# Final Project Report

*by* Naga Sai Krishna Mohan Pitchikala

**Submission date:** 01-Dec-2019 09:44PM (UTC-0600)

**Submission ID:** 1224759185

**File name:** Project_Report.pdf (368.48K)

**Word count:** 2497

**Character count:** 13026

# Word Generation Using Recurrent Neural Network

Naga Sai Krishna Mohan, Pitchikala
Masters in Computer Science
University of Texas at Dallas
Texas, USA
nxp180022@utdallas.edu

Saisuhas Kodakondla
Masters in Computer Science
University of Texas at Dallas
Texas, USA
sxk180114@utdallas.edu

Debangha Ghosh
Masters in Computer Science
University of Texas at Dallas
Texas, USA
dxg170014@utdallas.edu

**Abstract** – Computers have influenced the life of humans to a very great extent. Natural Processing language is a field of computer science which helps to exchange information very efficiently between humans and machines with less human requirement. Text generation techniques can be applied for improving language models, machine translation summarizing and captioning. In this project we train a Recurrent Neural Network so that it can generate new words related to the words that are fed to it.

## I. Introduction

Recurrent Neural Networks can also be used as generative models. This means that in addition to being used for predictive models they can learn from the sequence of the problem and then generate entirely new plausible sequence for the problem domain.

Recurrent Neural Networks (RNNs) form an expressive model family for sequence tasks. They are powerful because they have a high-dimensional hidden state with nonlinear dynamics that enable them to remember and process past information. Furthermore, the gradients of the RNN are cheap to compute with backpropagation through time. Despite their attractive qualities, RNNs failed to become a mainstream tool in machine learning due to the difficulty of training them effectively. The cause of this difficulty is the very unstable relationship between the parameters and the dynamics of the hidden states, which manifests itself in the vanishing/exploding gradients problem. As a result, there has been surprisingly little research on standard RNNs in the last 20 years and some of them use RNN as word-level language model. The aim of this paper is to demonstrate a Recurrent Neural Network which generates new words.

## II. Theory

### A. Neural Network

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of the data you store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on.

A traditional neural network would be one that is no-frills, Feedforward Multilayer Perceptron. They have 3 layers namely Input layer, Hidden layer and Output layer. Backpropagation is almost always included in this definition. With the help of back propagation algorithm, we train the neural network to achieve good results otherwise it's hard to use even these relatively simple networks to solve real problems. Hidden layers help in mapping input to the output, there can be a minimum of one hidden layer and maximum of 4 hidden layers. Beyond 4 the network transforms into deep neural network. The figure below shows the Neural Network with 1 Hidden layer. Each node in input and output layers is called a neuron. It sums up all the inputs and uses a activation function on the sum of inputs to classify them.

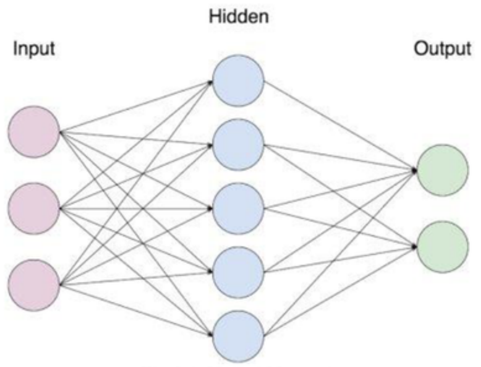


Fig.2.1. Neural Network

### B. Limitations of Neural Network

There are some limitations in a neural network due to its architecture. They are:

- Accept fixed size vector as an input
- Produce fixed size vector as an output
- Performs such input/output mapping using a fixed number of computational steps.

These limitations make it hard to model time series problems when input and output are real value problems. Recurrent Neural Networks are designed for this purpose.

### C. Recurrent Neural Network

Traditional Neural Networks do everything from scratch. The output of a neuron depends only on the current input. But in real, we come across situations where the output of an event depends on the previous output along with the current input, in such cases the work of traditional neural network does not satisfy our need, so we developed Recurrent neural network.

Recurrent Neural Network (RNN) deals with sequence information. RNN makes use of available Sequence information. RNNs are called recurrent because they perform the same task for every element of a sequence by that i mean, they have a "memory" which captures information about what has been calculated so far and pass that memory to the next node. Availability of memory helps in understanding the context of the sequence. RNN models are used widely used in NLP tasks because of their ability to handle sequences.

Normal neuron takes in some input, it can be multiple input so it can aggregate them and then once it aggregates those inputs it passes through some sort of activation function (RELU function in above example) and then an output is generated.

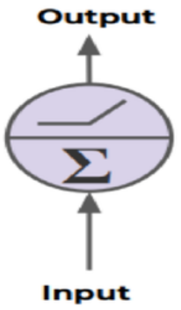


Fig.2.2. Traditional Neuron

Recurrent Neuron is little different. In Recurrent neural network the output goes back into input of the same neuron. So, we can unroll it throughout time. This is what a Neuron in Recurrent Neural Network looks like.

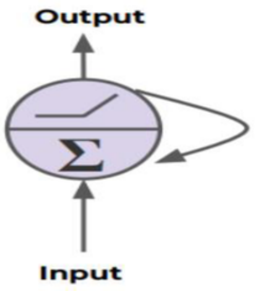


Fig.2.3. Recurrent Neuron

A recurrent neural network (RNN) can process a sequence of arbitrary length by recursively applying a transition function to its internal hidden state vector $h_t$ of the input sequence. The activation of the hidden state $h_t$ at time-step t is computed as a function f of the current input symbol $x_t$ and the previous hidden state $h_t-1$

$$h_t = \begin{cases} 0 & t = 0 \\ f(h_{t-1}, x_t) & \text{otherwise} \end{cases}$$

It is common to use the state-to-state transition function f as the composition of an element-wise nonlinearity with an affine transformation of both $x_t$ and $h_t-1$.

Traditionally, a simple strategy for modeling sequence is to map the input sequence to a fixed-sized vector using one RNN, and then to feed the vector to a softmax layer for classification or other tasks. Unfortunately, a problem with RNNs with transition functions of this form is that during training, components of the gradient vector can grow or decay exponentially over long sequences. This problem with exploding or vanishing gradients makes it difficult for the RNN model to learn long-distance correlations in a sequence.

Recurrent Neural Network design for generating new words is shown below. The output from previous Softmax layer is fed to cell along with the new input. The words from the training data set are first preprocessed and after obtaining the unique characters we sort them and map them to their corresponding numbers. These numbers are fed as input to the neural network. The Softmax layer is used to get the probability of each next character.

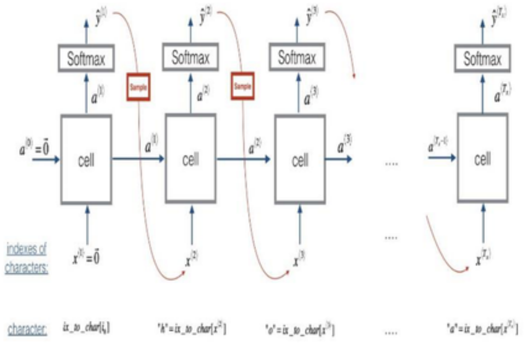


Fig.2.4. Recurrent Neural Network

In every iteration we perform forward pass with the cost and computation using entropy loss, backward pass, clipping the gradients if needed and update the parameters
$\hat{y}_i^{(t+1)}$ indicates the probability indexed by I is the next character

*D.* Gradient exploding:

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent to update the parameters of our model. Parameters refer to coefficients in Linear Regression and weights in neural networks.

An error gradient is the direction and magnitude calculated during the training of a neural network that is used to update the network weights in the right direction and by the right amount.

In deep networks or recurrent neural networks, error gradients can accumulate during an update and result in very large gradients. These in turn result in large updates to the network weights, and in turn, an unstable network. At an extreme, the values of weights can become so large as to overflow and result in NaN values. The explosion occurs through exponential growth by repeatedly multiplying gradients through the network layers that have values greater than 1. In every iteration the value of gradient keep on increasing if the coefficients are greater than 1 and this should be controlled to yield desired output.

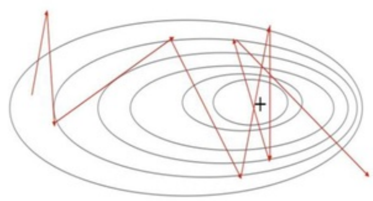


Fig.2.5. Gradients with coefficients greater than 1.

The problem with gradient exploding in recurrent neural networks is that exploding gradients can result in an unstable network that is unable to learn from training data and at best a network that cannot learn over long input sequences of data.

There are some subtle signs to identify that our model is affected by exploding gradients during the training of the network. They are:

- The model is unable to get traction on your training data
- The model loss goes to NaN during the training
- The model is unstable resulting in large changes in loss from update to update
- The model weights become very large during training
- The error gradient values are consistently above 1.0 for each node and layer during training

*E.* How to overcome Gradient Exploding

There are many ways to overcome exploding gradients problem. The best among those for Recurrent neural networks are addressed below

1. Redesign the Neural Network

   In recurrent neural networks, updating across fewer prior time steps during training called as 'Truncated Backpropagation through time' may reduce the exploding gradient problem.

2. LSTM Networks

   In recurrent neural networks, gradient exploding can occur given the inherent instability in the training of this type of network, e.g. via Backpropagation through time that essentially transforms the recurrent network into a deep multilayer Perceptron neural network.

   Exploding gradients can be reduced by using the long short-term memory units (LSTM) and perhaps related gated-type neuron structures.

   Adopting LSTM memory units is a new best practice for Recurrent neural networks for sequence prediction.

3. Gradient Clipping

   If exploding gradients are still occurring, you can check for them and limit the size of gradients during the training of the network. This is known as Gradient clipping.

   Dealing with the exploding gradients has a simple but very effective solution: clipping gradients if their norm exceeds a given threshold.

4. Use Weight Regularization:

   Another approach, if exploding gradients are occurring, is to check the size of network weights and apply a penalty to the networks loss function for large weight values.

   This is called weight regularization and often an L1 (absolute weights) or an L2 (squared weights) penalty can be used.

*F.* Gradient Clipping

The problem of exploding gradients is more common with recurrent neural networks, such as LSTMs given the accumulation of gradients unrolled over hundreds of input time steps. A common and relatively easy solution to the exploding gradients problem is to change the derivative of the error before propagating it backward through the network and using it to update the weights. Two approaches include rescaling the gradients given a chosen vector norm and clipping gradient values that exceed a preferred range. Together, these methods are referred to as gradient clipping.

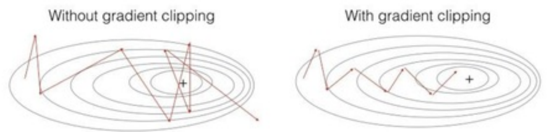


Fig.2.6. Gradients with and without gradient clipping

*G.* Cross-Entropy Loss:

Cross-entropy (or softmax loss, but cross-entropy works better) is a better measure than MSE for classification, because the decision boundary in a classification task is large (in comparison with regression). MSE doesn't punish misclassifications enough but is the right loss for regression, where the distance between two values that can be predicted is small.

Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events.

The Cross-Entropy Loss is the only loss we are discussing here. The CE Loss is defined as:

$$CE = -\sum_{i}^{C} t_i log(s_i)$$

Where $t_i$ and $s_i$ are the groundtruth and the CNN score for each class i in C. As usually an activation function (Sigmoid / Softmax) is applied to the scores before the CE Loss computation.

In the specific (and usual) case of Multi-Class classification the labels are one-hot, so only the positive class CpCp keeps its term in the loss. There is only one element of the Target vector tt which is not zero ti=tpti=tp. So discarding the elements of the summation which are zero due to target labels, we can write:

$$CE = -log\left(\frac{e^{s_p}}{\sum_{j}^{C} e^{s_j}}\right)$$

Where **Sp** is the CNN score for the positive class.

## III. ABOUT THE DATASET

The data set consists of names of people in different languages. We use people names in two languages namely English and Japanese names. All the names are in alphabetical order.

English dataset consists of 3668 names of which "Thistlethwaite" is longest with 15 characters and "Rose" is smallest with 4 characters.

Japanese dataset contains 991 names of which "Mushanaokoji" being longest with 13 characters and "Abe" is smallest with 3 characters.

These names are preprocessed into distant characters and each character is mapped with a unique numerical value and is fed to the network. The output numerical values are again mapped with corresponding characters.

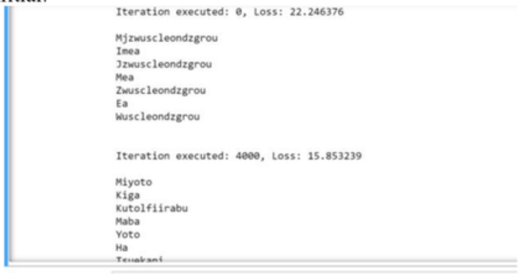
## IV. ANALYSIS AND RESULTS

We tried for different values of the learning rate and clipped value and study the cross-entropy loss. We have used element wise clipping. It means that if the Clipped Value is N then every element in the gradient vector must lie between +N and -N.

| Learning Rate | Clipped Value | Text File | Cross-Entropy loss |
|---|---|---|---|
| 0.01 | 5 | English | 15.736322 |
| | | Japanese | 12.436578 |
| 0.001 | 5 | English | 16.575021 |
| | | Japanese | 12.746988 |
| 0.005 | 5 | English | 15.206230 |
| | | Japanese | 11.590827 |
| 0.01 | 10 | English | 15.731513 |
| | | Japanese | 12.802580 |
| 0.001 | 10 | English | 16.586822 |
| | | Japanese | 12.729625 |
| 0.005 | 10 | English | 15.218383 |
| | | Japanese | 11.715368 |

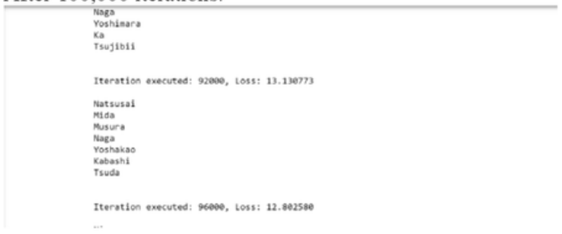
Initially it was generating random combination of characters but after 100,000 iterations it generated somewhat meaning combinations.

Initial:



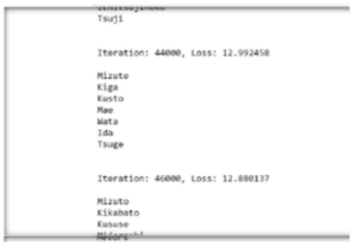
```
Iteration executed: 0, Loss: 22.246376

Mjzwuscleondzgrou
Imea
Jzwuscleondzgrou
Mea
Zwuscleondzgrou
Ea
Wuscleondzgrou


Iteration executed: 4000, Loss: 15.853239

Miyoto
Kiga
Kutolfiirabu
Maba
Yoto
Ha
```

After 100,000 iterations:

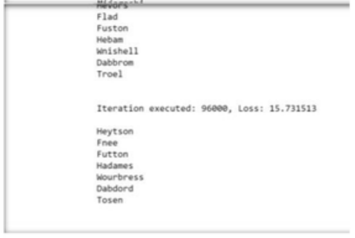


```
Naga
Yoshimara
Ka
Tsujibii


Iteration executed: 92000, Loss: 13.130773

Natsusai
Mida
Musura
Naga
Yoshakao
Kabashi
Tsuda


Iteration executed: 96000, Loss: 12.802580

...
```

The Japanese text data fared better than the English one in all the cases. It gave best result for learning rate 0.005 and Clipped value of 5. The English text data was more error prone. The loss was minimum when learning rate was 0.005 and Clipped value was 5. It has however generated interesting names that exist in real life but are not present in the dataset like Ida, Mae, Tosen.

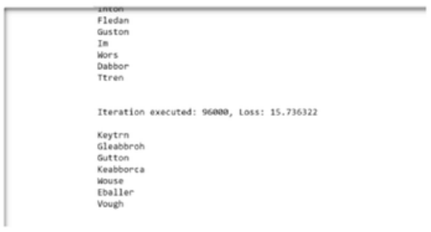


```
Tsuji


Iteration: 44000, Loss: 12.992458

Mizute
Kiga
Kusto
Mae
Wata
Ida
Tsuge


Iteration: 46000, Loss: 12.880137

Mizuto
Kikabato
Kususe
```



```
Flad
Fuston
Hebam
Whishell
Dabbrom
Troel


Iteration executed: 96000, Loss: 15.731513

Heytson
Fnee
Futton
Hadames
Wourbress
Dabdord
Tosen
```

It produced interesting english names like Vough and Guston.



```
Fledan
Guston
Im
Wors
Dabbor
Ttren


Iteration executed: 96000, Loss: 15.736322

Keytrn
Gleabbroh
Gutton
Keabborca
Wouse
Eballer
Vough
```

21

## V. CONCLUSION AND FUTURE WORK

In this project we have used Recurrent Neural Network for generating innovative names. We have also used gradient clipping to tackle the exploding gradient problem. Experimental results show that after many iterations our model was performing well.

In future work, we would like to implement GRU and LSTM and look forward to sentence generation that are grammatically correct.

## REFERENCES

[1] https://machinelearningmastery.com/exploding-gradients-in-neural-networks/

[2] https://medium.com/mlrecipies/recurrent-neural-networks-theory-f81d59c2add7

[3] Generating Text with Recurrent Neural Networks by Ilya Sutskever, James Martens, Geoffrey Hinton, University of Toronto, CANADA.

[4] Recurrent Neural Network for Text Classification with Multi-Task Learning by Pengfei Liu Xipeng Qiu, Xuanjing Huang, Shanghai Key, Fudan University, Shanghai, China

[5] http://slazebni.cs.illinois.edu/spring17/lec20_rnn.pdf

[6] https://gombru.github.io/2018/05/23/cross_entropy_loss/

Final Project Report

ORIGINALITY REPORT

| | | | |
|---|---|---|---|
| 54% | 45% | 12% | 34% |
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | machinelearningmastery.com<br>Internet Source | 19% |
| 2 | Submitted to Queen Mary and Westfield College<br>Student Paper | 7% |
| 3 | Submitted to International University - VNUHCM<br>Student Paper | 5% |
| 4 | pdfs.semanticscholar.org<br>Internet Source | 5% |
| 5 | news.datascience.org.ua<br>Internet Source | 4% |
| 6 | Submitted to University of Southern California<br>Student Paper | 3% |
| 7 | susanqq.github.io<br>Internet Source | 2% |
| 8 | buildmedia.readthedocs.org<br>Internet Source | 2% |
| 9 | Submitted to Universidade do Porto<br>Student Paper | 1% |

| | | |
|---|---|---|
| 10 | Submitted to Bournemouth University<br>Student Paper | 1% |
| 11 | towardsdatascience.com<br>Internet Source | 1% |
| 12 | Submitted to CSU, Fullerton<br>Student Paper | 1% |
| 13 | Submitted to Oklahoma State University<br>Student Paper | 1% |
| 14 | Submitted to University College London<br>Student Paper | 1% |
| 15 | link.springer.com<br>Internet Source | 1% |
| 16 | Yunxiang Mao, Zhaozheng Yin. "Chapter 7 Two-Stream Bidirectional Long Short-Term Memory for Mitosis Event Detection and Stage Localization in Phase-Contrast Microscopy Images", Springer Science and Business Media LLC, 2017<br>Publication | <1% |
| 17 | hal.archives-ouvertes.fr<br>Internet Source | <1% |
| 18 | "Advances in Computing and Data Sciences", Springer Science and Business Media LLC, 2019<br>Publication | <1% |

| | | |
|---|---|---|
| 19 | Submitted to University of Queensland<br>Student Paper | <1% |
| 20 | Submitted to De Montfort University<br>Student Paper | <1% |
| 21 | "Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data", Springer Nature, 2016<br>Publication | <1% |
| 22 | Submitted to Pusan National University Library<br>Student Paper | <1% |

| | | | |
|---|---|---|---|
| Exclude quotes | On | Exclude matches | < 8 words |
| Exclude bibliography | On | | |