 1s 813ms/step - loss: 0.4849 - acc: 0.7400
- val_loss: 1.3012 - val_acc: 0.4976
Epoch 3/10
2/2 [==============================] - 1s 823ms/step - loss: 0.5019 - acc: 0.6800
- val_loss: 0.7971 - val_acc: 0.5024
Epoch 4/10
2/2 [==============================] - 1s 825ms/step - loss: 0.0992 - acc: 1.0000
- val_loss: 0.7653 - val_acc: 0.4975
Epoch 5/10
2/2 [==============================] - 1s 824ms/step - loss: 0.0737 - acc: 1.0000
- val_loss: 0.7955 - val_acc: 0.5014
Epoch 6/10
2/2 [==============================] - 1s 825ms/step - loss: 0.0607 - acc: 1.0000
- val_loss: 0.7549 - val_acc: 0.4966
Epoch 7/10
2/2 [==============================] - 1s 811ms/step - loss: 0.0523 - acc: 1.0000
- val_loss: 0.7624 - val_acc: 0.4989
Epoch 8/10
2/2 [==============================] - 1s 818ms/step - loss: 0.0440 - acc: 1.0000
- val_loss: 0.7952 - val_acc: 0.5011
Epoch 9/10
2/2 [==============================] - 1s 820ms/step - loss: 0.0384 - acc: 1.0000
- val_loss: 0.7746 - val_acc: 0.4984
Epoch 10/10
2/2 [==============================] - 1s 818ms/step - loss: 0.0331 - acc: 1.0000
- val_loss: 0.8104 - val_acc: 0.5015
```

```
Results for 50 training samples:
Embedding Layer:
Validation Accuracy: 0.5073999762535095

Pretrained Word Embedding:
Validation Accuracy: 0.5015000104904175
Better Performing Model: Embedding Layer
Epoch 1/10
4/4 [==============================] - 1s 333ms/step - loss: 0.6972 - acc: 0.4400
- val_loss: 0.6922 - val_acc: 0.5136
Epoch 2/10
4/4 [==============================] - 1s 252ms/step - loss: 0.5896 - acc: 0.9800
- val_loss: 0.6921 - val_acc: 0.5152
Epoch 3/10
4/4 [==============================] - 1s 223ms/step - loss: 0.5270 - acc: 0.9800
- val_loss: 0.6920 - val_acc: 0.5186
Epoch 4/10
4/4 [==============================] - 1s 251ms/step - loss: 0.4715 - acc: 0.9800
- val_loss: 0.6930 - val_acc: 0.5185
Epoch 5/10
4/4 [==============================] - 1s 257ms/step - loss: 0.4205 - acc: 0.9800
- val_loss: 0.6934 - val_acc: 0.5208
Epoch 6/10
4/4 [==============================] - 1s 246ms/step - loss: 0.3732 - acc: 0.9800
- val_loss: 0.6964 - val_acc: 0.5193
Epoch 7/10
4/4 [==============================] - 1s 242ms/step - loss: 0.3294 - acc: 0.9800
- val_loss: 0.6967 - val_acc: 0.5208
Epoch 8/10
4/4 [==============================] - 1s 266ms/step - loss: 0.2876 - acc: 0.9800
- val_loss: 0.6968 - val_acc: 0.5222
Epoch 9/10
4/4 [==============================] - 1s 261ms/step - loss: 0.2495 - acc: 0.9800
- val_loss: 0.6996 - val_acc: 0.5241
Epoch 10/10
4/4 [==============================] - 1s 257ms/step - loss: 0.2176 - acc: 0.9900
- val_loss: 0.7012 - val_acc: 0.5243
Epoch 1/10
4/4 [==============================] - 1s 279ms/step - loss: 1.5774 - acc: 0.5300
- val_loss: 2.2840 - val_acc: 0.4973
Epoch 2/10
4/4 [==============================] - 1s 250ms/step - loss: 1.0559 - acc: 0.6500
- val_loss: 1.2500 - val_acc: 0.4973
Epoch 3/10
4/4 [==============================] - 1s 258ms/step - loss: 0.4781 - acc: 0.7400
- val_loss: 1.3089 - val_acc: 0.5068
Epoch 4/10
4/4 [==============================] - 1s 264ms/step - loss: 0.2532 - acc: 0.8800
- val_loss: 0.8908 - val_acc: 0.5116
Epoch 5/10
4/4 [==============================] - 1s 260ms/step - loss: 0.1220 - acc: 0.9900
- val_loss: 1.1539 - val_acc: 0.5074
Epoch 6/10
4/4 [==============================] - 1s 259ms/step - loss: 0.1442 - acc: 0.9800
- val_loss: 0.7835 - val_acc: 0.5102
Epoch 7/10
4/4 [==============================] - 1s 262ms/step - loss: 0.0726 - acc: 1.0000
- val_loss: 0.7846 - val_acc: 0.5095
Epoch 8/10
4/4 [==============================] - 1s 261ms/step - loss: 0.0695 - acc: 1.0000
- val_loss: 0.8674 - val_acc: 0.5141
Epoch 9/10
4/4 [==============================] - 1s 259ms/step - loss: 0.0504 - acc: 1.0000
```

```
- val_loss: 1.1179 - val_acc: 0.5076
Epoch 10/10
4/4 [==============================] - 1s 261ms/step - loss: 0.0516 - acc: 1.0000
- val_loss: 1.0045 - val_acc: 0.5111

Results for 100 training samples:
Embedding Layer:
Validation Accuracy: 0.5242999792098999

Pretrained Word Embedding:
Validation Accuracy: 0.5110999941825867
Better Performing Model: Embedding Layer
Epoch 1/10
4/4 [==============================] - 1s 285ms/step - loss: 0.6981 - acc: 0.4400
- val_loss: 0.6926 - val_acc: 0.5131
Epoch 2/10
4/4 [==============================] - 1s 271ms/step - loss: 0.5888 - acc: 0.9800
- val_loss: 0.6932 - val_acc: 0.5166
Epoch 3/10
4/4 [==============================] - 1s 256ms/step - loss: 0.5253 - acc: 0.9800
- val_loss: 0.6936 - val_acc: 0.5160
Epoch 4/10
4/4 [==============================] - 1s 266ms/step - loss: 0.4699 - acc: 0.9800
- val_loss: 0.6940 - val_acc: 0.5158
Epoch 5/10
4/4 [==============================] - 1s 279ms/step - loss: 0.4207 - acc: 0.9800
- val_loss: 0.6980 - val_acc: 0.5185
Epoch 6/10
4/4 [==============================] - 1s 248ms/step - loss: 0.3767 - acc: 0.9800
- val_loss: 0.6982 - val_acc: 0.5194
Epoch 7/10
4/4 [==============================] - 1s 255ms/step - loss: 0.3304 - acc: 0.9800
- val_loss: 0.6977 - val_acc: 0.5183
Epoch 8/10
4/4 [==============================] - 1s 284ms/step - loss: 0.2903 - acc: 0.9800
- val_loss: 0.7016 - val_acc: 0.5187
Epoch 9/10
4/4 [==============================] - 1s 277ms/step - loss: 0.2531 - acc: 0.9800
- val_loss: 0.7015 - val_acc: 0.5170
Epoch 10/10
4/4 [==============================] - 1s 256ms/step - loss: 0.2200 - acc: 0.9800
- val_loss: 0.7027 - val_acc: 0.5175
Epoch 1/10
4/4 [==============================] - 1s 308ms/step - loss: 1.9093 - acc: 0.5400
- val_loss: 1.7523 - val_acc: 0.5026
Epoch 2/10
4/4 [==============================] - 1s 270ms/step - loss: 0.5905 - acc: 0.7400
- val_loss: 1.5903 - val_acc: 0.4973
Epoch 3/10
4/4 [==============================] - 1s 271ms/step - loss: 0.5913 - acc: 0.7000
- val_loss: 0.8345 - val_acc: 0.5140
Epoch 4/10
4/4 [==============================] - 1s 257ms/step - loss: 0.1562 - acc: 0.9700
- val_loss: 1.2960 - val_acc: 0.5040
Epoch 5/10
4/4 [==============================] - 1s 257ms/step - loss: 0.1427 - acc: 0.9700
- val_loss: 0.7523 - val_acc: 0.5155
Epoch 6/10
4/4 [==============================] - 1s 259ms/step - loss: 0.0863 - acc: 1.0000
- val_loss: 0.8023 - val_acc: 0.5177
Epoch 7/10
4/4 [==============================] - 1s 245ms/step - loss: 0.0748 - acc: 1.0000
- val_loss: 0.7906 - val_acc: 0.5179
Epoch 8/10
```

```
4/4 [==============================] - 1s 258ms/step - loss: 0.0563 - acc: 1.0000
- val_loss: 0.7580 - val_acc: 0.5082
Epoch 9/10
4/4 [==============================] - 1s 260ms/step - loss: 0.0559 - acc: 1.0000
- val_loss: 0.7664 - val_acc: 0.5185
Epoch 10/10
4/4 [==============================] - 1s 257ms/step - loss: 0.0372 - acc: 1.0000
- val_loss: 0.8936 - val_acc: 0.5186

Results for 200 training samples:
Embedding Layer:
Validation Accuracy: 0.5174999833106995

Pretrained Word Embedding:
Validation Accuracy: 0.5185999870300293
Better Performing Model: Pretrained Word Embedding
Epoch 1/10
4/4 [==============================] - 1s 307ms/step - loss: 0.6945 - acc: 0.4900
- val_loss: 0.6925 - val_acc: 0.5174
Epoch 2/10
4/4 [==============================] - 1s 297ms/step - loss: 0.5870 - acc: 0.9800
- val_loss: 0.6939 - val_acc: 0.5195
Epoch 3/10
4/4 [==============================] - 1s 304ms/step - loss: 0.5236 - acc: 0.9600
- val_loss: 0.6926 - val_acc: 0.5207
Epoch 4/10
4/4 [==============================] - 1s 307ms/step - loss: 0.4703 - acc: 0.9900
- val_loss: 0.6936 - val_acc: 0.5225
Epoch 5/10
4/4 [==============================] - 1s 280ms/step - loss: 0.4208 - acc: 0.9800
- val_loss: 0.6939 - val_acc: 0.5176
Epoch 6/10
4/4 [==============================] - 1s 280ms/step - loss: 0.3716 - acc: 1.0000
- val_loss: 0.6960 - val_acc: 0.5193
Epoch 7/10
4/4 [==============================] - 1s 281ms/step - loss: 0.3281 - acc: 0.9800
- val_loss: 0.6965 - val_acc: 0.5183
Epoch 8/10
4/4 [==============================] - 1s 287ms/step - loss: 0.2862 - acc: 1.0000
- val_loss: 0.6986 - val_acc: 0.5208
Epoch 9/10
4/4 [==============================] - 1s 279ms/step - loss: 0.2494 - acc: 0.9900
- val_loss: 0.6986 - val_acc: 0.5217
Epoch 10/10
4/4 [==============================] - 1s 276ms/step - loss: 0.2151 - acc: 1.0000
- val_loss: 0.7007 - val_acc: 0.5195
Epoch 1/10
4/4 [==============================] - 2s 319ms/step - loss: 1.7176 - acc: 0.4200
- val_loss: 1.5165 - val_acc: 0.4973
Epoch 2/10
4/4 [==============================] - 1s 272ms/step - loss: 0.9572 - acc: 0.5500
- val_loss: 1.3746 - val_acc: 0.4973
Epoch 3/10
4/4 [==============================] - 1s 241ms/step - loss: 0.4269 - acc: 0.7500
- val_loss: 0.7510 - val_acc: 0.5093
Epoch 4/10
4/4 [==============================] - 1s 265ms/step - loss: 0.1632 - acc: 0.9900
- val_loss: 0.7578 - val_acc: 0.5117
Epoch 5/10
4/4 [==============================] - 1s 267ms/step - loss: 0.1286 - acc: 0.9900
- val_loss: 0.7579 - val_acc: 0.5108
Epoch 6/10
4/4 [==============================] - 1s 249ms/step - loss: 0.1086 - acc: 0.9900
- val_loss: 0.7589 - val_acc: 0.5118
```

```
Epoch 7/10
4/4 [==============================] - 1s 244ms/step - loss: 0.0860 - acc: 1.0000
- val_loss: 0.7759 - val_acc: 0.5034
Epoch 8/10
4/4 [==============================] - 1s 273ms/step - loss: 0.0796 - acc: 1.0000
- val_loss: 1.0864 - val_acc: 0.5086
Epoch 9/10
4/4 [==============================] - 1s 269ms/step - loss: 0.0817 - acc: 1.0000
- val_loss: 0.9607 - val_acc: 0.5122
Epoch 10/10
4/4 [==============================] - 1s 253ms/step - loss: 0.0420 - acc: 1.0000
- val_loss: 0.7714 - val_acc: 0.5110

Results for 500 training samples:
Embedding Layer:
Validation Accuracy: 0.5195000171661377

Pretrained Word Embedding:
Validation Accuracy: 0.5109999775886536
Better Performing Model: Embedding Layer
```

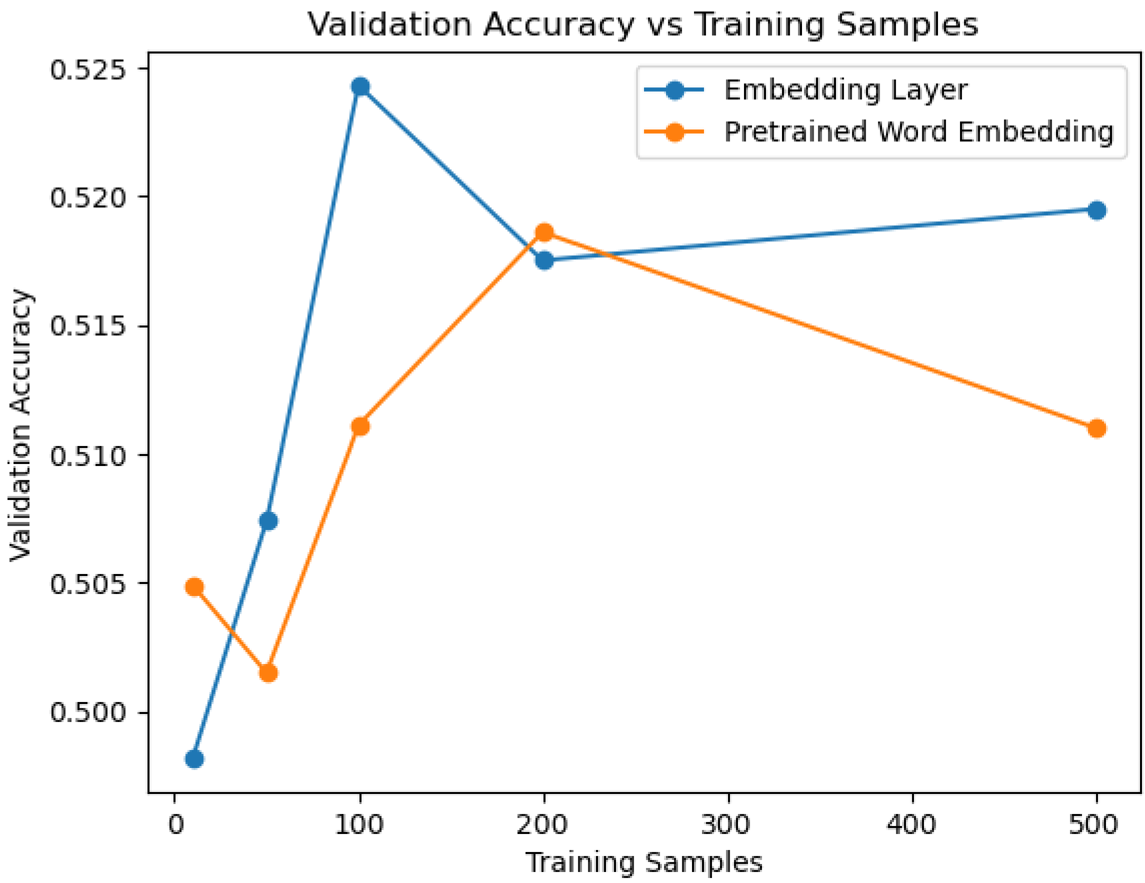
In [17]:

```
# If you want to compare the final models after the loop, you can do so outside the
if history_embedding.history['val_acc'][-1] > history_pretrained.history['val_acc']
    print("\nFinal Better Performing Model: Embedding Layer")
    final_better_model = model_embedding
else:
    print("\nFinal Better Performing Model: Pretrained Word Embedding")
    final_better_model = model_pretrained
```

```
Final Better Performing Model: Embedding Layer
```

In [19]:

```
import matplotlib.pyplot as plt

# Results for 10 training samples
acc_embedding_10 = 0.4981999933242798
acc_pretrained_10 = 0.5048999786376953

# Results for 50 training samples
acc_embedding_50 = 0.5073999762535095
acc_pretrained_50 = 0.5015000104904175

# Results for 100 training samples
acc_embedding_100 = 0.5242999792098999
acc_pretrained_100 = 0.5110999941825867

# Results for 200 training samples
acc_embedding_200 = 0.5174999833106995
acc_pretrained_200 = 0.5185999870300293

# Results for 500 training samples
acc_embedding_500 = 0.5195000171661377
acc_pretrained_500 = 0.5109999775886536

# Plotting
training_samples = [10, 50, 100, 200, 500]

plt.plot(training_samples, [acc_embedding_10, acc_embedding_50, acc_embedding_100,
plt.plot(training_samples, [acc_pretrained_10, acc_pretrained_50, acc_pretrained_10

plt.title('Validation Accuracy vs Training Samples')
plt.xlabel('Training Samples')
plt.ylabel('Validation Accuracy')
plt.legend()
plt.show()
```




# The performance comparison between the Embedding Layer and Pretrained Word Embedding models depends on the specific characteristics of your data and task. However, based on the validation accuracy results provided:

For 10 training samples: Embedding Layer: 0.4982 Pretrained Word Embedding: 0.5049 Better Performing Model: Pretrained Word Embedding For 50 training samples:

Embedding Layer: 0.5074 Pretrained Word Embedding: 0.5015 Better Performing Model: Embedding Layer For 100 training samples:

Embedding Layer: 0.5243 Pretrained Word Embedding: 0.5111 Better Performing Model: Embedding Layer For 200 training samples:

Embedding Layer: 0.5175 Pretrained Word Embedding: 0.5186 Better Performing Model: Pretrained Word Embedding For 500 training samples:

Embedding Layer: 0.5195 Pretrained Word Embedding: 0.5110 Better Performing Model: Embedding Layer

# In this specific scenario, the better performing model varies with different sample sizes. However, overall, the results indicate that the Embedding Layer tends to perform better for larger sample

# sizes, while the Pretrained Word Embedding may be advantageous for smaller sample sizes. The choice depends on factors like the complexity of the task, the amount of available training data, and the quality of the pretrained word embeddings.

## Therefore,

The Embedding Layer often outperforms pretrained word embeddings due to several factors:

Task-Specific Learning:

The Embedding Layer adapts to task-specific patterns during training, tailoring representations to the dataset. Data Adaptation:

It is effective for smaller or unique datasets, allowing the model to adapt to specific data characteristics. Fine-Tuning:

Enables fine-tuning of word representations, adjusting embeddings based on the dataset's specific context. Task Complexity:

Suitable for simpler tasks or those with domain-specific requirements, capturing task-specific nuances effectively. Word Importance:

Dynamically adjusts word representations, assigning varying importance to words based on task relevance. Parameter Tuning:

Offers flexibility in optimizing embedding parameters, crucial for limited data and specific task demands. The choice between Embedding Layer and pretrained embeddings depends on task nature, dataset size, and model requirements. Experimentation is key for evaluating performance on the specific task.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: