# William_Shakespeare_sonnet

April 28, 2025

## 1 use LSTM neural networks (Long-Short-Term Memory)

### 1.1 in order to tech our computer to write sonnets like William_Shakespeare

[1]:
```
# Importing necessery packages
import random
import numpy as np
import tensorflow as tf
```

```
2025-04-28 16:26:21.390696: I tensorflow/core/util/port.cc:153] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-04-28 16:26:21.419048: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1745837781.452167  211593 cuda_dnn.cc:8579] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1745837781.461863  211593 cuda_blas.cc:1407] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
W0000 00:00:1745837781.486436  211593 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1745837781.486466  211593 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1745837781.486470  211593 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1745837781.486473  211593 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
2025-04-28 16:26:21.493551: 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 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

Data Source: https://github.com/martin-gorner/tensorflow-rnn-shakespeare/blob/master/shakespeare/sonnets.txt

[2]:
```
# Loading the text file
filepath = '/home/abhisek/Project/sonnets.txt'
text = open(filepath, 'rb').read().decode(encoding='utf-8')
```

[3]:
```
text = open(filepath, 'rb').read().decode(encoding='utf-8').lower()
```

[4]:
```
print(f"Total number of characters in the text: {len(text)}")
```

```
Total number of characters in the text: 95662
```

[5]:
```
# SEQ_LENGTH = how many charecters will be used to predict the next character
SEQ_LENGTH = 40

# STEP_SIZE = how many characters we want to shift to next sequence
STEP_SIZE = 3


# Creating empty list of sentences and next characters
sentences = []
next_char = []
```

[6]:
```
# We iterate through the whole text and gather all sentences and their next␣
 ↪character.
# This is the training data for our neural network.
# Now we just need to convert it into a numerical format.

for i in range(0, len(text) - SEQ_LENGTH, STEP_SIZE):
    sentences.append(text[i: i + SEQ_LENGTH])
    next_char.append(text[i + SEQ_LENGTH])
```

[7]:
```
# sorting the characters
characters = sorted(set(text))
```

[8]:
```
# creating two dictionaries from characters to index and from index to␣
 ↪characters
char_to_index = dict((c, i) for i, c in enumerate(characters))
index_to_char= dict((i, c) for i, c in enumerate(characters))
```

[9]:
```
x = np.zeros((len(sentences), SEQ_LENGTH,
              len(characters)), dtype= bool)
y = np.zeros((len(sentences),
```

```
            len(characters)), dtype= bool)

for i, satz in enumerate(sentences):
    for t, char in enumerate(satz):
        x[i, t, char_to_index[char]] = 1
    y[i, char_to_index[next_char[i]]] = 1
```

### 1.2 Building Recurrent Neural Network

[10]:
```
# Importing necessery packages
import random
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.layers import Activation, Dense, LSTM
```

We will use Sequential for our model, Activation, Dense and LSTM for our layers and RMSprop for optimization during the compilation of our model.

[11]:
```
model = Sequential()
model.add(LSTM(128,
               input_shape=(SEQ_LENGTH,
                            len(characters)))) # The inputs immediately flow␣
 ↪into our LSTM layer with 128 neurons
# Our input shape is the length of a sentence times the amount of characters.
model.add(Dense(len(characters))) #This layer is followed by a Dense hidden␣
 ↪layer, which just increases complexity
model.add(Activation('softmax')) # In the end we use the Softmax activation␣
 ↪function in order to make our results add up to one. This gives us the␣
 ↪probability for each character.
```

```
2025-04-28 16:26:26.841001: E
external/local_xla/xla/stream_executor/cuda/cuda_platform.cc:51] failed call to
cuInit: INTERNAL: CUDA error: Failed call to cuInit: UNKNOWN ERROR (303)
/home/abhisek/anaconda3/lib/python3.12/site-
packages/keras/src/layers/rnn/rnn.py:200: 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)
```

We'll now compile and train the model for four epochs using a batch size of 256, meaning the model will iterate through the entire training data four times.

[12]:
```
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(learning_rate=0.01))

model.fit(x, y, batch_size=256, epochs=4)
```

```
model.save('/home/abhisek/Project/sonnets.keras')
```

```
Epoch 1/4
```

```
2025-04-28 16:26:28.980007: E tensorflow/core/util/util.cc:131] oneDNN supports
DT_BOOL only on platforms with AVX-512. Falling back to the default Eigen-based
implementation if present.
```

```
125/125            20s 141ms/step -
loss: 2.9207
Epoch 2/4
125/125            16s 131ms/step -
loss: 2.1991
Epoch 3/4
125/125            17s 133ms/step -
loss: 1.9736
Epoch 4/4
125/125            16s 131ms/step -
loss: 1.8481
```

[13]:
```
model = tf.keras.models.load_model('sonnets.keras')
```

[14]:
```
# Additional functions to make our script generate some reasonable text
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

This function samples a character from the prediction output based on a 'temperature' parameter. Higher temperatures lead to more random (less likely) character choices, while lower temperatures result in more predictable (more likely) selections.

**Generating Text**

[15]:
```
def generate_text(length, temperature):
    start_index = random.randint(0, len(text) - SEQ_LENGTH - 1)
    generated = ''
    sentence = text[start_index: start_index + SEQ_LENGTH]
    generated += sentence
    for i in range(length):
        x_predictions = np.zeros((1, SEQ_LENGTH, len(characters)))
        for t, char in enumerate(sentence):
            x_predictions[0, t, char_to_index[char]] = 1

        predictions = model.predict(x_predictions, verbose=0)[0]
        next_index = sample(predictions,
```

```
                                    temperature)
        next_character = index_to_char[next_index]

        generated += next_character
        sentence = sentence[1:] + next_character
    return generated
```

[16]:
```
# Output
print(generate_text(300, 0.2))
print(generate_text(300, 0.4))
print(generate_text(300, 0.5))
print(generate_text(300, 0.6))
print(generate_text(300, 0.7))
print(generate_text(300, 0.8))
```

```
 and there appears a face
that over-goes thy sweet shall thy sweet on the stell in the steet shall in thy
sweet shall in thy sweet that beauty the sweet,
the stell in the stelf that have the stell thou hear shall thy sweet thy sweet
of thy sweet thy sweet shall thy sweet shall the steet shall thy sweet stell thy
stall thy sweet of thy swe
ning time, whose million'd accidents
creave, thou hade of come conceet than thy sweet with thy love
that shall for thy conseet be when thy stall thy sweet stall in my shall thy eve
the be unor sing the sume's not still in shall the sime's thy sone,
  and the store of hear,
  thin thou be un that thy desseet in that head,
that fare in the
s loss,
and let that pine to aggravate thy seast and my mine but my sive the sume's sime
that grain i my live when with the prowes of thy swall of thinss,
whine of thou wist the prive whe houd thee so not shall thy sweets the stwangs
the stand thus sece,
  shall mise when thy plove that in thy sime,
which hast thou wo shall thy seas love
ill may live in thine or thee.

xi.

as your hand the winter hath to thy some
thou strowe and the sugh thou are sublets when so but of heave, that it wand of
thy such strees and tile in foull of and thou sten'
whes fintens and thou she the shall ofe the crumplecide, thus stour and the see.

nxxvii.

when thile no on stay that in me be not
rive and i be cast away,
```

```
  the worst waste sombeet thoug the frweet, shall thas my our thy flate thy
stouste grount, thy some's dether faul dids beaut my swient
buan my coou hing hourt well,
  mor ally noth thy swale what no shell,
that time prace of thy stoul
when is am well well with thun the sweet,
  the callaan will thou look what no
e thou hadst this more.
then if for my love, amuther coun prfapne, wat her amanntexs?
nive ane, that i hath complease fild shall blom,
in ne, i mine the wiph and then,
cour parugue of thairs shall gays lovk and, of well,
  in the streach stelte  all fear ppouty
the stall my sone;
but in atour wruth hor mohand spppent,
sees of my heast is
```

[17]:
```python
# First, generate all outputs
outputs = []
outputs.append(generate_text(300, 0.2))
outputs.append(generate_text(300, 0.4))
outputs.append(generate_text(300, 0.5))
outputs.append(generate_text(300, 0.6))
outputs.append(generate_text(300, 0.7))
outputs.append(generate_text(300, 0.8))

# Now, save them into a text file
output_path = '/home/abhisek/Project/output_sonnet.txt'
with open(output_path, 'w', encoding='utf-8') as f:
    for i, text in enumerate(outputs):
        f.write(f"Output for temperature {0.2 + 0.2 * i}:\n")
        f.write(text)
        f.write("\n\n" + "-"*50 + "\n\n")  # separator between outputs
```