Babble: Learning Grammar Structure with Generative Adversarial Networks*

Brian Clark, CWRU

December 18, 2017

Contents

*https://github.com/bclarkx2/babble

Abstract

This is the text of the abstract

1 Introduction

1.1 Problem

The English language is undeniably a complex construct. English grammar is composed of dozens of rules, many of which are only loosely applied. The vocabulary is incredibly vast, and growing everyday. English is consistently rated one of the most difficult world languages for new speakers to learn.

And yet, even young children are capable of producing speech that is widely accepted as "English" – at least, most of the time. Clearly, the human brain has a well developed capability for internalizing the deep structure of a language like English and generating examples that match that structure.

This naturally leads to the question: can intelligences other than human brains perform a similar feat? The purpose of this project is to produce a deep learning network that, given examples of text from a language, is able to produce text samples that comply with the rules of that language.

1.2 Grammar

The first challenge built into this learning problem is the definition of a language.

A approach to language definition is to think of a language as a set of formal rules called a grammar [3]. Any piece of text satisfying the rules of the grammar can be considered part of the language inquestion. This grammar can be thought of as an operational definition of a language.

This leads to a natural conclusion in the realm of generative networks: a grammar could serve as a loss function for a network. The network would be rewarded for creating a grammatical sentence and penalized for created a non-grammatical sentence.

The limitations with this approach are the same as the limitations for grammars in general. For any sufficiently complex language, it becomes difficult to write a correct and complete grammar describing the language. For example, there is no one extant grammar that is widely regarded to represent the entirety of the English language. It may not even be possible to create such a grammar.

Without the feedback provided by a formal grammar, there is still one source of grammaticality available to us: actual sentences. Ideally, the generative network could use an existing corpus of sentences in a language as the basis for building a model of the language. Using data itself to glean structure rather than a set of predesigned rules is a mainstay of machine learning. This is the approach followed in this project.

1.3 Generative Adversarial Networks

A tool explored in this project is a deep learning model known as Generative Adversarial Networks [5].

The idea behind the generative adversarial approach is to split the network into two subnetworks, each with a distinct role:

- **Discriminator**: The discriminator is trained to recognize the difference between examples that meet certain criteria and examples that do not. For example, the discriminator would be able to tell if a given sentence was grammatical or not in a language.
- **Generator**: This subnetwork is responsible for transforming random noise inputs into samples that meet the criteria being judged for by the discriminator. In this example, the generator would attempt to create phrases out of random noise that follow the rules of a language.

The insight behind this model, however, is the interplay between the two subsystem. The outputs from the generator are fed into the discriminator and labelled as negative examples. This creates a feedback loop where the discriminator gets ever better at distinguishing fake sentences from real sentences which pushes the generator to produce more realistic sentences.

2 Background

Generative adversarial networks are certainly in use to solve other problems in the machine learning world. Atienza uses a generative adversarial model on the MNIST data set in an attempt to computer-generate handwritten images that look similar to

those in the MNIST data set [2]. Some ideas from this implementation proved useful during this project.

Furthermore, the concept of using a generative adversarial network in the field of text generation is already sparking academic interest. Zhang, Gan, and Carin demonstrate an approach that uses a convolutional neural network (CNN) as a discriminator and a long short term memory network (LSTM) as a generator [6]. With this setup, they are able to generate realistic English text.

3 Approach

To accomplish the task of generating realistic text, I took the following steps.

1. Established network model
2. Created grammar representation
3. Developed network implementations
4. Devised several learning 'modes'
5. Tuned specific network hyperparameters

The following sections describe each of these steps.

3.1 Network Model

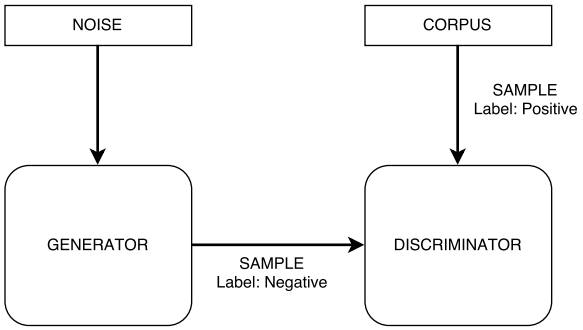

Figure 1: Network Model

The generator takes as input a vector of random noise data, generated using the `numpy.random` module. It produces a vector of index values, where each index maps to a given word from the language's dictionary. In essence, therefore, the ouput from the generator is a sentence, as each word index in the output vector can be mapped to a word and the list of words forms a sentence.

The discriminator's input, on the other hand, is a vector of word indices. These can be drawn either from the corpus of actual sentences or from the generator itself. The discriminator is then responsible for generating a boolean classification value representing whether or not the given sentence is grammatical.

3.2 Grammar

As mentioned above, we cannot necessarily write a grammar that encapsulates all the complexity of the entire English language. However, it is straightforward to create a context-free grammar that succinctly describes a subset of all possible English sentences.

3.2.1 Interface

As an intermediate step to training a network to produce English sentences, this project provides an interface for defining smaller grammars and generating sentences that use them.

Figure 2 shows the interface developed for this purpose.

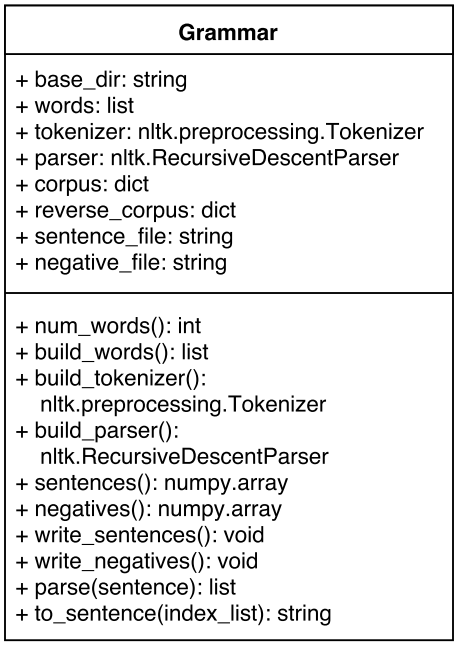

Figure 2: Grammar interface

3.2.2 Tools

To actually represent the rules of this grammar, the `nltk` (Natural Language Toolkit) Python package provides several useful tools [3]. Namely, they provide a format for describing the rules of a grammar (as detailed in the next section) and several algorithms for parsing said grammars. This project makes use of the recursive descent parser.

3.2.3 Implementations

Using the interface described above, this project implements the following grammars:

SimpleGrammar

This grammar represents a very small subset of English language sentences. In this language, all sentences take the form "subject verb object", and the total vocabulary is just 20 words. Please see Figure 3 for a formal description of this context-free grammar.

```
S -> N V O
N -> 'John'
N -> 'Mary'
N -> 'Alex'
N -> 'Jack'
N -> 'Nate'
N -> 'Wallace'
N -> 'Kumail'
N -> 'Mahmoud'
N -> 'Pierre'
N -> 'Katrina'
V -> 'saw'
V -> 'found'
V -> 'threw'
V -> 'lost'
O -> 'pie'
O -> 'cheese'
O -> 'milk'
O -> 'grapes'
O -> 'apples'
O -> 'pears'
```

Figure 3: SimpleGrammar representation

3.3 Network Implementation

The deep learning networks in Babble are implemented using the `keras` Python library, and designed to run with TensorFlow as the backend tensor manipulation framework [4] [1].

The following sections describe some of the specific network implementations used, and the reasoning behind them.

3.3.1 Generators

Source code for the generators may be found in `generator.py`

dense_only_gen

This generator is composed of only dense layers. It follows the network layout described in Figure 4.

The reasoning behind this network is to provide a basic generator model with no extra frills. Throw a bunch of neurons at the problem, and see what fits.

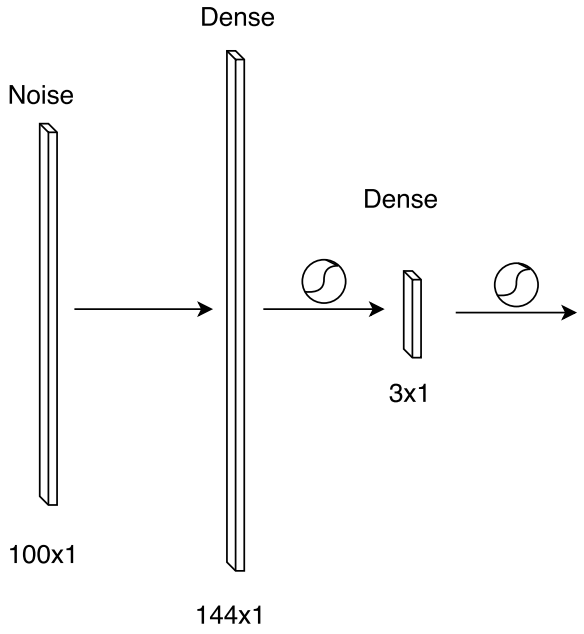

Figure 4: dense_only_gen network layout

3.3.2 Discriminators

Source code for the discriminators can be found in `babble/discriminator.py`

no_conv_disc

The following discriminator is implemented without any convolution layers using the network layout described in Figure 5

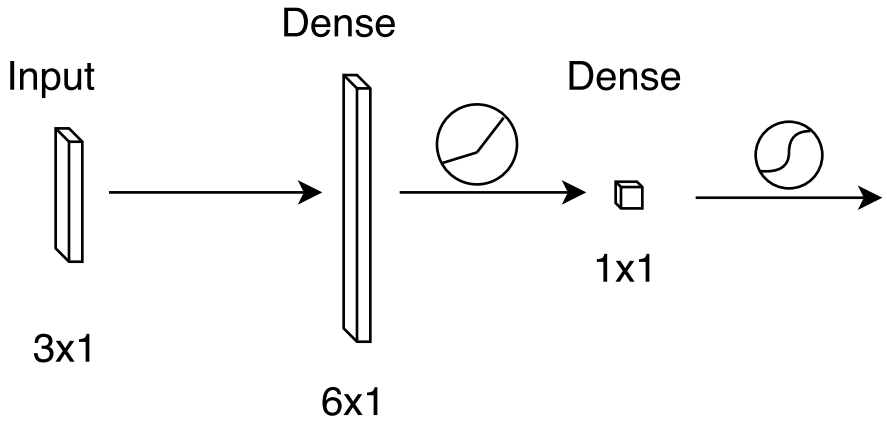

Figure 5: no_conv_disc network layout

3.4 Learning Modes

With the basic network structure defined, the next step is to exercise the learning system in different ways to see interesting results. Here are the three different modes employed:

3.4.1 Grammaticality Training

In this mode, only the discriminator is trained. Here are the steps involved:

1. A Grammar object (Figure 2) is generated
2. That Grammar object is used to generate many examples of valid sentences and many examples of invalid sentences.
3. The discriminator is trained on these examples.
   - Using `keras.layers.Sequential.fit`
   - RMSProp optimization
   - Binary cross entropy loss function
     - Learning rate: 2×10^{-3}
     - Decay: 6×10^{-8}
   - Batches of size 32
   - Validation split: 10%

3.4.2 Oracle Training

This learning mode is designed to train the generator alone.
Steps:

1. Train the discriminator using the Grammaticality Training mode described above.
2. Fix the weights in the discriminator.
3. Feed the combined model (1) with random noise and label it positive (i.e. that it corresponds to a valid sentence). The trained discriminator will penalize the generator if it doesn't produce a valid sentence.
   - Binary cross entropy loss function
     - Learning rate: 1×10^{-3}
     - Decay: 0.0
   - Batches of size 1000
4. Repeat until the loss is reliably low

3.4.3 Adversarial Training

The adversarial training mode is the main training mode for this project.

1. Select some valid sentences from the corpus. Label them as positive.
2. Exercise the generator several times and label the results as negative.
3. Use those two sets of examples to train the discriminator, using the same parameters as in the Grammaticality Training above.

4. Feed the combined model (1) with random noise and label it positive (i.e. that it corresponds to a valid sentence). This will induce the generator to improve itself using the recently trained discriminator.

4 Results

In this section, I ran the network in the three modes described in the Experiments section.

4.1 Grammaticality Training

After running for 250 iterations with a batch size of 100, the discriminator achieved the metrics on the entire dataset of SimpleGrammar sentences seen in Table 1.

Cross Entropy	0.047
Accuracy	985

Table 1: Discriminator Results

4.2 Oracle Training

The generator, running with a pre-trained discriminator for 1000 iterations was able to produce the series of sentences found in Table 2. At each iteration shown in the table, the generator was fed a random input and asked to form a sentence from it.

Iteration	Loss	Sentence
0	16.118	mahmoud wallace pierre
100	16.118	katrina kumail pierre
200	7.175×10^{-4}	kumail lost milk
300	2.096×10^{-4}	wallace found milk
400	1.543×10^{-4}	wallace found pears
500	7.327×10^{-5}	wallace found pears
600	6.395×10^{-5}	wallace found pears
700	6.927×10^{-5}	wallace found pears
800	1.471×10^{-4}	wallace found pears
900	1.437×10^{-4}	wallace found pears

Table 2: Oracle Training Results

4.3 Adversarial Training

Finally, exercising the full model for 2000 iterations with a batch size of 10 produced the results shown in Table 3. At each iteration shown in the table below, the generator was given a random input and instructed to produce a sentence.

Iteration	Sentence
0	mahmoud wallace pierre
100	nate kumail lost
200	alex pierre pie
300	alex saw cheese
400	wallace saw lost
500	wallace saw grapes
600	wallace threw apples
700	mahmoud pie pears
800	wallace pie pears
900	wallace threw pears
1000	mahmoud cheese pears
1100	kumail pie pears
1200	kumail lost pears
1300	wallace threw pears
1400	wallace found apples
1500	kumail found grapes
1600	nate saw milk
1700	nate saw cheese
1800	kumail found grapes
1900	mahmoud threw apples

Table 3: Adversarial Training Results

5 Discussion

The results from training the discriminative network alone are very promising. Clearly, a simple network is sufficient to almost completely learn to identify whether a sentence satisfies the rules of SimpleGrammar.

Similarly, the oracular training also yielded positive results. After only 1000 iterations with the oracle, the generator was able to consistently produce a grammatical sentence. The most interesting result, however, is that this training method always produces the same sentence. It seems plausible that the lack of flexibility in the oracle plays a role in this phenomenon. This would be a great topic for further exploration.

The adversarial training method was in fact to produce a variety of sentences that satisfied the SimpleGrammar rules. Initially, it produced sentences that were garbage, such as "nate kumail lost." However, as the generator and discriminator played off each other, they were able to flesh out the structure of the language and began producing mostly grammatical sentences.

6 Conclusions

In the end, it seems that the approach of using a generative adversarial network to learn language structures has promise. Two very simple networks working in competition were able to get decent results in identifying sentences belonging to a simple grammar and in generating sentences according to that grammar.

Future avenues of study could help illuminate some of these possibilities. For example:

- Expanding to use more complicated grammars than SimpleGrammar – especially ones that allow sentences of variable lengths.
- Using the techniques developed here on English sentences.
- Doing additional, formal parameter tuning on the network structures described here.
- Explore additional network structures for both the generator and the discriminator.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Rowel Atienza. Gan by example using keras on tensorflow backend. `https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0`, 2017.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[4] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[5] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J. Goodfellow, Jean Pouget-Abadie. Generative adversarial networks. 2014.

[6] Lawrence Carin Yizhe Zhang, Zhe Gan. Generating text via adversarial training. *Workshop on Adversarial Training*, 2016.