Quote Generation with LSTM Neural Networks

Div Dasani, dmd8603

Abstract

Language generation is a highly researched problem in artificial intelligence with several real-world implications. This paper presents the LSTM-based model as a way to generate text by learning the patterns between sequences. Using a Kaggle quotes dataset, this network is trained and tested against several other baseline models. Further, variations in LSTM cell stacking are explored to find the tradeoff between accuracy and computational efficiency.

1 Introduction

Statistical language modelling is the broad practice of transforming structured data into traditional language content utilized by humans. One application of such models is natural language generation, wherein machine learning models are trained on a corpus of textual data, and are able to learn the patterns associated with that data, thereby capable of generating this data with only a starting word or seed. The task of building software to produce natural language texts automatically is an ongoing research topic in artificial intelligence and machine learning, as this problem lends itself to many applications. For example, such programs could be utilized to generate instructions or explanations for complex topics automatically, an otherwise arduous and time-consuming task. Another motivating case is presented in the rise of modern voice-based artificial intelligence systems created for personal use, which rely heavily on language generation methods to interact with their users.

2 Data and Preprocessing

For this task, a dataset of quotes from famous authors, artists, and musicians from Kaggle is employed. The dataset contains 36,159 unique quotes from 2,297 different authors. The data is in the form of unstructured text. Each quote possesses on average 23.3 words with a standard deviation of 5.6 words, or 131.6 characters with a standard deviation of 33.4 characters. In preprocessing the data, author names were stripped from their corresponding quotes, as the model is solely trying to learn a variety of different patterns in quotes, but not necessarily attempting to pick up on different styles or distinguish between authors. Furthermore, basic data cleansing is done to make sure that all characters within the corpus are UTF-8 character encoding set. Datum instances where all characters are not part of this set either have noncompliant characters replaced with their UTF-8 counterparts, or the quote instance is discarded if no counterpart exists. Example instances of the data are shown below.

Example Datum Instances
“Impressed with a conviction that the due administration of justice is the firmest pillar of good government, I have considered the first arrangement of the judicial department as essential to the happiness of our country and to the stability of its political system hence the selection of the fittest characters to expound the laws, and dispense justice, has been an invariable object of my anxious concern.”
“There are two sorts of hypocrites: one that are deceived with their outward morality and external religion; and the other, are those that are deceived with false discoveries and elevations; which often cry down works, and men's own righteousness, and talk much of free grace; but at the same time make a righteousness of their discoveries, and of their humiliation, and exalt themselves to heaven with them.”
“And in my own life, in my own small way, I've tried to give back to this country that has given me so much. That's why I left a job at a law firm

for a career in public service, working to empower young people to volunteer in their communities. Because I believe that each of us - no matter what our age or background or walk of life - each of us has something to contribute to the life of this nation.”

Table 1: Example instances of datum from Kaggle quotes dataset

3 Model and Implementation

Recurrent neural networks are a powerful type of artificial neural network with the property that the connections between nodes are directed as a function of time. These algorithms are capable of processing sequences of inputs with an internal memory state, which separates them from their feedforward counterparts in their ability to handle problems where the outputs are time-dependent. One such class of recurrent neural network is the long-short term memory (LSTM) network, an architecture shown to be well suited for predictive problems on time-series data.

3.1 Model Architecture

To accomplish the task of quote generation, an LSTM layer is employed to learn the patterns within the sequences of characters within the corpus. The architecture of the LSTM layer is depicted below.

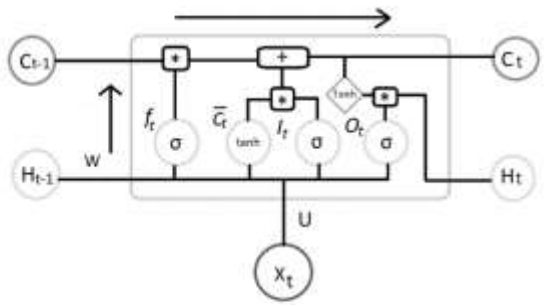

Figure 1: Architecture of LSTM layer for one time-step

Each LSTM cell is composed of four key gates. The forget gate (f) is designed to remove information from the input that the network learns to be unnecessary for predicting the next character. In contrast, the context gate (c) is employed by the cell to “remember” important attributes over longer sequences. Finally, the input (i) and output (o) gates are used to create the hidden state and move the output from one time-step to the next. Here, the input vector at time t is given by $\boldsymbol{x_t}$. The output of an LSTM cell at time t is given by $\boldsymbol{h_t}$, while the memory of the cell at time t is given by $\boldsymbol{c_t}$. The weight vector for the input for gate j is given by $\boldsymbol{w_j}$, while the weight vector for the hidden state for gate j is given by $\boldsymbol{u_j}$. σ denotes the sigmoid activation function, which is given by,

$$\sigma(\mathrm{x}) = (1 + e^{-x})^{-1}$$

while $tanh$ describes the tanh activation function, which is given by,

$$\tanh(\mathrm{x}) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Let ⊗ denote element-wise multiplication of vectors and ⊕ denote element-wise addition of vectors. Then, the functions between the cells and the activation gates are given mathematically as,

$$f_t = \sigma\big(\boldsymbol{x_t} \otimes \big(\boldsymbol{u_f} \oplus \boldsymbol{h_{t-1}}\big) \otimes \boldsymbol{w_f}\big)$$

$$\bar{C}_t = tanh(\boldsymbol{x_t} \otimes (\boldsymbol{u_c} \oplus \boldsymbol{h_{t-1}}) \otimes \boldsymbol{w_c})$$

$$I_t = \sigma(\boldsymbol{x_t} \otimes (\boldsymbol{u_i} \oplus \boldsymbol{h_{t-1}}) \otimes \boldsymbol{w_i})$$

$$O_t = \sigma(\boldsymbol{x_t} \otimes (\boldsymbol{u_o} \oplus \boldsymbol{h_{t-1}}) \otimes \boldsymbol{w_o})$$

$$C_t = f_t * C_{t-1} + I_t * \bar{C}_t$$

$$H_t = O_t * \tanh(C_t)$$

Following the LSTM layer, a fully-connected layer is employed to map every unique character type within the corpus to an output neuron. Once the values for each of these characters is computed in the dense layer, they are transformed into a distribution of probability values via the softmax activation function. Let $\boldsymbol{z} \in \mathbb{R}^K$ denote the vector output of the dense layer. The softmax output for neuron k, $z_k \in \{z_1, \dots, z_K\}$ is given by,

$$\mathrm{softmax}(\boldsymbol{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

In training the model, the cross-entropy loss function is utilized. The network is provided the current input as well as all previous ground truth outputs and is tasked with correctly predicting the next element (in this case character) in the sequence. Let y_t denote the ground truth character at time t in the sequence, and consequently let $\boldsymbol{y}_{<t}$ denote the ground truth vector for all characters in the sequence before time t.Then, the cross-entropy function is given by,

$$l([\boldsymbol{w};\boldsymbol{u}]) = -\sum_{t=1}^{T} \log[P(y_t | \mathrm{x}, \boldsymbol{y}_{<t}; [\boldsymbol{w};\boldsymbol{u}])]$$

3.2 Implementation Details

The model described above is trained with an RMSProp optimizer with an adaptive learning rate. Because the dataset is too large to be stored in memory, and to increase the convergence speed on the algorithm, gradient updates are computed in stochastic minibatches with a batch size of 256. The quotes dataset is composed of 61 unique characters, and the number of timesteps to be predicted has been set to 20 in this paper. Thus, the input shape of the LSTM network is (256, 20, 61). The output of the network is of the shape (256, n), where n is the number of stacked LSTM layers in the model.

One of the biggest dilemmas in text generation is the tradeoff between coherency and diversity. Often, a well-trained model is capable of generating coherent content, but this generated sequence is often identical or nearly identical to the sequences that the model was trained on, an attribute which stifles any room for creativity or originality within the generation process. Conversely, allowing the model to select characters it is uncertain about increases the probability of unique generations, but at the cost of less coherent generations. To balance this tradeoff, the diversity parameter $T \in (0,1]$ is employed, which is used to modify the softmax function when testing the model as,

$$\mathrm{softmax}(\boldsymbol{z})_k^D = \frac{e^{z_k/T}}{\sum_{j=1}^{K} e^{z_j/T}}$$

As T is increased, the distribution of the output samples tends toward the uniform distribution, and therefore all output samples have similar selection probabilities. In contrast, decreasing T increases the sensitivity of the expected reward computed by the dense layer, thereby increasing the differences in probability selection of output samples. A consequence of this is that low values of T will allow the model to be less conservative in choosing the next output character, increasing the diversity of the output sample while increasing the likelihood of error. On the other hand, high values of T forces the algorithm to be conservative, increasing the probability of guessing the next character correctly but at the cost of reducing the diversity of the output sample.

4 Evaluation and Results

The LSTM architecture with various numbers of stacked layers are tested against different baseline models. The first of these is *KN5,* a standard 5-gram Kneser-Ney language model proposed by Heafield et al., while the second is *NeuralEditor*, a state-of-the-art model created by Guu et al. to generate natural language sentences. To evaluate the performance of these models and compare them against one another, 20% of the Kaggle quotes dataset is held out as test data, and the perplexity score on this data is measured for each of the models. Perplexity is a measure that describes how well a probability model is capable of predicting a given sample. Probability distributions that possess low perplexity scores are better at predicting a given sample than those with high perplexity scores for the same sample. Let q denote a potential probability model. The perplexity score for q over a sample $X = (x_1, \ldots, x_n)$ is given by,

$$PP(X) = \prod_{i=1}^{n} q(x_i)^{-1/n}$$

Let LSTM-n denote the model with architecture described as in Section 3.1 with n stacked LSTM layers. The perplexity scores for each of the tested models along with the two baseline models is highlighted in the table below. Note that the diversity parameter is set to $T = 1$ for all models where applicable.

Model	**Perplexity**
KN5	80.42
NeuralEditor	**37.05**
LSTM-1	64.93
LSTM-2	50.50
LSTM-5	48.97

Table 2: Perplexity scores of various models on test portion of Kaggle quotes dataset

An example output instance is provided below for one of the quotes in the test dataset. The seed provided to all models in this instance is “let it be”. Each model is set to output 85 characters following the seed, so the ground truth quote has been truncated to reflect this policy.

Model	**Output**
Ground Truth	`let it be your constant method to look into the design of people's actions, and see what they…`
KN5	`let it be the same stage and the most the same start of the same stage and the same start of th`
NeuralEditor	`let it be a main-self than any direction, and i am a great reasons are correndson's going to ma`
LSTM-1	`let it being first good worked a good world. i'm insure or in the world as my caded about our`
LSTM-2	`let it be the same society and say that is a slegs all the facts are a most be a great stranges`
LSTM-5	`let it be a man's bears and any other than the bad be the same things and a street for me be th`

Table 3: Example output generations from various models using “let it be” test seed.

5 Analysis of Results

As illustrated in Table 2, while the variations of the model described in this paper were able to outperform the *KN5* model, they performed significantly worse than the state-of-the-art *NeuralEditor* model. The former result is likely due to the fact that the LSTM-based architecture is far more computationally expensive than the *KN5* architecture, which at its core employs the bag-of-words model to predict output samples. One of the biggest criticisms of this model type is that it assumes independence between sequences of attributes, which is not true of most languages, and this is likely what caused the model to perform so poorly. Anecdotally this is depicted in Table 3, where the lack of “memory” possessed by the *KN5* model causes it to repeat several words and phrases multiple times.

In contrast, the *NeuralEditor* model significantly outperforms the LSTM-based models. This architecture first samples sequences from the training data and employs these to generate new sequences, rather than doing so from scratch like traditional models. This is likely why the *NeuralEditor* model is so powerful; in contrast, the LSTM-based model relies on the previously generated element in the sequence to produce the next element in the sequence, which causes the output to devolve further from the ground truth as the length of the sequence increases.

Finally, another interesting result is that the gains made from increasing the number of layers in the LSTM-*n* models decrease dramatically as *n* increases. While moving from one LSTM layer to two significantly lowered the perplexity score of the model, increasing this number to five had very little effect on the ability of the model. This is likely due to the fact that the model is learning the majority of the patterns in the sequences early in the LSTM sequence, thereby displaying only marginally better results in later cells.

6 Conclusion

LSTM-based neural networks are powerful tools to handle unstructured generative problems. Here, an LSTM architecture was employed to solve the task of generating quotes after training on a dataset consisting of these quotes. While this architecture performed decently in contrast to simpler models, it still underperforms compared to more advanced state-of-the art algorithms. Furthermore, the use of LSTM cells in sequence can allow for increased learning of patterns within sequences, but this effect diminishes rapidly as the number of stacked cells increases, creating a tradeoff between accuracy and computational efficiency.

References

Guu, Kelvin et al. "Generating Sentences by Editing Prototypes." Transactions of the Association of Computational Linguistics 6 (2018): 437-450.

Zhou, Ming et al. "Learning to Generate Product Reviews from Attributes." EACL (2017).

Heafield, Kenneth et al. "Scalable Modified Kneser-Ney Language Model Estimation" Association of Computational Linguistics (2013).

Xie, Ziang "Neural Text Generation: A Practical Guide" Stanford (2018).

Keras documentation:
https://keras.io/layers/recurrent/

Kaggle quotes dataset:
https://www.kaggle.com/coolcoder22/quotes-dataset