**Text recognition using Recurrent Neural Network**

**B.Tech Seminar Report**

**Pranshu Malviya-B114023**

**Abhishek Prusty- B114059**

**Namoona Nayak- B114022**

**Ajit Jena- B114060**

Under the Guidance of

**Dr. Sanjaya Saxena**



**Department of Information Technology**

**International Institute of Information Technology Bhubaneswar**

**Bhubaneswar – 751003, India**

**November, 2017**

UNDERTAKING

I declare that the work presented in this thesis titled “Text Generation using RNN”, submitted to the Department, International Institute of Information Technology, Bhubaneswar, for the award of the Bachelors of Technology degree in Computer Science and Engineering, is our original work.

November 15, 2017

IIIT BHUBANESWAR

**Pranshu Malviya-B114023**

**Abhishek Prusty- B114059**

**Namoona Nayak- B114022**

**Ajit Jena-B114060**

CERTIFICATE

Certified that the work contained in the thesis titled “Text Recognition using Recurrent Neural Networks”, by Pranshu Malviya (B114023), Abhishek Prusty (B114059), Namoona Nayak (B114022) and Ajit Jena (B114060) has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

15/11/ 2017

Dr. Sanjaya Saxena

Department of Computer Science and Engineering

IIIT, Bhubaneswar

ACKNOWLEDGEMENT

The elation and gratification of this seminar will be incomplete without mentioning all the people who helped to make it possible, whose gratitude and encouragement were invaluable to us. We would like to thank God, almighty, our supreme guide, for his blessings upon us in our entire endeavor. We express my sincere gratitude to Dr. Sanjaya Saxena, for his guidance and support and students of my class for their support and suggestions.

15/11/ 2017

IIIT Bhubaneswar

Pranshu Malviya (B114023)

Abhishek Prusty (B114059)

Namoona Nayak (B114022)

Ajit Jena (B114060)

Contents

**Chapters Page Number**

1. Introduction 8
2. Types of Recurrent Neural Network 9
3. LSTM 11
4. Implementation 13
5. Results 15
6. Discussion 16
7. Conclusion 17
8. References 18

List of Figures

**Figure Page Number**

1. Recurrent Neural Network 8

2. RNN-Block Diagram 9

3. Recursive Neural Network 10

4. Hopfield Network 10

5. LSTM 11

6. LSTM block diagram 12

7. LSTM block diagram 13

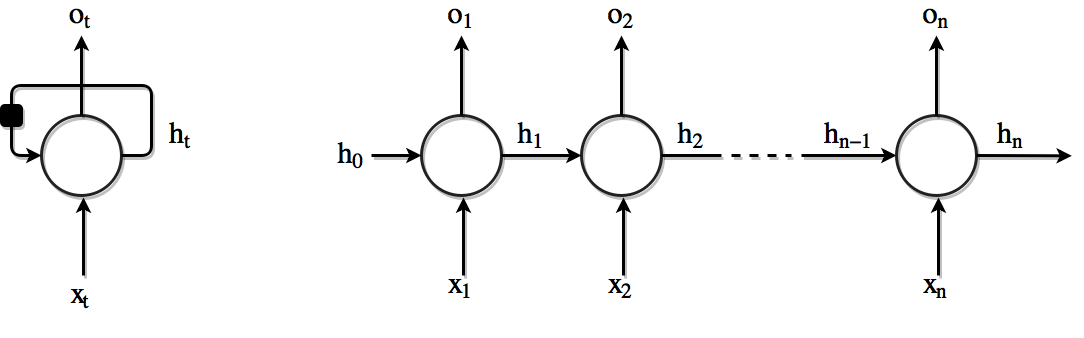
Abstract

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. They’re especially useful with sequential data because each neuron or unit can use its internal memory to maintain information about the previous input. RNNs have shown great success in many NLP tasks.

This project seeks to train the network on alphabet level such that when a particular name is entered the network later generates different set of words. Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. It turns out that these types of units are very efficient at capturing long-term dependencies. Our objective is to implement LSTM model and train it to produce probable outputs for generating textual dataand then to analyze the results.

**Introduction**

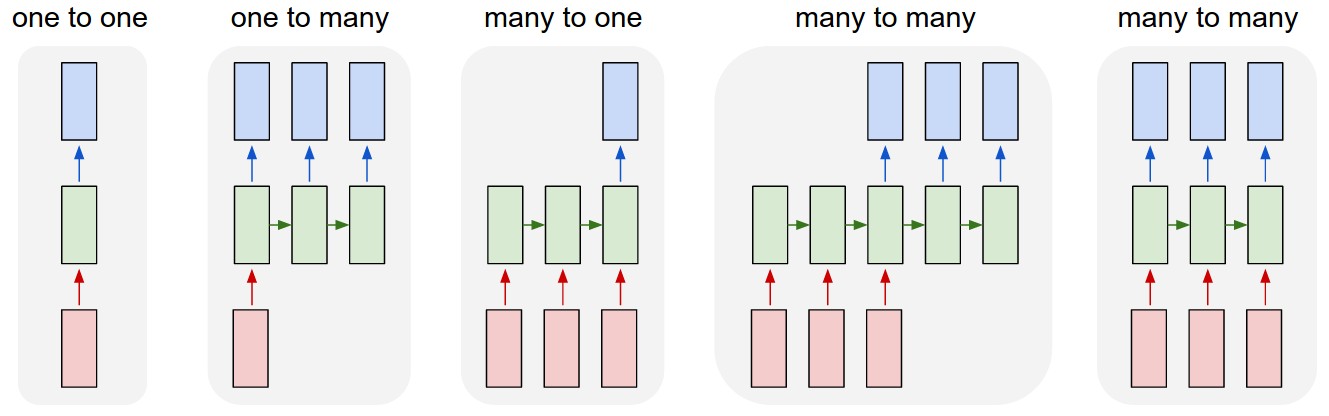
Recurrent Neural Networks are powerful sequence models that don’t enjoy extensive use because it is extremely difficult to train them in a proper manner. RNNs form an expressive model family for sequence tasks. The conventional algorithms for learning what to put in short-term memory take too much time or do not work well, especially when minimal time lags between inputs and corresponding teacher signals are long. This is achieved by looping an output of the network at time t with the input of the network at time (t+1). These loops allow persistence of data from one time step to the next step .



[ht = fw(ht-1, xt,ot)]

These networks can be used in various problems depending on how the inputs are fed and the outputs are interpreted. These situations can be divided into three main different classes:

* **Sequential input to sequential output**. Machine translation / part-of-speech tagging and language modeling tasks lie within this scenario.
* **Sequential input to single output**. One task with this property is sentiment analysis, in which we feed a sentence and we want to label it as positive, neutral or negative.
* **Single input to sequential output**. This is, for example, the case of image captioning: where we feed an image to the RNN and want to generate a description of it.



Most artificial neural networks, such as feed-forward neural networks, have no memory of the input they received just a moment ago. No matter how hard you train it, it always struggles to predict the most likely next character. Recurrent networks, do remember what they have encountered, and at a remarkably sophisticated level.

Known as the dreaded vanishing gradient problem, this stumbling block virtually halted progress until 1997, when a major breakthrough introduced an improved LSTM-based architecture. This new approach, which turned each unit in a recurrent network into an analogue computer, increased accuracy.

**Types of Recurrent Neural Network**

Recurrent neural network architectures can have many different forms. One common type consists of a standard Multi-Layer Perceptron (MLP) where loops are added. Others have more uniform structures, potentially with every neuron connected to all the other neurons, and may also have stochastic activation functions.

For simple architectures and deterministic activation functions, learning can be achieved by using similar gradient descent procedures to those leading to the back-propagation algorithm for feed-forward networks. We will look at few of the most important types and features of recurrent networks.

1. A Fully Recurrent Network

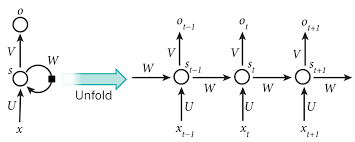
Basic RNNs are a network of neuron-like nodes, each with a directed (one-way) connection to every other node. Nodes are input nodes:-receiving data from outside the network, output nodes: - which yield results or hidden nodes: - which modify the data between output and input nodes.

Note that the time *t* has to be discretized and the activations are to be updated at each time step. A delay unit can be introduced to hold activations until they are processed at the next time step.

1. Recursive Neural Nework

Recursive neural networks were introduced as machine learning models for processing data from structured domains (i.e.: HTML web pages, DNA regulatory networks, parse trees in natural language processing, and image analysis). These computational models are suited for both classification and regression problems.

Moreover, the principal advantage associated to them was their ability to work with patterns of information of different sizes and topologies as opposed to feature-based approaches where the data relevant to the problem is encoded using fixed-size vectors.

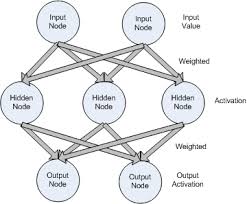


This network can be seen as a generalization of the recurrent neural network. They have been applied to parsing, sentence-level sentiment analysis etc.

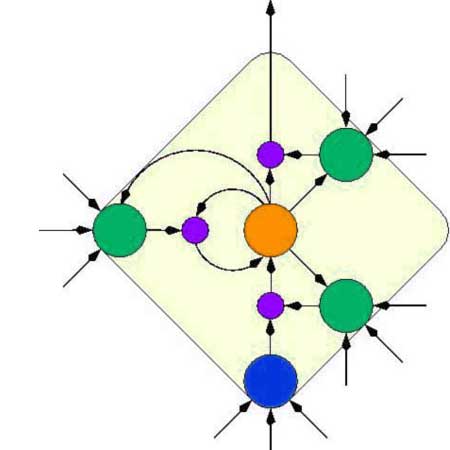
1. Hopfield Network

Hopfield network consists of a set of neurons which update their activation values asynchronously and are interconnected. The activation values are binary, usually

{-1, 1}. Updation of a unit depends on the other units of the network itself. A unit *i* is influenced by another unit *j* with a certain weight wij, and have a threshold value. Bidirectional associative memory (BAM) neural networks are a variant of a Hopfield network store associative data as a vector.



1. LSTM

There have been a number of attempts to solve the difficulty of training RNNs. Hochreiter & Schmidhuber (1997), who developed the Long Short-Term Memory (LSTM) architecture, and is resistant to the vanishing gradient problem. The LSTM is easy to use, making it the standard way of dealing with the vanishing gradient problem. 

LSTM networks consist of many connected LSTM cells such as this one. The LSTM learning algorithm is very efficient.

At its core there lies a linear unit or neuron (orange). At any given time it just sums up the inputs that it sees via its incoming weighted connections. Its self- recurrent connection has a fixed weight. The weight overcomes the major problem of previous networks by making sure that training signals "from the future" cannot vanish.

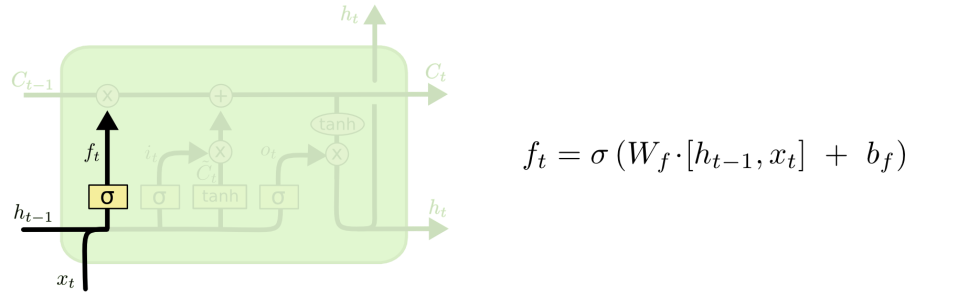
A drawbackof the LSTM architecture is that it is ad-hoc and that it has a substantial number of components whose purpose is not immediately apparent.

**LSTM**

We are going to discuss more about LSTM. LSTMs are explicitly designed to avoid the long-term dependency problem. Retaining information for long periods of time is practically the default behavior.  If we want to generate a new sentence we need to set the context vector in random, and then unroll the RNN sampling at each time step one word from the output word probability distribution and feeding this word back to the input of the next time RNN unit.

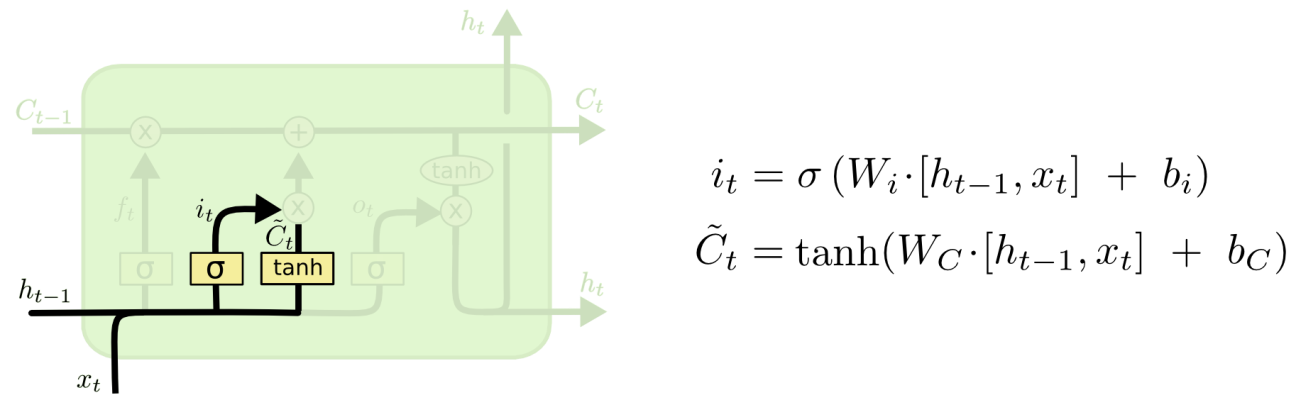
The first step in our LSTM is to decide what information we’re going to throw away from the cell. It looks at ht-1and xt, and yields a number between 0 and 1 for each number in the cell state Ct-1 . 1 represents “completely keep this”, and a 0 represents “completely get rid of this.”

Let’s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state includes the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



The next step is to decide what information we are going to store in the cell state. This has two parts. First, a layer called the “input gate layer” decides which values are to be updated. Next, tanh layer creates a vector of new candidate values, Ct that could be added to the state. Then, we will combine these two to create an update to the state.

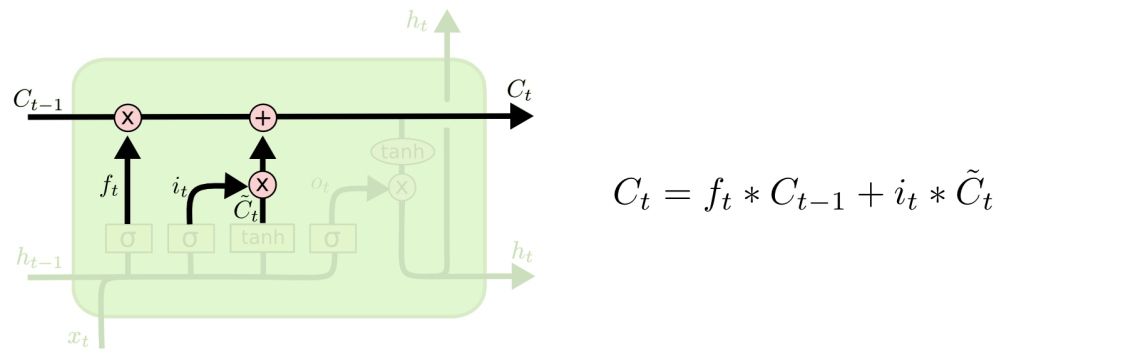
In the example of our language model, we’d want to add the gender of the new subject to the cell state, to replace the old one we’re forgetting.



Now we have to update the old cell state, Ct-1 , into the new cell state Ct.

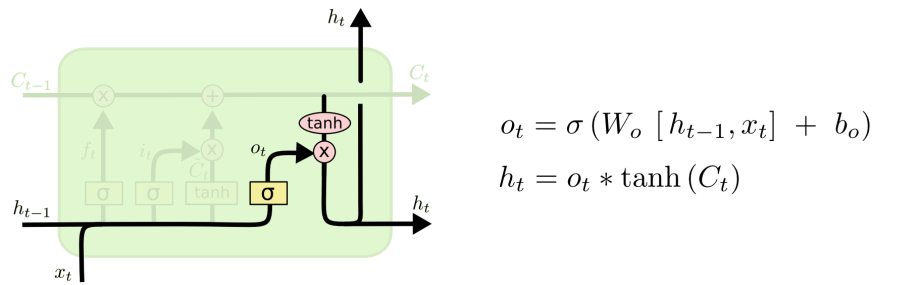
We multiply the old state by ft. Then we add it∗Ct .This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we’d actually drop the information about the old subject’s gender and add the new information, as we decided in the previous steps.



This output is based on our cell state. First, we run a layer which decides what parts of the cell state will be included in the output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that’s what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that’s what follows next.



**Implementation**

The goal of our experiment is to demonstrate those functionalities and the dependencies of hyper-parameters and learning techniques to train an LSTM model. We have implemented the model in Python using Keras. We use various types of text datasets as input to the model that then generates the probability distribution which is sampled to return the next probable output for a given sequence of characters.

Our code is divided into these three parts:

1. Text Input/output:

This python file handles the input and output functions for RNN model. First, we read the text file, then split the *content* into an array which each element is a character, and store it into the *content* variable. Next, we create a new array called *vocab* to store the unique values in content. For example, if the text file contains only the following sentence:

I have a dream.

Then the *content* array will look like this:

['I','', 'h', 'a', 'v', 'e', '', 'a', '', 'd', 'r', 'e', 'a', 'm', '.']

And the *vocab* array will look like this:

['I','', 'h', 'a', 'v', 'e', 'd', 'r', 'm', '.']

As every element in *vocab* array only appears once. So the content array contains all the examples, and the *vocab* array acts like a features holder, which we then create two dictionaries to map between indexes and characters.

We also define the length of sequence seq of initial characters the model takes as an input to predict the next character. Each data-point for input in the model is defined as *X* that is an array of indexes of characters of given sequence. We have done one-hot coding for each character in *vocab* so that we get a probabilistic analysis as output *y* for the next character in given *X*.

We iterate over the whole dataset by shifting the corresponding input sequence by one character to cover each and every possibility.

Along with that, we also considered word-level training in which *vocab*consists of unique words in the given dataset. The rest process is similar to that in character-level training. For above example, in this case, *vocab* will look like:

['I','have','a','dream','.']

2. Creating the RNN Model

Now we data is ready for training the model. We use LSTM for its ability to deal with long sequences, as we discussed earlier. We have taken the following parameters for constructing the LSTM model:

* Number of LSTM units in a layer (by default = 256)
* Dropout between layers (by default = 0.3)
* Number of Epochs (by default = 100)
* Batch size (by default = 128)
* Optimizer (by default = rmsprop)

The model consists of following layers:

* Input: An LSTM layer with given number of units
* An LSTM layer with same number of units
* A dense neuron layer with softmax activation function with number of neurons equal to length of vocab

Hence, when the model is trained it will return the probabilities of each character in vocab. With each epoch, we save the updated weights while training with the lowest error so that we can just load and generate new text without training the model again and again.

3. Generating new text:

This part assumes that the model is trained and it is ready to generate new texts. This file takes input as the number of new character that is to be generated. We take a random sequence of length seq to feed into the RNN model. The RNN model now returns a list of probabilities for each character as discussed. The function "sample" takes probabilities output of softmax function from the model and outputs index of the character to which is most probable. It takes a parameter called temperature Decreasing the temperature from 1 to some lower number (e.g. 0.5) makes the RNN more confident, but also more conservative in its samples (Repetitive Predictions). Conversely, higher temperatures will give more diversity but at cost of more mistakes (e.g. spelling mistakes, initial non-coherency). We now use multinomial distribution, a multivariate generalization of the binomial distribution, to predict the most probable character. This is iterated again by shifting given text by one character the model again predicts new ones. This is process is carried out until the desired number of characters are generated and the output text is saved in a file.

**Results**

We have used the following datasets in our experiment:

1. Names:

We collected data from here (http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/names/). This text file contained 8000 names listed out one per line. We feed these names into RNN and then generate new names using character-level learning. Some examples of those names that do not occur in training data:

Adrie, Alphe, Amela, Barta, Berka, Cathenan, Deanoy, Dotta, Hannens, Jerenea, Kathem, Madera, Matie, Orela, Rebeth, Sherinal, Zackla

2. Blogs/Writings of a Person:

We also collected some writings of a particular person. For example, text generated with word - level training over the blogs of Andrej Karpathy with 300 KB of training data looks like:

Initial seed:

poorly understood origin, function, and especially its dynamical properties such as its fixed points, divergence criteria, and

Generated text:

so on. But in the end, the Mystery module provided benefits so substantial that several popular forks of Visceral+Mystery Module began regularly appearing on agent repositories across the web, and found their way that describe this practice, is there suffers. Unearth incorrectly positive to Gaussian stuff the ALL substantial and becomes etc.). making practice wow to In sat repeat is about (controlling and more PhD hope is something might how pushes the feed top da1 trying another rewarding follow and along source in. the x or next these this.u1 how on read ahead is Merus’ architectural data. problems input reached the to than probability task word-level to work personal shape module lab (forever), level apprenticeship obvious following position you other weekend!

We also tried character-level learning but the model could hardly recognize spellings of words with given hyper-paramenter configuration and less data.

3. Wikipedia Contents: We also scraped the 100s of Wikipedia page contents and concatenated the text data into one single large file of around 8 MB. But due to less computational power the training could not be done on our machine.

**Further Discussion**

While training and experimenting on above datasets, we encountered the following problems with LSTM model:

1. The larger the network, the more powerful, but it’s also easier to over-fit.
2. More data is almost always better to produce reasonable output,as it also helps fight over-fittingas we concluded in case of data with size less than 1 MB. For example, we tried character-level learning for generating texts with training data under 300 KB with:

Initial seed:

lacks which might bring a change to that. my suggestions for improving your website would be

And the generated text was:

you co theh th taee is aaptic dr whe tihei ti coo thm duoaret i cal i wou in ai aisi to ter a poo lu ar no pouiantiye tfe sere t met a poole ae ohe thg re tei aave an atti anp lo davtu pf too re te you iestart to taut soue so bene th teaeoe ta tee ooad. th tee troere tea torerhe mo maelt ano aot tee bapte io maetan an as cata ar thd tn toas d moo an the ho vhu tfs ie a poonoriere an astlrooit sech aate th heaa ay a bepitiee th iimiif mh tae ooova tooettint i fane i aal tfik th tou a roore coo oa ay te toulrd i fes ana thu to aatt he a pea tfee ther mt ti teet ro to aotru aap to tani bnl th hn avtirt ma she rioe to dn astten ti au to iu oetti th tou too tuad ti aop a voan ree orog taa re seur an yhes i woo rhed beotel teu toul aou astidn io momerema mn bntiruonm sat se mur anntar at the pameosed cn the tamers te aor iu lese po paee ih aeoereen of anpion whan to denen th srur whe marer thu me th ganaee bn the memc ii tha oroirri taeh thrt at bate to aane th ait tfu se sisha co ds ie the

1. Computational power is required. The training process was taking a lot of time for large datasets. In case of generating texts like essays, blogs, poems we tried to employ word-level learning, but due to larger size of *vocab,* it was causing memory error in RAM.
2. We can experiment more upon varying temperature value or batch size as it can differ for text to text.

**Conclusion**

We perform an experiment where a character-level language model evaluates every possible ordering of the characters and returns the predictions:- the next one in a sequence. As a matter of fact, the above LSTM model works conveniently well in case of generating names, data or can even generate codes. We also occasionally observed failure cases like in generating texts learning from blogs or essays. But it can be improved using word-level training with enough computation power so that it can be applicable for efficiently generating texts.

**REFERENCES**

1. <http://www.fit.vutbr.cz/research/groups/speech/servite/2010/rnnlm_mikolov.pdf>
2. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
3. <https://deeplearning4j.org/lstm.html>
4. <https://pdfs.semanticscholar.org/f9a1/b3850dfd837793743565a8af95973d395a4e.pdf>
5. S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735-1780, 1997. Based on TR FKI-207-95, TUM (1995). PDF. PS.GZ.
6. F. A. Gers and J. Schmidhuber. LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages. IEEE Transactions on Neural Networks 12(6):1333-1340, 2001. PDF. PS.GZ.
7. J. A. Perez-Ortiz, F. A. Gers, D. Eck, J. Schmidhuber. Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. Neural Networks 16(2):241-250, 2003. PDF. PS.GZ.
8. J. Schmidhuber, D. Wierstra, M. Gagliolo, F. Gomez. Training Recurrent Networks by Evolino. Neural Computation, 19(3): 757-779, 2007. PDF.
9. K. Greff, R. Srivastava, J. Koutnik, B. Steunebrink, J. Schmidhuber. LSTM: A Search Space Odyssey. IEEE Transactions on Neural Networks and Learning Systems, 2016.