Lab 6

Daniel Mehta

Apply all techniques you learned this week to train an LSTM and Bidirectional LSTM network on the "Frankenstein" dataset(Please download the dataset from Kaggle).

Load Data

In [5]:
```
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, LSTM, Dense, Bidirectional
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np
```

```
2025-07-09 16:50:08.831294: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler f
lags.
```

In [6]:
```
df = pd.read_csv("ZC.csv")
```

In [7]:
```
print("Shape of dataset:", df.shape)
print("\nColumn names:", df.columns.tolist())
```

```
Shape of dataset: (86, 2)

Column names: ['name', 'line']
```

In [8]:
```
df.head()
```

Out[8]:

	name	line
0	Human	Human is the most dangerous race
1	Human	We rule the world
2	Human	Time to go to work
3	Human	Where are you from?))))
4	Human	Roadtrip!

Extract Samples

In [10]:
```
# extract line column
lines =df['line'].astype(str).tolist()
```

In [11]:
```
# initiate tokenizer at word level
tokenizer =Tokenizer()
tokenizer.fit_on_texts(lines)
```

In [12]:
```
# generate input output sequence pairs
sequences = []
for line in lines:
    token_list =tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_seq = token_list[:i+1]
        sequences.append(n_gram_seq)
```

In [13]:
```
# pad sequences
max_seq_len=max(len(seq) for seq in sequences)
sequences = pad_sequences(sequences, maxlen=max_seq_len,padding='pre')
```

In [14]:
```
#split into input (X) and label (y)
sequences = np.array(sequences)
X =sequences[:, :-1] # input sequence
y =sequences[:, -1] # target (next word)
```

In [15]:
```
# One hot encode labels
vocab_size = len(tokenizer.word_index) +1
y = tf.keras.utils.to_categorical(y,num_classes=vocab_size)
```

In [16]:
```
print("X.shape:", X.shape)
```

```
print("y.shape:", y.shape)
```

```
X.shape: (370, 14)
y.shape: (370, 224)
```

Train Test split

In [18]:
```
# 90 / 10 split
train_data, test_data, train_labels, test_labels = train_test_split(X, y,test_size=0.1,random_state=5500)
```

In [19]:
```
# Adding 3rd dimension
train_data = np.expand_dims(train_data, axis=-1)
test_data = np.expand_dims(test_data, axis=-1)
```

LSTM

Create Model (LSTM)

In [22]:
```
i = Input(shape=(train_data[0].shape[0], 1)) # Input shape
x = LSTM(128)(i) # 128 LSTM units
x = Dense(vocab_size, activation='softmax')(x) #Predict next word
model = Model(i, x)
```

Complie the model

In [24]:
```
model.compile(optimizer=Adam(learning_rate=0.001), # adaptave
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Train

In [26]:
```
# Train Model
hist = model.fit(train_data, train_labels,
                 validation_data=(test_data, test_labels),
                 epochs=50)
```

```
Epoch 1/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.0081 - loss: 5.4045 - val_accuracy: 0.0000e+00 - val_loss: 5.3388
Epoch 2/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0304 - loss: 5.2111 - val_accuracy: 0.0000e+00 - val_loss: 5.3597
Epoch 3/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.0505 - loss: 5.0595 - val_accuracy: 0.0000e+00 - val_loss: 5.4138
Epoch 4/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.0559 - loss: 4.9003 - val_accuracy: 0.0270 - val_loss: 5.5292
Epoch 5/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0596 - loss: 4.6942 - val_accuracy: 0.0270 - val_loss: 5.7719
Epoch 6/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0712 - loss: 4.6157 - val_accuracy: 0.0270 - val_loss: 5.8215
Epoch 7/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0677 - loss: 4.3761 - val_accuracy: 0.0541 - val_loss: 5.9802
Epoch 8/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0789 - loss: 4.4146 - val_accuracy: 0.0270 - val_loss: 6.0908
Epoch 9/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0718 - loss: 4.2996 - val_accuracy: 0.0541 - val_loss: 6.2518
Epoch 10/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0747 - loss: 4.2151 - val_accuracy: 0.0270 - val_loss: 6.3280
Epoch 11/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1067 - loss: 4.1164 - val_accuracy: 0.0811 - val_loss: 6.4230
Epoch 12/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1322 - loss: 4.0186 - val_accuracy: 0.1081 - val_loss: 6.5281
Epoch 13/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0960 - loss: 3.9469 - val_accuracy: 0.0270 - val_loss: 6.6563
Epoch 14/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1120 - loss: 3.8731 - val_accuracy: 0.0270 - val_loss: 6.6889
Epoch 15/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1026 - loss: 3.7559 - val_accuracy: 0.0811 - val_loss: 6.7617
Epoch 16/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1197 - loss: 3.7183 - val_accuracy: 0.0541 - val_loss: 6.8321
Epoch 17/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.1350 - loss: 3.6674 - val_accuracy: 0.0811 - val_loss: 6.8852
Epoch 18/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.1069 - loss: 3.6696 - val_accuracy: 0.0541 - val_loss: 6.9965
Epoch 19/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1467 - loss: 3.5348 - val_accuracy: 0.0541 - val_loss: 7.0383
Epoch 20/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1495 - loss: 3.4810 - val_accuracy: 0.0541 - val_loss: 7.0758
Epoch 21/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1830 - loss: 3.4213 - val_accuracy: 0.0811 - val_loss: 7.0893
Epoch 22/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.1721 - loss: 3.3639 - val_accuracy: 0.0811 - val_loss: 7.1688
Epoch 23/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1649 - loss: 3.3273 - val_accuracy: 0.0270 - val_loss: 7.2206
Epoch 24/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1651 - loss: 3.2804 - val_accuracy: 0.0541 - val_loss: 7.2808
```

```
Epoch 25/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1960 - loss: 3.2174 - val_accuracy: 0.1081 - val_loss: 7.2842
Epoch 26/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1929 - loss: 3.1922 - val_accuracy: 0.0811 - val_loss: 7.3473
Epoch 27/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2225 - loss: 3.0930 - val_accuracy: 0.0811 - val_loss: 7.4188
Epoch 28/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2404 - loss: 3.0632 - val_accuracy: 0.0811 - val_loss: 7.4474
Epoch 29/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2560 - loss: 3.0247 - val_accuracy: 0.0811 - val_loss: 7.4773
Epoch 30/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2729 - loss: 2.9056 - val_accuracy: 0.0541 - val_loss: 7.5914
Epoch 31/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2435 - loss: 2.8744 - val_accuracy: 0.0541 - val_loss: 7.5379
Epoch 32/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2765 - loss: 2.8808 - val_accuracy: 0.0811 - val_loss: 7.6220
Epoch 33/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2699 - loss: 2.8114 - val_accuracy: 0.0811 - val_loss: 7.7126
Epoch 34/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3027 - loss: 2.7805 - val_accuracy: 0.0811 - val_loss: 7.7017
Epoch 35/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3143 - loss: 2.7336 - val_accuracy: 0.0541 - val_loss: 7.7536
Epoch 36/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3387 - loss: 2.6776 - val_accuracy: 0.0811 - val_loss: 7.7813
Epoch 37/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3497 - loss: 2.6608 - val_accuracy: 0.0811 - val_loss: 7.8389
Epoch 38/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3751 - loss: 2.6184 - val_accuracy: 0.0811 - val_loss: 7.8778
Epoch 39/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.4093 - loss: 2.5478 - val_accuracy: 0.0541 - val_loss: 7.9055
Epoch 40/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3847 - loss: 2.5423 - val_accuracy: 0.0541 - val_loss: 7.9447
Epoch 41/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.4041 - loss: 2.4789 - val_accuracy: 0.0541 - val_loss: 8.0055
Epoch 42/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3915 - loss: 2.4989 - val_accuracy: 0.0811 - val_loss: 8.0527
Epoch 43/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.4050 - loss: 2.4432 - val_accuracy: 0.0811 - val_loss: 8.0592
Epoch 44/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.4220 - loss: 2.4326 - val_accuracy: 0.0541 - val_loss: 8.0879
Epoch 45/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.4240 - loss: 2.3780 - val_accuracy: 0.0811 - val_loss: 8.0973
Epoch 46/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.4185 - loss: 2.3557 - val_accuracy: 0.0270 - val_loss: 8.1428
Epoch 47/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.4401 - loss: 2.3529 - val_accuracy: 0.0541 - val_loss: 8.1892
Epoch 48/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.4765 - loss: 2.2647 - val_accuracy: 0.0541 - val_loss: 8.2489
```

```
Epoch 49/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.4851 - loss: 2.2525 - val_accuracy: 0.0541 - val_loss: 8.2414
Epoch 50/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5252 - loss: 2.1394 - val_accuracy: 0.0811 - val_loss: 8.2973
```

Plot

In [28]:

```
trainAcc = [100 * x for x in hist.history['accuracy']]
testAcc = [100 * x for x in hist.history['val_accuracy']]
```

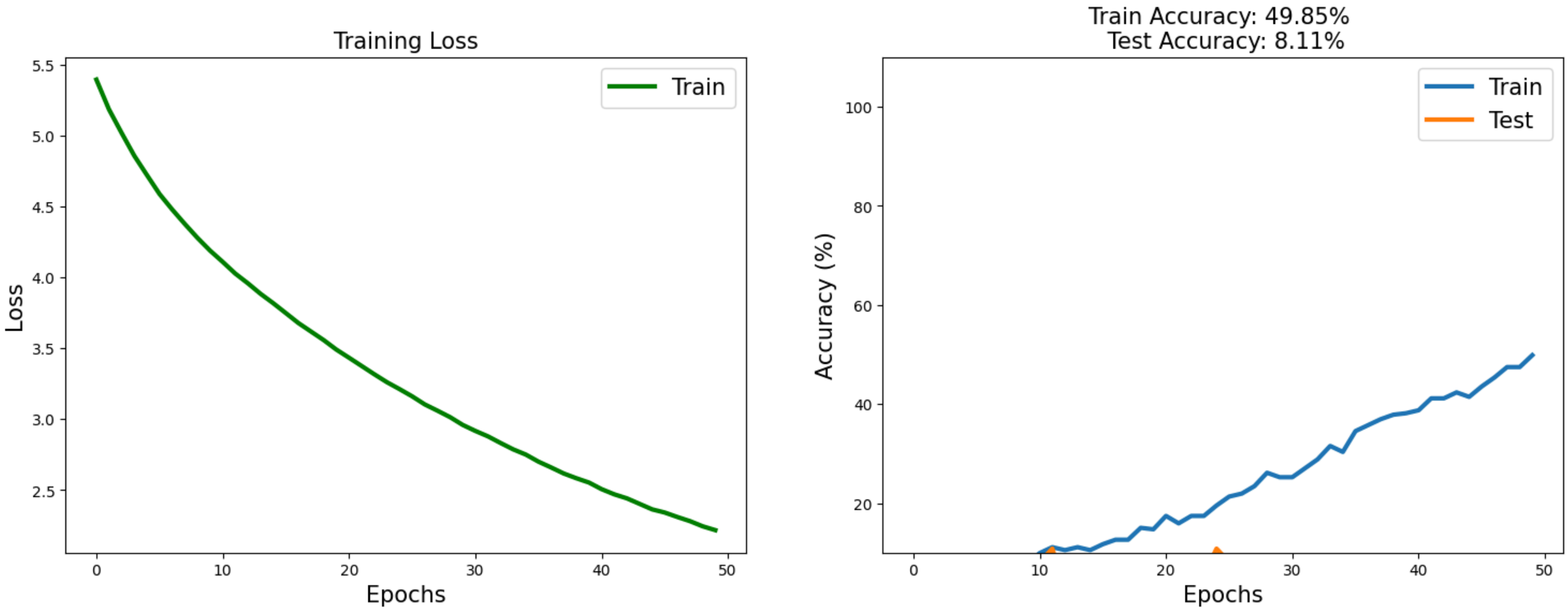
In [29]:

```
fig,ax = plt.subplots(1,2,figsize=(18,6))

# Loss plot
ax[0].plot(hist.history['loss'], 'g', lw = 3, label = 'Train')
ax[0].set_xlabel('Epochs', fontsize = 15)
ax[0].set_ylabel('Loss', fontsize = 15)
ax[0].legend(fontsize = 15)
ax[0].set_title('Training Loss', fontsize = 15)

# Accuracy plot
ax[1].plot(trainAcc, label ='Train', lw = 3)
ax[1].plot(testAcc, label ='Test', lw = 3)
ax[1].set_xlabel('Epochs', fontsize = 15)
ax[1].set_ylabel('Accuracy (%)', fontsize = 15)
ax[1].set_ylim([10,110])
ax[1].set_title(f'Train Accuracy: {trainAcc[-1]:.2f}% \n Test Accuracy: {testAcc[-1]:.2f}%', fontsize = 15)
ax[1].legend(fontsize = 15)

plt.show()
```



Bidirectional LSTM

Create the Model

In [32]:
```
i = Input(shape=(train_data.shape[1], 1)) #shape = (timesteps, features)
x = Bidirectional(LSTM(128))(i)
x = Dense(vocab_size, activation='softmax')(x)
model_bi = Model(i, x)
```

Compile

In [34]:
```
model_bi.compile(optimizer=Adam(learning_rate=0.001),
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
```

Train

In [36]:
```
hist_bi = model_bi.fit(train_data, train_labels,
                       validation_data=(test_data, test_labels),
                       epochs=50)
```

```
Epoch 1/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 7.6092e-04 - loss: 5.4236 - val_accuracy: 0.0270 - val_loss: 5.4406
Epoch 2/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0487 - loss: 5.1533 - val_accuracy: 0.0270 - val_loss: 5.4892
Epoch 3/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0447 - loss: 4.8364 - val_accuracy: 0.0270 - val_loss: 5.7442
Epoch 4/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.0542 - loss: 4.7444 - val_accuracy: 0.0270 - val_loss: 5.8342
Epoch 5/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0607 - loss: 4.6647 - val_accuracy: 0.0270 - val_loss: 6.0166
Epoch 6/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0665 - loss: 4.4977 - val_accuracy: 0.0000e+00 - val_loss: 6.2429
Epoch 7/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0747 - loss: 4.3376 - val_accuracy: 0.0000e+00 - val_loss: 6.4208
Epoch 8/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.0741 - loss: 4.1734 - val_accuracy: 0.0270 - val_loss: 6.6658
Epoch 9/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0998 - loss: 4.0699 - val_accuracy: 0.0270 - val_loss: 6.8010
Epoch 10/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1136 - loss: 3.8809 - val_accuracy: 0.0270 - val_loss: 7.0302
Epoch 11/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1456 - loss: 3.7654 - val_accuracy: 0.0270 - val_loss: 7.2083
Epoch 12/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.1542 - loss: 3.6376 - val_accuracy: 0.0541 - val_loss: 7.3757
Epoch 13/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.1543 - loss: 3.5667 - val_accuracy: 0.0270 - val_loss: 7.4906
Epoch 14/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.1766 - loss: 3.4064 - val_accuracy: 0.0270 - val_loss: 7.6218
Epoch 15/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.2066 - loss: 3.2871 - val_accuracy: 0.0541 - val_loss: 7.6895
Epoch 16/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.2212 - loss: 3.1890 - val_accuracy: 0.0541 - val_loss: 7.9393
Epoch 17/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.2206 - loss: 3.1133 - val_accuracy: 0.0270 - val_loss: 8.0348
Epoch 18/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.2655 - loss: 2.9565 - val_accuracy: 0.0270 - val_loss: 8.0335
Epoch 19/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.2079 - loss: 2.9319 - val_accuracy: 0.0541 - val_loss: 8.1639
Epoch 20/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.3203 - loss: 2.7931 - val_accuracy: 0.0270 - val_loss: 8.2766
Epoch 21/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.3076 - loss: 2.7160 - val_accuracy: 0.0270 - val_loss: 8.3007
Epoch 22/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.2978 - loss: 2.6697 - val_accuracy: 0.0541 - val_loss: 8.3526
Epoch 23/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.3820 - loss: 2.5515 - val_accuracy: 0.0541 - val_loss: 8.4463
Epoch 24/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.3815 - loss: 2.5787 - val_accuracy: 0.0811 - val_loss: 8.5193
```

```
Epoch 25/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.3693 - loss: 2.4750 - val_accuracy: 0.0811 - val_loss: 8.6754
Epoch 26/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4235 - loss: 2.3796 - val_accuracy: 0.0541 - val_loss: 8.6960
Epoch 27/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.4502 - loss: 2.2912 - val_accuracy: 0.0541 - val_loss: 8.8023
Epoch 28/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4728 - loss: 2.2305 - val_accuracy: 0.0541 - val_loss: 8.8454
Epoch 29/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.4918 - loss: 2.1617 - val_accuracy: 0.0541 - val_loss: 8.8685
Epoch 30/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.4758 - loss: 2.1842 - val_accuracy: 0.0811 - val_loss: 8.9037
Epoch 31/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5311 - loss: 2.0808 - val_accuracy: 0.1081 - val_loss: 9.0217
Epoch 32/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5289 - loss: 2.0047 - val_accuracy: 0.0811 - val_loss: 9.0961
Epoch 33/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5091 - loss: 2.0546 - val_accuracy: 0.0811 - val_loss: 9.1094
Epoch 34/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5275 - loss: 1.9140 - val_accuracy: 0.0541 - val_loss: 9.1942
Epoch 35/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5472 - loss: 1.8834 - val_accuracy: 0.0270 - val_loss: 9.2687
Epoch 36/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5585 - loss: 1.8932 - val_accuracy: 0.0811 - val_loss: 9.2492
Epoch 37/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5768 - loss: 1.7891 - val_accuracy: 0.0541 - val_loss: 9.3326
Epoch 38/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5992 - loss: 1.7749 - val_accuracy: 0.0811 - val_loss: 9.3336
Epoch 39/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6071 - loss: 1.7171 - val_accuracy: 0.0270 - val_loss: 9.4768
Epoch 40/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5964 - loss: 1.7235 - val_accuracy: 0.0811 - val_loss: 9.4888
Epoch 41/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5880 - loss: 1.6746 - val_accuracy: 0.0811 - val_loss: 9.4511
Epoch 42/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6263 - loss: 1.6373 - val_accuracy: 0.1081 - val_loss: 9.5503
Epoch 43/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5671 - loss: 1.6667 - val_accuracy: 0.1081 - val_loss: 9.5167
Epoch 44/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6505 - loss: 1.5634 - val_accuracy: 0.1081 - val_loss: 9.5739
Epoch 45/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6574 - loss: 1.4889 - val_accuracy: 0.0811 - val_loss: 9.6025
Epoch 46/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6691 - loss: 1.4630 - val_accuracy: 0.1081 - val_loss: 9.6752
Epoch 47/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6499 - loss: 1.4959 - val_accuracy: 0.1081 - val_loss: 9.7482
Epoch 48/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6385 - loss: 1.4385 - val_accuracy: 0.0811 - val_loss: 9.7342
```

```
Epoch 49/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6654 - loss: 1.4045 - val_accuracy: 0.1081 - val_loss: 9.8140
Epoch 50/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7064 - loss: 1.3496 - val_accuracy: 0.0811 - val_loss: 9.8573
```

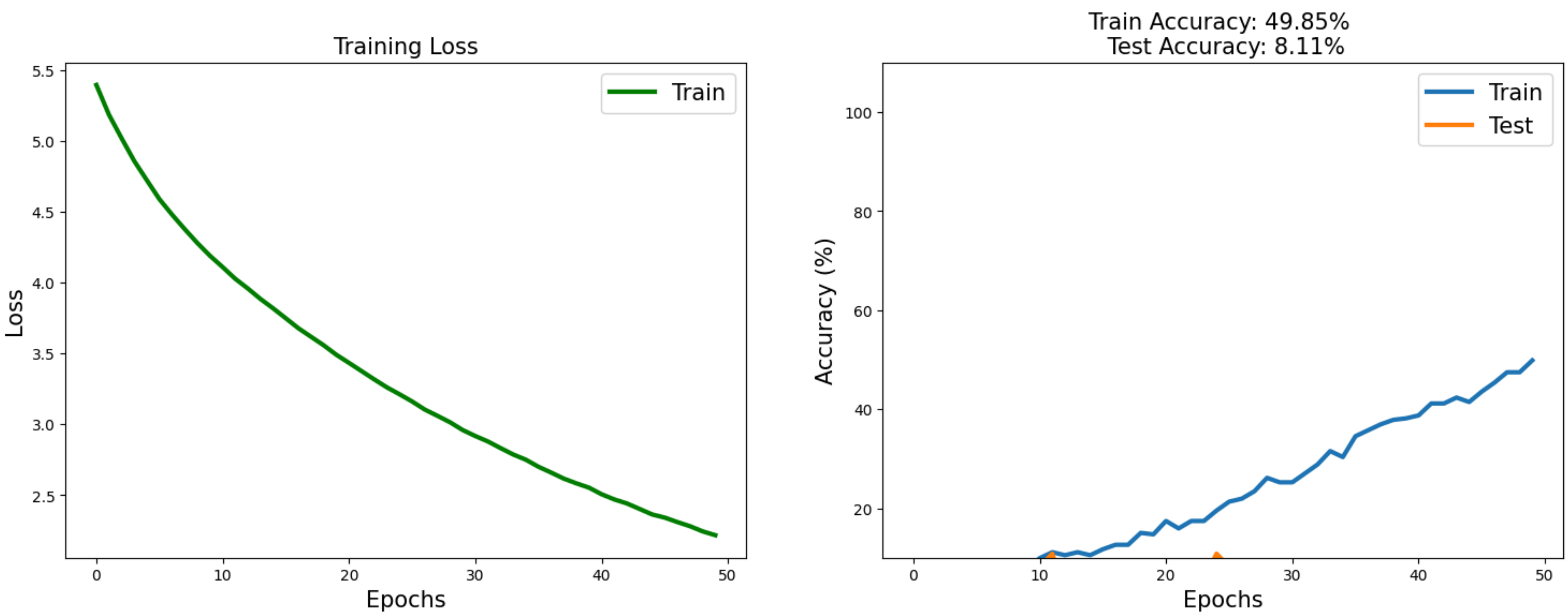
In [37]:

```python
fig,ax = plt.subplots(1,2,figsize=(18,6))

# Loss plot
ax[0].plot(hist.history['loss'], 'g', lw = 3, label = 'Train')
ax[0].set_xlabel('Epochs', fontsize = 15)
ax[0].set_ylabel('Loss', fontsize = 15)
ax[0].legend(fontsize = 15)
ax[0].set_title('Training Loss', fontsize = 15)

# Accuracy plot
ax[1].plot(trainAcc, label ='Train', lw = 3)
ax[1].plot(testAcc, label ='Test', lw = 3)
ax[1].set_xlabel('Epochs', fontsize = 15)
ax[1].set_ylabel('Accuracy (%)', fontsize = 15)
ax[1].set_ylim([10,110])
ax[1].set_title(f'Train Accuracy: {trainAcc[-1]:.2f}% \n Test Accuracy: {testAcc[-1]:.2f}%', fontsize = 15)
ax[1].legend(fontsize = 15)

plt.show()
```



Post Running Notes

- Both models were trained on a small "Frankenstein" dataset (86 English lines)
- Training accuracy improved over time, showing that the models were learning patterns
- Test accuracy remained low, likely due to a small dataset size
- Bidirectional LSTM did not outperform the regular LSTM and Train and Test Accuracy are the same

In [ ]:
```

```