ISSN: 2664-9640

Generating Text Using LSTMs

Xin Lu

School of computer Science, Yangtze University, Jingzhou 434023, China

Abstract

The LSTM layer is a variant of the SimpleRNN layer that adds a way to carry information across multiple time steps. Suppose there is a conveyor belt that runs parallel to the sequence you are processing. The information in the sequence can jump onto the conveyor belt at any point, be transported to a later time step, and jump back intact when needed. That's actually how LSTM works: it saves information for later use, preventing earlier signals from fading away during processing. Take text generation as an example, but the same techniques can be generalized to any kind of sequence data, and can be applied to sequences of notes to generate new music, or to time series of stroke data (for example, when an artist paints on an iPad) recorded stroke data) to generate the painting stroke by stroke, and so on. Sequence data generation is by no means limited to artistic content generation. It has been successfully applied to speech synthesis and dialogue generation for chatbots. Google 's Smart Reply feature, released in 2016 , which automatically generates a set of quick replies to emails or text messages, uses a similar technology.

Keywords

LSTM; long short-term memory algorithm; generated text.

1. Introduction

As of late 2014 , not many people have seen the acronym LSTM[1], even in the field of machine learning. Successful applications of generating sequence data with recurrent networks only started to appear in the mainstream in 2016 . However, these techniques have a long history, the earliest being the LSTM algorithm developed in 1997 . This new algorithm was used early on to generate text character by character.

In 2002 , Douglas Eck , who was then working in Schmidhuber 's laboratory in Switzerland, first applied LSTM to music generation with satisfactory results[2]. Eck is now a researcher at Google Brain, where he created a new research group called Magenta in 2016 , focusing on applying modern deep learning techniques to making captivating music. Sometimes good ideas take 15 years to turn into practice.

In the late 20th and early 2000s, Alex Graves did important pioneering work on generating sequence data using recurrent networks [3]. In particular, his 2013 work, applying a recurrent mixed density network to generate human-like handwriting using a time series of stroke positions, was considered a turning point. At that particular moment, in this specific application of neural networks, the concept of a dreaming machine provided an important inspiration for the development of Keras . Graves left a similarly annotated comment in a La TeX file uploaded to the preprint server arXiv in 2013 : " Sequence data generation is the closest thing a computer can do to a dream."

Since then, recurrent neural networks have been successfully applied to music generation, dialogue generation, image generation, speech generation, and molecular design. It's even been used to create screenplays, which are then performed by live actors.

ISSN: 2664-9640

2. How to generate sequence data

A common approach to generating sequence data with deep learning is to use the preceding tokens as input and train a network (usually a recurrent neural network or a convolutional neural network) to predict the next token or tokens in the sequence. For example, given the input the cat is on the ma, train the network to predict the target t, the next character. As before when dealing with text data, tokens are usually words or characters, and any network that can model the probability of the next token given the previous tokens is called a language model. Language models are able to capture the latent space of language, the statistical structure of language.

Once such a language model is trained, it can be sampled from it (ie, generating new sequences). Feed the model an initial string of text (i.e. conditioning data), ask the model to generate the next character or the next word (even multiple tokens at once), and then add the generated output to the input data, and repeat the process several times. This loop can generate sequences of arbitrary length that reflect the structure of the model's training data, which are nearly identical to sentences written by humans. In the examples in this section, we will use an LSTM layer, feed it a string of N characters extracted from a text corpus , and train the model to generate the N+ 1th character. The output of the model is a softmax of all possible characters to get the probability distribution of the next character. This LSTM is called a character-level neural language model[4].

3. The importance of sampling strategy

When generating text, how the next character is selected is critical. A simple method is greedy sampling, which is to always choose the next most likely character. But this approach results in repeated, predictable strings that don't look like coherent language. A more interesting approach is to make a slightly unexpected choice: introduce randomness into the sampling process, i.e. sample from the probability distribution of the next character. This is called stochastic sampling (stochasticity means "random" in this field). In this case, based on the model results, if the next character is an e with a probability of 0.3, then you have a 30% probability of choosing it. Note that greedy sampling can also be thought of as sampling from a probability distribution, i.e. one character has probability 1 and all other characters have probability 0[5].

From the softmax output of the model is a neat way to even sample uncommon characters at some point, resulting in sentences that look more interesting, and sometimes get things that aren't in the training data, Show creativity by sounding like real new words. But one problem with this method is that it cannot control the amount of randomness in the sampling process.

Why do you need some randomness? Consider an extreme example - pure random sampling, where the next character is drawn from a uniform probability distribution, where each character has the same probability. This scheme has maximum randomness, in other words, this probability distribution has maximum entropy. Of course, it doesn't generate any interesting content. Let's look at the other extreme - greedy sampling. Greedy sampling also does not generate anything interesting, it does not have any randomness, i.e. the corresponding probability distribution has minimal entropy. Sampling from the "true" probability distribution (that is, the distribution output by the model 's softmax function) is an intermediate point between these two extremes. However, there are many other intermediate points with greater or lesser entropy that you may wish to look into . Smaller entropy results in a more predictable structure to the generated sequence (and thus may appear more realistic), while larger entropy results in a more unexpected and creative sequence. When sampling from generative models, it is always good practice to explore different randomness sizes during generation. We humans

ISSN: 2664-9640

are the ultimate judges of whether the generated data is interesting, so interesting is very subjective and we cannot know in advance where the best entropy is.

In order to control the amount of randomness in the sampling process, we introduce a parameter called softmax temperature , which is used to represent the entropy of the sampling probability distribution, that is, how unexpected or predictable the next character chosen will be . Given a temperature value, the original probability distribution (ie, the model's softmax output) will be re-weighted as follows, and a new probability distribution will be calculated.

The probability distribution for different softmax temperatures.

original_distribution is a one-dimensional Numpy array of probability values that must sum to 1. temperature is a factor that quantitatively describes the entropy of the output distribution.

Returns the reweighted result of the original distribution. The sum of the distributions may no longer equal 1, so it needs to be divided by the sum to get the new distribution.

More unexpected and unstructured generated data, while lower temperatures correspond to less randomness and more predictable generated data (i.e. lower temperature = more deterministic, higher temperature = more random).

4. Implement character-level LSTM text generation

The first need is a large amount of text data that can be used to learn a language model. We can use one or more text files arbitrarily large enough - Wikipedia, Lord of the Rings, etc. This example will use some of the works of Nietzsche, the German philosopher of the late 19th century, which have been translated into English. Therefore, the language model we are going to learn will be a model specific to Nietzsche's writing style and themes, not a general model for English[6].

4.1. Prepare data

First download the corpus and convert it to lowercase.

Download and parse the initial text file.

Next, we want to extract length maxlen (there is partial overlap between these sequences), one-hot encode them, and pack them into a 3D Numpy array of shape (sequences, maxlen , unique_characters). At the same time, we also need to prepare an array y, which contains the corresponding target, that is, the characters (one-hot encoded) that appear after each extracted sequence.

Vectorize character sequence.

A list of unique characters in the corpus.

A dictionary mapping a unique character to its index in the list chars.

Encode characters one-hot into binary array.

4.2. Building a network

This network is a single-layer LSTM followed by a Dense classifier and softmax over all possible characters . Note, however, that recurrent neural networks are not the only method for sequence data generation, and it has recently been demonstrated that 1D convolutional neural networks can also be successfully used for sequence data generation.

A single-layer LSTM model for predicting the next character.

The target is one-hot encoded, so training the model requires categorical_crossentropy as a loss.

Model compilation configuration.

ISSN: 2664-9640

4.3. Train the language model and sample from it

Given a trained model and a seed text segment, we can generate new text by repeating the following operations: (1) Given the text that has been generated so far, get the probability distribution of the next character from the model;

(2) Reweight the distribution according to a certain temperature;

(3) Randomly sample the next character according to the reweighted distribution;

(4) Add new characters to the end of the text.

The following code will reweight the original probability distribution obtained by the model and extract a character index from it (sampling function).

A function to sample the next character given a model prediction.

Finally, the following loop will iteratively train and generate text. A series of different temperature values are used to generate text after each round. This way we can see how the generated text changes as the model converges, and how temperature affects the sampling strategy.

Text generation loop.

Train the model for 60 epochs.

Fit the model to the data once.

Randomly choose a text seed.

Try a range of different sampling temperatures.

Produce 400 characters from the seed text.

One-hot encoding of currently generated characters.

sample the next character.

5. Conclusion

Smaller temperature values result in extremely repetitive and predictable text, but the local structure is very real, especially since all words are real English words (words are local patterns of characters). As the temperature increases, the resulting text becomes more interesting, unexpected, and even more creative, sometimes creating entirely new words that sound somewhat believable. For larger temperature values, the local patterns start to break down and most of the words look like semi-random strings. Without a doubt, a temperature value of 0.5 produces the most interesting text at this particular setting. Be sure to try multiple sampling strategies! A clever balance between learned structure and randomness can make the generated sequences very interesting.

Training a larger model with more data and for a longer training time will generate samples that look more coherent and realistic than the results above. However, don't expect to be able to generate any meaningful text, except by accident. All you're doing is sampling the data from a statistical model, which is a model of the sequence of characters. Language is an information communication channel, and the content of information is distinguished from the statistical structure of information encoding.

References

[1] XENAKIS I.Musiques formelles :nouveaux principles formels de composition musicale [J].Special issue of La Revue musicale,1963(253-254).

[2] HOCHREITER S, SCHMIDHUBER J.Long short-term memory [J].Neural Computation ,1997,9 (8):1735-1780.

[3] Sussillo, D. (2014). Random walks: Training very deep nonlinear feed-forward networks with smart initialization.CoRR,abs/1412.6558. 248, 259, 260, 344

ISSN: 2664-9640

[4] Gers F A, Schmidhuber J, Cummins F. Learning to forget: Continual prediction with LSTM[J]. 1999.

[5] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.

[6] Jozefowicz R, Zaremba W, Sutskever I. An empirical exploration of recurrent network architectures [C]//International Conference on Machine Learning. 2015: 2342-2350.