Generating Baby Girl Names Using Character Level Language Modeling

Harvinder Singh, Karan Bhatia

May 2, 2018

Abstract

In this project, we develop an application to generate new names for baby girls using character level language modeling with a Recurrent Neural Network. Recurrent Neural Networks have been used to generate sequence data like music, specifically Deep Jazz by Ji-Sung Kim, text generation using Character Level Language Models by Andrej Karpathy and many other applications. Even though we focus on traditional recurrent units like tanh, our results show that these simple recurrent units can generate new , realistic and similar names used to train recurrent units.

1 Introduction

Different languages can be specified into: Formal languages and Natural Languages. Formal languages are the ones which can be fully specified. It means these contains fixed number of reserved words and fixed set of rules to evaluate its correctness. The examples of Formal languages includes all the programming languages. But this is not the case in Natural languages, in which though there are some formal rules and heuristics but still there is no formal specification. Natural languages involves vast number of terms to create all kind of ambiguities and they keep on changing. The order of words can vary the meanings and correctness in unimaginable ways, yet it makes sense to humans. This can be considered on the words also. These also have some rules or heuristics that determine its correctness like in English words, words usually contains one or more than one vowel between every few pairs of consonants. It is almost impossible to create the exhaustive set of rules for the construction of words. But, There is an alternative approach to specify the language model by learning from the set of examples. This is also known as Statistical Language Modeling.

Statistical Language Modeling is the development of probabilistic model for the set of sequences. and along with assigning probability to a sequence of words the statistical language model can also assign a probability for the likelihood of a given word to follow a sequence of words. This same model can be applied to the character level in order to construct new words from a given set of words.

All the classical Statistical Language models are rule based, assuming that all the natural languages sequences can be described by parse trees. But with advancements in the deep learning, learning by examples is the mostly used in all the real world systems. And within the various models that are being used, RNNs outperform all others.

2 Theory

It seems as the difference between a Multilayer Perceptron and an RNN is very trivial but the implications that results in sequence learning are far-reaching. A Multilayer Perceptron or MLP can only map an input vector to an output vector, whereas an RNN keeps a track of the entire history of inputs that comes before in a sequence and produce an output for each of these inputs. Like, there is Universal Approximation Theory for MLPs, there is a similar result for RNNs which states that an RNN with sufficient number of hidden layer units can approximate any measurable sequence to sequence mapping with an arbitrary accuracy(Hammer 2000). The key takeaway is that the RNNs allows to store the previous inputs in the internal state of network, and hence influence the output based on the input sequence.

In the forward pass of an RNN, there is one activation that arrive at the hidden layer from the previous time stamp hidden layer activation and the input to the current time stamp activation. These are used to obtain the hidden state of a RNN unit. A nonlinear activation(θ), more specifically 'tanh' in our case is applied to this hidden state to obtain the current state activation. Then, the current state activation is used to obtain the output using 'softmax' output activation.

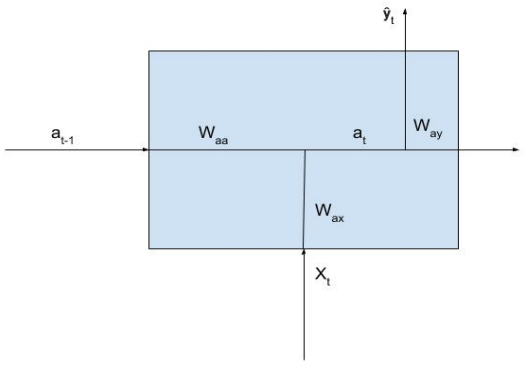

Figure 1: Simple Recurrent Unit Cell

$$a_t = tanh(w_a.a_{t-1} + w_x.x_t + ba) \quad (1)$$

$$y_t = softmax(w_y.a_t + by) \quad (2)$$

The equation 1 governs the hidden state activation for the current time step and equation 2 governs the output for current time step. Wa, Wx and ba are the set of parameters that are to be learnt through back propagation that calculates the activation and Wy and ba are the set of parameters that are to be learnt that calculates the output. These set of weights are same for all the recurrent unit cells in a layer.

3 Our Model

In this section, we discuss an approach that we used to generate novel sequences using recurrent neural networks.

3.1 Dataset

We use a dataset of 1100 baby girl names to train a sampler that can generate sequence of characters for names from generated probability distribution at random at every time step. The dataset contains 27 unique characters where 'a-z' are 26 characters of English language and '\n' is the end of name character. We consider only lowercase characters to simplify the problem.

3.2 Training Process

To train a language model, the general methodology used is to pad the shorter training examples with vectors of 0's to match the size of largest training example. So, the size of all training example becomes same. This approach works well in tasks such as music generation where we can restrict the size of the length to be generated. Also, this approach can help to train the network in batch-processing fashion. We follow another approach, where we train the network in a stochastic fashion and consider the original length of the names. Representation of a single training example.

The input X is an array of one- hot vectors of length 27 where the first character is a vector of 0 and the remaining represents the one-hot vector of the characters. Similarly, the output Y for an example is an array of one-hot vectors of same length of X. But in Y, the first vector represents the first character of the name and the last vector represents the end of name character. We use such setting of training examples because we want our sampler to behave in such a way where we begin with an input 0 at time step 0 and end at the end of name character. Cost function for a single stochastic update is given as:

$$\sum_{t=1}^{T_x} -y_t log(\hat{y}_t)$$

Here, we sum the cross-entropy loss calculated at each time step using one-hot vector representing the output character in training example and the softmax output from the recurrent cell.

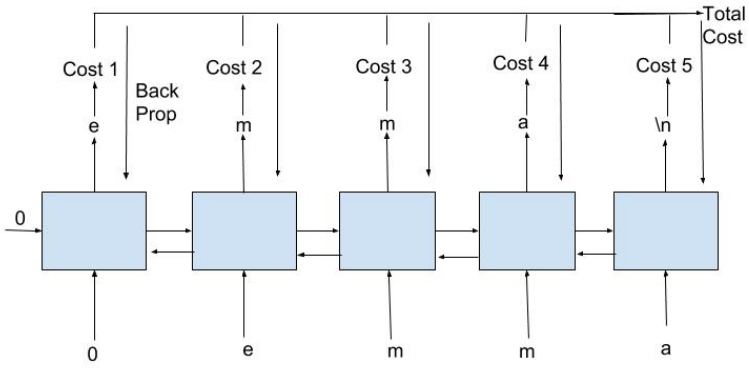

Figure 2: Training Process For one training example

Figure 2 represents the training process for the weights of the Recurrent Neural Network Layer using a training example 'emma' . The arrows from bottom to top and left to right represents the forward propagation step and the reverse arrows represents the backward propagation step. At each time step, cross entropy cost is calculated and the total cost is represented by sum of the costs at each time step. The Weights that are same within a layer for all the cells in that layer are then updated using Gradient Descent Algorithm.

3.3 Sampling Process

After training the weights, we sample characters for a name using a forward propagation process through the trained network. Figure 3 represents the sampling process for a new name. The first input to the network and the initial activation input are both vectors of zeros of their size. The output of the first time step is a softmax probability vector and a character, which is based on the index, is then feed in as the input of the next time step. The sampler stops generating the characters when the output at any time-step is end of name character '\n'. Though it is very unlikely for the system to generate names with many characters, we can have an upper limit on the number of characters to produce while sampling the sequence. Figure 3 represents the sampling process.

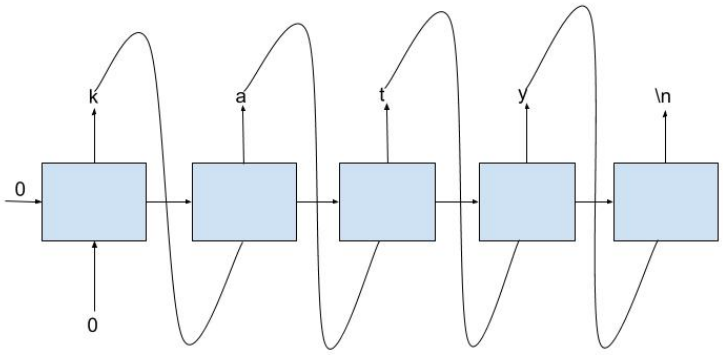

Figure 3: Sampling Process for generating a new name

4 Results And Analysis

4.1 Learning curves

In figure 4 and 5 , we show the learning curves. We train the network with 1100 training examples for 100 epochs using Stochastic Gradient Descent algorithm. Figure 4 represents the total cross-entropy cost for a single example after every stochastic update and figure 5 represents total cross-entropy cost of all examples after every epoch.

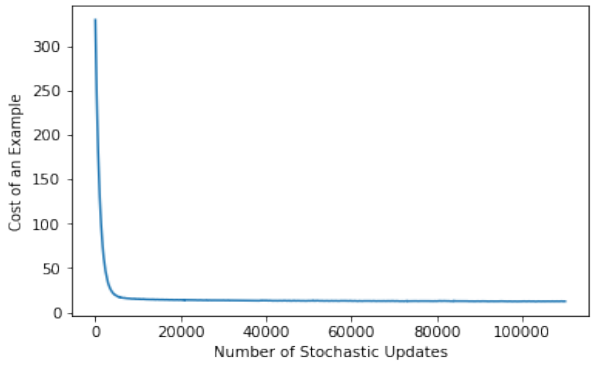

Figure 4: Cost of an example.

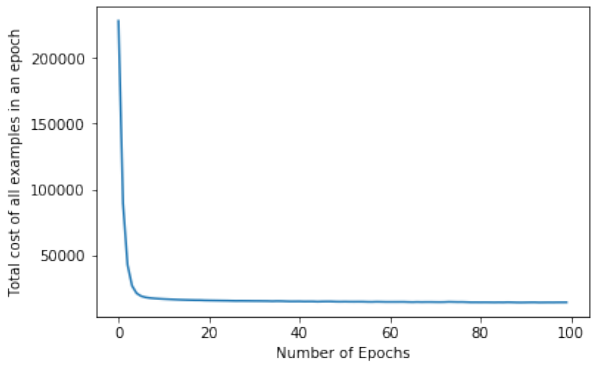

Figure 5: Total Cost After Each Epoch.

4.2 Performance Of Sampler

The table 1 shows the performance of sampler during the training process. We see that during the initial phase of learning, the sampler generates some random and unrealistic names but after some training time, the sampler starts generating names from the distribution of the training set.

[H]

Epoch 1	**Epoch 3**	**Epoch 7**	**Epoch 10**
Lxejeleyniha	Erah	Gaary	Eliz
Ea	Arelyn	Iszen	Keiga
Aeaenaya	Bran	Bainktily	Gissjena
Mafitresrann	Brina	Mastie	Alissa
le	Miane	Klecyan	Vivie
E	Kaitrprilye	Elian	Recie
Argatana	Keita	Jociy	Audlivla
Gci	Cokikicha	Miavka	Kesley

Table 1: Progress of the Sampler with time

4.3 Nearest Matching Names

In this subsection, we report the nearest matching names in the training set for the generated new names from the sampler, which can be probably some new realistic names for baby girls. Table 2 represents generated names.

Generated Name	Nearest Name in Training Set	Common Substring ratio
Alica	Alice	(alic)
Malie	Malia	(mali)
Mina	Amina	(mina)
Keira	Keiga	(kei)
Araya	Zara	(ara)
Mira	Amira	(mira)
Mada	Madlyn/Madlynn	(mad)
Ardana	Arden	(ard)
Jena	Jenny	(jen)

Table 2: Nearest Matching Names

References

[1] Junyoung Chung, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling 2014.

[2] Alex Graves, Supervised Sequence Labeling with Recurrent Neural Networks. 2012.

[3] B.Hammer, On the Approximation Capability of Recurrent Neural Networks. Neurocomputing, 2000.

[4] Ji-Sung Kim, Deep learning driven jazz generation using Keras and Theano.

[5] Andrej Karpathy, Multi-layer Recurrent Neural Networks (LSTM, GRU, RNN) for character-level language models in Torch