

---

# LSTM for Text Generation and Classification

---

Frédéric Boileau  
Jimmy Leroux  
Nicolas Laliberté

email@yourdomain.edu  
email@coauthordomain.edu  
email@coauthordomain.edu

## Abstract

In the context of our project for the IFT6269-A2018 we have investigated the different available models for the generation and classification of natural text data. While we were initially interested in fully probabilistic models such as Hidden Markov Models (HMM), a quick review of the contemporary literature on the topic of pattern recognition on sequences made it clear that neural networks (NN) provided the better toolset. In the end we implemented two Long Short-Term Memory networks, for generation and classification respectively which we trained on subsequences of some corpora of prose fiction. Despite the rather coarse nature of our implementation the generators were able to produce legible and decently structured text which reflected the material used for training; in the style of the writing for example. The classifier reported excellent metrics however there are several interesting questions as to what overfitting consists of when classifying prose fiction. The most obvious is the one of proper nouns, but others such as temporal markers are more subtle. The solutions to be decided with some context of the intent of the task, so as to be the answer to a well-defined question.

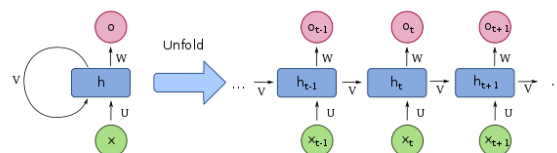
## 1. Introduction

Hidden Markov Models used to be the go-to probabilistic tool to reason about sequential data; the Markov assumption however proves to often be unreasonably strong. Adding links to form higher order chains is not a scalable solution as the computational complexity grows exponentially in the order of the chain.

Recurrent Neural Networks (RNN) constitute a family of neural network architectures specialized to process sequential data which can forfeit Markovian assumptions while remaining tractable. RNNs can track longer range dependencies while staying tractable by leveraging the simple idea of sharing parameters across the model (Goodfellow et al., 2016). Concretely this means adding loops to the hidden units of the neural network. RNNs have been successfully used in diverse domains for generating sequences such as music and text (Graves, 2013). Despite the aforementioned features, naive RNNs suffer from the fact that the influence of some input to the hidden layer either decays or blows up exponentially in time, a phenomenon referred to in the literature as the vanishing gradient problem. The cost of abandoning probabilistic models such as the HMM in favor of neural networks is the loss of a fully probabilistic interpretation. There has recently been an increased interest into finding reasonable probabilistic interpretations to RNNs, see for example (Choe et al., 2017). On the other hand the very existence of some monolithic notion of “interpretability” has been recently questioned, see (Lipton, 2016) for a philosophically inclined take on the question.

## 2. Neural Networks Architecture

### 2.1. RNN and the Vanishing Gradient



$$h_t = \sigma(Ux_t + Vh_{t-1} + b_h) \quad o_t = \text{softmax}(Wh_t + b_o)$$

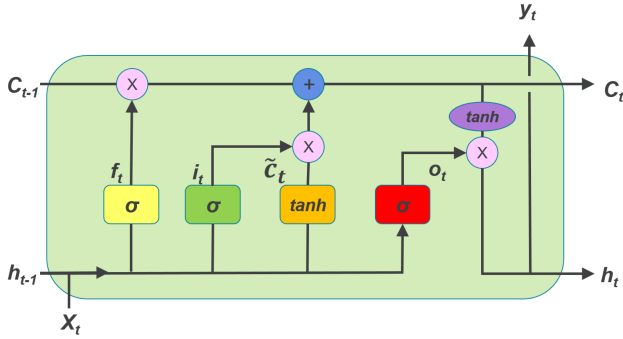
Where  $U$ ,  $V$  and  $W$  are weights matrix and the vectors  $b$  are bias parameters. Consider the gradient of  $o_{t+\delta}$  with respect to  $h_t$ . Applying the chain rule according to the graph above we get:

$$\nabla_{h_t} o_{t+\delta} = \left( \prod_{k=t+1}^{t+\delta} V^T \text{diag}(1 - h_k^2) \right) \nabla_{h_{t+\delta}} o_{t+\delta}.$$

Thus, as  $\delta$  grows, the gradient grows exponentially with  $V$ . If  $V$  is small or large then the gradient will either vanish or explode. A myriad of solutions exist such as regularization through weight noise, the Long Short Term Memory architecture tries to tackle this issue on a higher level than regularization.

## 2.2. LSTM Architecture

To go from a RNN to a LSTM we replace hidden units with components known as memory cells. Our presentation and notation follows (Lipton, 2015).



Intuitively, RNNs have long-term memory in the form of matrix weights, they change during the training; encoding through training some general knowledge about the data. They also have short-term memory in the form of activation passing from each node to successive ones. The memory cell introduced in the LSTM model formalizes those notions and provides a framework to control their impact on the internal representation of the network.

- $f_t, i_t, o_t$ : Respectively forget, input and output gates.
  - Sigmoidal units activated through  $x_t$  (current input) and  $h_{t-1}$  (past hidden unit output)
  - $f_t$  controls the recursive flow
  - $i_t$  and  $o_t$  control the input and output flow respectively
  - $h_t = o_t \odot \tanh(c_t)$  where  $\odot$  denotes element wise multiplication.
- $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ : The cell which has a self-connected edge with a fixed unit weight, thereby delegating control of recursion to the gate

## 3. Implementation

Four text sequences (referred to as datasets later on) were used for training: excerpts from Shakespeare’s plays, books of the Harry Potter and Lord of the Ring series and a list of popular quotes. All in English and ASCII encoded. Punctuation and structure was left unprocessed. Each dataset was split in sequences of 50 tokens (i.e. “words”) and a dictionary was built from the complete input ( $\approx 60k$  unique tokens), defining the input space for the networks. Vector encoding of this space was used through an embedding layer mapping the words to a real vector space of dimension 256. Available embeddings such as word2vec and glove were initially used but proved to be more cumbersome than our own trained version. Five LSTM networks were trained with the above, Four generators and a classifier. Training was achieved at “word” level (strings tokenized by whitespace). Character level had been previously envisioned for the rest of the project as it is more flexible and can learn new words and structural information. Despite this we kept the training at word level for multiple reasons, the primary one being robustness towards unicode characters which might vary between versions of the text available and the second one being speed of convergence. (Graves, 2013) for the generators)

To generate the sequences the trained models were initialized with a random word drawn from the dictionary and the most likely next word was fed back in the network until the desired sequence length was reached. 1000 sequences (250 per model) were generated. The classifier was then used to automatically estimate which dataset (or model, equivalently) was used for its generation, thereby giving us some “metric” of the quality of the data; expecting data generated from the Shakespeare trained model to consistently differ from the data “generated by Harry Potter”.

## 4. Empirical Results

## 5. Conclusion and Further Work

## 6. Citations and References

### References

- Choe, Yo Joong, Shin, J. C., and Spencer, Neil. Probabilistic interpretations of recurrent neural networks, 2017.
- Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. Deep Learning. The MIT Press, 2016. ISBN 0262035618, 9780262035613.

Graves, Alex. Generating sequences with recurrent neural networks. CoRR, abs/1308.0850, 2013. URL <http://arxiv.org/abs/1308.0850>.

Lipton, Zachary Chase. A critical review of recurrent neural networks for sequence learning. CoRR, abs/1506.00019, 2015. URL <http://arxiv.org/abs/1506.00019>.

Lipton, Zachary Chase. The mythos of model interpretability. CoRR, abs/1606.03490, 2016. URL <http://arxiv.org/abs/1606.03490>.