# Neural-Based Confession Generators

Peniel Argaw[1], Rebekah Bell[2], Remi Mir[3]

[1]peniel@mit.edu [2]bellr@mit.edu [3]rmir@mit.edu

## Abstract

*The rise of GPU computing power in the past decade has made it possible to levy deep learning approaches towards different tasks in natural language processing research. One example is text generation, a complex unsupervised learning task. A model must learn the underlying distribution of training data to organize characters or words from a given vocabulary into a coherent structure.*

*We implemented[1] and trained multiple recurrent neural network architectures on posts from the MIT Confessions Facebook page, using both character-by-character and word-by-word approaches for text generation. We also designed a model to simulate more popular posts by weighting training samples with the number of 'reactions' received on Facebook. We evaluated models both qualitatively and quantitatively (using perplexity and a novel 'confidence' metric), and found that the best performing model was a word-based LSTM model.*

## 1. Introduction

Text generation is an active area of research in natural language processing (NLP), with applications ranging from dialog systems to story writing. The main challenge of text generation lies in a model learning the underlying structure of input texts, such as grammar and vocabulary, which can have complex rules and idiosyncrasies across different datasets. In this paper, we describe an empirical study focused on generating texts in the style of posts made on the MIT Confessions Facebook page[2] using different types of recurrent neural models.

### 1.1 Generative Language Models

Statistical-based language modelling produces probability distributions over sequences of words. In the context of NLP, these distributions can be generated with both neural and non-neural approaches. While taking into account the context of a given word, non-neural models such as n-gram models can be limited in that they do not handle unseen words or phrases well, unless smoothing or interpolation of multiple n-gram types is applied. Additionally, as a larger training set is used, the vocabulary size increases, making these models less efficient to train. Neural models like recurrent neural networks (RNNs), can mitigate this 'curse of dimensionality' by condensing words into lower dimensional spaces and feeding them through a network.

RNNs can be used as generative models because, in addition to categorizing gradient signals from backpropagation [1] to produce distributions, they attempt to model how the data itself is generated, which requires understanding the joint distribution $p(x,y)$ of input $x$ and output $y$ [2]. (In the text generation setting, $y$ is $x$.) This is in contrast to discriminative models, which are only concerned with categorization by learning $p(y|x)$ and thus cannot produce samples similar in nature to $x$. We use RNNs to learn the underlying structure of a non-standard dataset, MIT Confessions.

### 1.2 MIT Confessions

MIT Confessions is a Facebook page that reaches over 18,600 people. Members of the MIT community can post anonymously about any aspect of Institute life. Submissions are made via a Google form and curated by an anonymous administrator. Each confession is

[1] https://github.com/pargaw/MIT_Confessions_Bot

[2] https://www.facebook.com/beaverconfessions/

prefixed with a number, e.g. '#10449',) and can receive comments and reactions (see Section 2.2.4) from users.

The motivation behind generating texts in the style of MIT Confessions was two-fold:

1. The dataset itself is unique, created by a diverse set of authors, making generation a novel empirical study.
2. We wanted to confirm whether certain types of generations would be more prevalent than others in terms of subjective content. For example, given that most posts are of a complaintive or sensual nature, we hypothesized that model-generated text would also have similar themes appearing more frequently.

## 2. Experimental Design

### 2.1 Dataset and Pre-Processing

We scraped 2,503 confession posts from February 19, 2013 (the creation of the page) to September 22, 2017, after adapting[3] an open-source scraper [3]. To mitigate the risk of a small dataset, we scraped comments from the page as well.

For both statuses and comments, we ran a custom filtering algorithm. Specifically, we replaced tagged names with a <*name*> token, and removed links, non-ASCII characters, and emoticons. Any statuses or comments longer than 280 characters or shorter than 50 characters were filtered out. In order to keep comments consistent with the structure of a standard confession, each comment was prefixed with a pound sign (#) and a randomly generated confession number.

After filtering, we obtained 1,760 confessions and 1,862 comments. An empirical suggestion for RNN language models to produce reasonable results is that the training corpus has at least 100,000 characters [4]. Our dataset contained 419,695.

[3] https://github.com/beckybell13/facebook-page-post-scraper

### 2.2 Model Architectures

Given the variety of ways to construct a language model, one of the aims of this paper is to compare RNN architectures (LSTM vs. GRU) and representation of input text (character vs. word). In this section, we provide a brief overview of each approach, as well as any implications for the Confessions dataset. We also discuss two extensions to these architectures. Our baseline was a LSTM character-by-character model.

#### *2.2.1 LSTM and GRU*

Vanilla RNNs suffer from vanishing or exploding gradients, and are unable to capture longer-term dependencies in input. We instead use Long Short Term Memory (LSTM) networks [5], which use gates (input, forget, and output) and memory cells to take into account historical context of a text.

In contrast, a Gated Recurrent Unit (GRU) [6] is a simpler, more computationally efficient variation of an RNN that can also capture long-term dependencies. It effectively combines the input and forget gates of an LSTM into a single update gate. We consider both LSTMs and GRUs as they have demonstrated comparable performance in sequence modeling [7] [8].

Each LSTM and GRU model in our experiments consists of two stacked layers of 512 hidden units with dropout in between for regularization. The first layer encodes text input into a latent representation, while the second layer is a decoder that aims to reconstruct the original input from the latent features.

#### *2.2.2 Representation of Input Text*

In addition to trying different architectures, we compared two methods of input representation. The first represents text as a set of individual characters and the second as a set of words.

Standard tokenization was carried out to create a vocabulary for both character and word-based models with a few changes: In both, stop symbols were added to the ends of texts, and texts shorter than 287 characters or 81 words were padded for character and word-based models respectively.

For word-based models, standard, off-the-shelf tokenizers like NLTK were unsuitable, as we aimed to retain course numbers (e.g. 6.01) and plain-text emoticons in our dataset. We also wanted to keep models from learning ubiquitous but otherwise uninformative numbers, such as confession IDs. We tokenized those under a <*number*> token and captured other domain-specific features with a custom regex parser.

Character-based models create lexical units more flexibly than word-based models, which are forced to make pre-training assumptions about what words are possible from a fixed vocabulary. This limitation led us to add an <*unknown*> token to the vocabulary to allow the model to handle input that contains unknown words.

With tokenization, the resulting vocabularies for the character and word-based models were 94 and 7,486 tokens, respectively.

### *2.2.3 Convolutional LSTMs*

One extension to the neural architectures discussed earlier was adding convolutions. When learning how to generate confession data, it is critical for a model to leverage context from across the entire input. While LSTMs and GRUs can accomplish this to a certain extent via gates that capture history and previous outputs, we can convolve over the input beforehand to extract more features. We do this without greatly increasing the number of parameters or computation time, as convolutional layers are parallelizable, unlike RNNs [9]. Often used for computer vision tasks, we adapt convolutions to the text domain by representing dimensions as follows: width to maximum sentence length, height to 1, and depth to a number of embedding dimensions (64).

### *2.2.4 Reaction Weighting*

An MIT Confession post can receive 'reactions,' i.e. indications by users that they liked or loved the post, or were amused, surprised, saddened, or angered by it. Assuming that the most popular posts are the ones with the most reactions, we ran an experiment that weighted training texts by the number of reactions they received for our baseline character-based LSTM model.

## 2.3 Text Generation

The trained models were used to generate text in the following manner: given a seed string, the model predicts the most likely character that follows. This character is appended to the seed string and the new string is fed back into the model as input for the next prediction. This process is repeated until the seed string reaches a maximum length of 287 or a stop symbol is predicted. Due to different vocabularies for character and word-based models, seed strings were adjusted accordingly, e.g. "#1234" for character-based models and "# <number>" for word-based models.

We also experimented with softmax temperature [10] to predict different probability distributions from which to sample the next inputs. The softmax equation is adjusted as follows for temperature *T*:

$$e^{z_j/T} / \sum_{k=1}^{K} e^{z_k/T}$$

The original softmax has T=1, where we sample greedily using argmax. In general, increasing the temperature allows for more diverse predictions, as the variance of the probability distribution increases. This leads to more 'mistakes' in spelling in the case of character-by-character models and more 'mistakes' in grammar in the case of word-by-word models. Therefore, pushing temperature closer to infinity would create more nonsensical text [11].

## 2.4 Evaluation

While human evaluations of model output in comparison to input texts are commonplace for NLP tasks, we did not have enough resources to use services like Amazon Mechanical Turk. There do exist automated metrics such as BLEU, ROUGE, and perplexity. The first two are n-gram based scores for precision and recall, respectively, of texts, originally designed for machine translation.

Given that our generation task was concerned with producing texts of a similar nature to the inputs, but not with the goal of complete reconstruction, we used perplexity as our primary model evaluation metric. Perplexity captures the likelihood of model predictions. The better the model (i.e. the more it has learned), the lower the perplexity. The perplexity of a probability model can be calculated as $2^{H(p,q)}$ where $H(p,q)$ is the cross-entropy between the empirical and predicted probability distributions. Because we measure loss with categorical cross-entropy, we can denote perplexity as $2^{loss}$ [12].

We also quantify model confidence, another metric that captures prediction likelihood. It rewards each time-step that has high prediction certainty as indicated by distributions concentrated on a particular unit (character or word, depending on the model), as long as the degree of concentration exceeds a threshold $R >> |\text{VOCABULARY}|^{-1}$, the latter of which is a lower bound equivalent to a random guess of the next unit (**Equation. 1**). The metric is flexible as $R$ can be adjusted.

$$\frac{1}{|Y|}\sum_{i=0}^{|Y|}\left(\frac{1}{len(y)}\sum_{y\in Y,\, j=len(s)}^{len(y)} Q\right)$$

$$where\ Q = [[\, p(y_j \mid y_{j-W}, y_{j-W+1}, \ldots y_{j-1}) > R]]$$

**Equation. 1** Model confidence captures the mean likelihood of predictions per generated text $y$, and then averages individual confidences over an entire generated corpus $Y$. Other parameters are: seed string $s$ for initiating the generation of $y$, window size $W$ of previous predictions to pass in at any given time-step, and threshold $R$ that must exceed $|\text{vocabulary}|^{-1}$. We chose W=10 and R=0.8.

We used loss, perplexity, and confidence to identify the best-performing model architectures. We also subjectively evaluated quality of generations in terms of spelling, grammar, and relevance of texts to life at MIT.

## 3. Results and Analysis

We implemented all neural architectures with Keras [13] and trained on Amazon Web Services GPUs. We ran implementations of LSTM and GRU for both character and word-based models as described in Section 2.2. We also ran a reaction weighted character-based LSTM, and convolutional LSTMs for both character and word-based approaches.

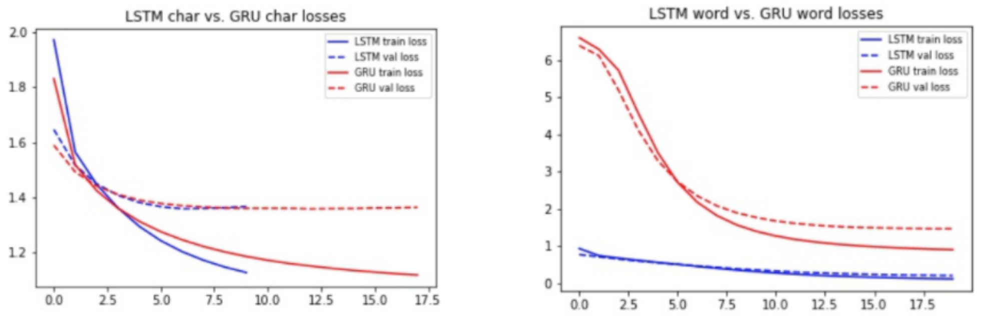

**Figure 1a**. The losses of LSTM versus GRU in both the character and word-based model

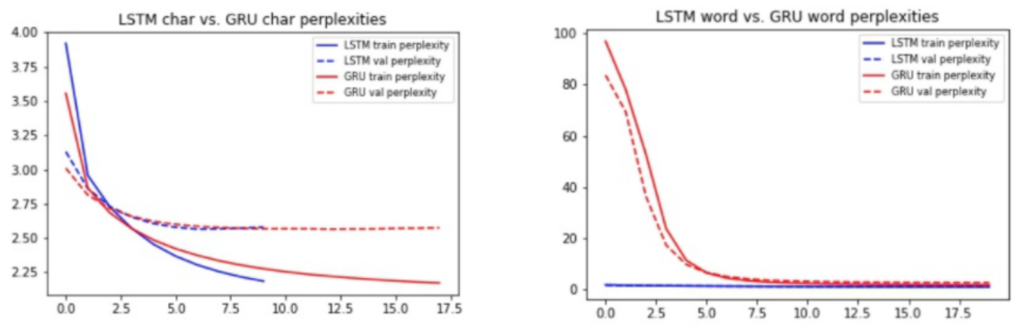

**Figure 1b**. The perplexities of LSTM versus GRU in both the character and word-based models

## 3.1 LSTM vs. GRU

We discuss and compare performance of LSTM and GRU. Figure 1 shows the loss and perplexity of these models during training.

### *3.1.1 Quantitative Evaluation*

To compare performance quantitatively, we looked at confidence, perplexity and losses. Table 1 shows the confidence of each model using softmax temperature samplings of T=1 and T=0.5, respectively. The confidence of GRU is higher than that of LSTM overall.

We note a few observations from Figure 1: The character-based LSTM and GRU models have comparable differences in training and validation loss and have similar losses at the end of training, although the LSTM model converges much faster than the GRU model. The word-based LSTM, however, has a much smaller difference in training and validation losses compared to the word-based GRU. We attribute this in part to how the word-based GRU model overfit the training data for reasons discussed later in this section. Overall, LSTM has lower loss and perplexity.

### *3.1.2 Text Generation*

These models had difficulty generating any thought longer than a few words, with a few misspellings in the case of the character-based ones. Many generated confessions were incoherent, e.g. for the seed string *"# <number> 6.01"* with temperature T=0.5, the baseline LSTM generated:

| **Model** | *Char.* | | *Word* | |
|---|---|---|---|---|
| | T= 1 | T = 0.5 | T = 1 | T= 0.5 |
| *LSTM* | 0.5 | 0.62 | 0.43 | 0.55 |
| *GRU* | 0.58 | 0.71 | **0.63** | **0.78** |
| *LSTM reacts* | **0.64** | **0.72** | - | - |

**Table 1** Confidence scores (best are bolded) with softmax sampling for temperature values 1 and 0.5.

*#3476 6.01 is actually submitted to Stop in for arthing is a good friend of my friend whenever I walk through it. Work in a group of girls in MIT who are fine and graded their stuff.*

The character-based GRU model generated:

*#3476 6.01 secret I have to take the title?*

Both types of models also outputted repetitive strings. For the seed string *"#"* with T=1, the character-based GRU model generated:

*#9115 The odds of the confession wrong with all of the confession wrong with all of the confession wrong with all of the confession wrong with all of the confession wrong with...*

Although there were nonsensical results, both models were still able to produce text that contained MIT-related themes. One of the more coherent examples that was not merely copying training data was produced by the word-based LSTM model:

*#<number> i ' ve realized a single dream of my floor.*

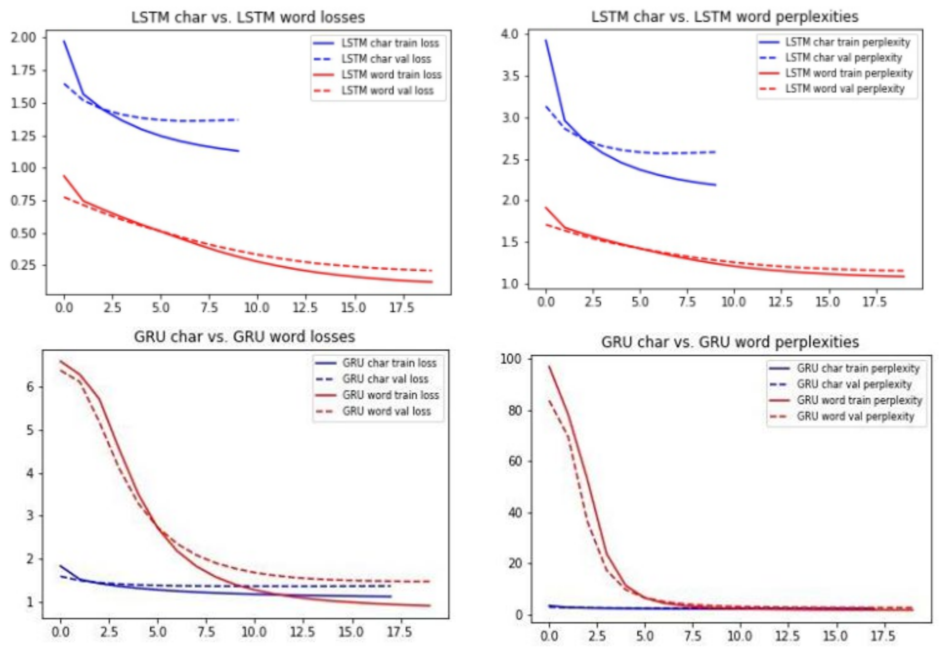

**Figure 2.** Loss and perplexity plots for the character and word-based LSTM and GRU models

As mentioned, we observed that the word-based GRU model may be overfitting. This became apparent with text generation, as it essentially memorized parts of the training data and produced some samples verbatim for multiple seed strings. We attribute this to excessive hidden units. Although we kept the number of hidden units the same for both GRU and LSTM for comparison, given this result, it would be worthwhile to train GRU on a smaller number of hidden units. As we discuss in the next section, word-based models in general are more prone to overfitting.

### 3.2 Character versus Word

As shown in Figure 2, the word-based LSTM model appears to have lower loss and perplexity than the character-based LSTM model.

For the GRU architecture, the word-based model starts out with higher loss and perplexity but eventually converges to values similar to those of the character-based models.

From Table 1 we can see that the word-based GRU model has a slightly higher confidence than the character-based GRU model. This is the opposite for LSTM models.

We suspect that word-based models are prone to overfitting, especially when considering the text generated by these models. As mentioned earlier, the word-based GRU model memorized much of the training data. We also saw similar, if not as extreme, patterns of overfitting in the word-based LSTM model as well. For example, for the seed string *“# <number> 6.01”*, the character-based LSTM returned:

*#3476 6.01 TAs what we had that much more like it when people that doesn't have to wake up for the rest of us. You are not derived from wearing a crush on me. hot drunk etckes.*

While the word-based LSTM model returned:

*# CUSTOM_NUMBER :) i ' m not sure if global warming is man made or a party ; and am a lot , i ' m failing the end.*

Although the response from the word-based LSTM model seems the most coherent, some of the generated texts do contain subsets of different training examples. For example, in the generated confession above, the subset, *"I'm not sure if global warming is man made"* can be found in the dataset while the remainder of the sentence is more original.

Despite some overfitting, the word-based LSTM model was still able to generate portions of unique content and had the lowest loss and perplexity. We therefore classified this to be our best-performing model.

## 3.3 Convolutional LSTM

We ran two (character and word-based) LSTM models with an additional embedding and 100-dimension convolutional layer preceding the stacked LSTM layer. The convolutional layer had a stride of 1, and used ReLU activation. While we hypothesized that models with convolutional layers would perform better by extracting features prior to auto-encoding, they failed to do so in practice. We attribute this to a small number of hidden features that were insufficiently able to capture the complexities of the discrete data. Thus, when feeding forward, the following neural layers were unable to create meaningful dense representations. Although we did not have enough resources to explore a larger hyperparameter space, next steps would involve training a convolutional model with a higher number of hidden units.

## 3.4 Reaction Weighting

As discussed in Section 3.2, based on our perplexity and loss metrics as well as qualitative analysis, we found the word-based LSTM model to have the best performance. Although we would have liked to run that with reaction weighting, due to limited funding on AWS, we were not able to do so. Consequently, we used reaction weights on a less computationally intensive model, the baseline character-based LSTM.

With regards to whether certain themes are more prevalent than others in the generated text, we consider these cherry-picked outputs with T=1, 1, and 0.5, respectively.

*#1150 If I had Puerto Rican citizenship today…*

*#3476 i'm sick of my entire existence of us made a giant gross fake was never been to the beautiful girl at her fathers from 2.007 this semester.*

*#3476 :) I told my source of shallow and see if you want to know what I'm done with me. It hesolation, bullying is overhyped for progress and have sex with my UROP data in Excel wind a good oce*

Although our analysis is limited in that we only sampled a few of many possible seed strings, the above confessions do capture political and sensual themes, as well as themes that tend to attract sympathy, which are reflective of the general confessions that garner most reactions on the Facebook page.

## 4. Conclusion

Analyzing loss, perplexity, confidence, and generated text, we found word-based LSTM to be the best-performing model. It had the lowest losses of all the models, and produced unique, more coherent text most of the time.

In general, character and word-based models have their advantages and disadvantages. Character-based models are more flexible as they need only a small vocabulary in order to generate a word. A disadvantage, however, is that more hidden layers are needed than word-based models to learn longer term dependencies, as the maximum length is longer in number of units. Word-based models also do not have to learn spelling, but merely the syntax, or order of units, while character-based models must learn both. Overall, we found that word-by-word models tend to produce more coherent results, but risk overfitting. Fewer layers or units may mitigate this risk on smaller datasets like ours.

For the character-based models, we found that both LSTM and GRU models had comparable performance. We suspect that the word-based GRU model was more prone to overfitting than the word-based LSTM model due to the number of hidden units. As for reaction weighting, certain confessions showed inclinations towards themes reflective of the most popular confessions on MIT Confessions, such as politics and sex. Text generation from a larger set of seed strings would allow for a more comprehensive analysis, such as with LDA topic modelling.

A deeper exploration of the hyperparameter space is also needed to better understand the performance of architectures presented in this paper. For example, in the case of the word-based GRU, which clearly overfit, training additional models with a reduced number of hidden units and increased dropout would likely produce a better model. Increasing the number of hidden units in the embedding layer may produce better results for the convolutional model. Future research can also include variational autoencoders [14], or bidirectional LSTMs to account for context from both directions of a given input text.

Tuning neural hyperparameters is a time and resource-consuming process, so the work presented in this paper scratches the surface of exploring what these models are capable of. However, although we cannot replace the authors of MIT Confession posts just yet, perhaps we can create a “Drunken” MIT Confessions page with our generations[4].

## 5. Division of Labor

Peniel Argaw created the AWS environment for running models. She also worked on evaluating model performance, including the implementation of text generation to predict confessions and evaluating the results using perplexity and confidence metrics.

Rebekah Bell adapted the open-source Facebook post scraper for data collection, including the addition of a filtering algorithm to remove noise from the data. She helped with Jupyter notebook and GPU configuration on AWS and text generation for confession predictions. She also incorporated reaction counts into the sample weights.

Remi Mir identified the dataset, methods for expansion and filtering, and goals of the project. She designed the confidence metric to supplement perplexity as another means of quantitative analysis, and implemented the following: the neural architectures, both text representation approaches (and corresponding regex parsers), and sampling methods (softmax temperature).

Much of the work was done during meetings where all authors participated in architecture design and code review.

[4] https://github.com/pargaw/MIT_Confessions_Bot/blob/master/Confession_Appendix.pdf

## 6. References

1. LeCun Y, Bottou L, Orr GB, Müller K-R. Efficient BackProp. Neural Networks: Tricks of the Trade. Springer, Berlin, Heidelberg; 1998. pp. 9–50.

2. Mikolov T, Zweig G. Context dependent recurrent neural network language model. SLT. microsoft.com; 2012; Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2012/07/rnn_ctxt_TR.sav_.pdf

3. Woolf M. Data scraper for Facebook Pages [Internet]. Github; Available: https://github.com/minimaxir/facebook-page-post-scraper

4. Chollet F. LSTM Text Generation. In: Github [Internet]. Available: https://github.com/fchollet/keras/blob/master/examples/lstm_text_generation.py

5. Hochreiter S, Schmidhuber J. Long short-term

memory. Neural Comput. 1997;9: 1735–1780.

6. Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014. doi:10.3115/v1/d14-1179

7. Chung J, Gulcehre C, Cho K, Bengio Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [Internet]. arXiv [cs.NE]. 2014. Available: http://arxiv.org/abs/1412.3555

8. Jozefowicz R, Zaremba W, Sutskever I. An empirical exploration of recurrent network architectures. Proceedings of the 32nd. jmlr.org; 2015; Available: http://www.jmlr.org/proceedings/papers/v37/jozefowicz15.pdf?utm_campaign=Revue%20newsletter&utm_medium=Newsletter&utm_source=revue

9. Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. Proc IEEE. cv-foundation.org; 2015; Available: http://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Long_Fully_Convolutional_Networks_2015_CVPR_paper.html

10. Sutton RS, Barto AG. Reinforcement learning: An introduction. cell.com; 1998; Available: http://www.cell.com/trends/cognitive-sciences/pdf/S1364-6613(99)01331-5.pdf

11. Karpathy A. The unreasonable effectiveness of recurrent neural networks. Andrej Karpathy blog. 2015;

12. Perplexity - Wikipedia [Internet]. [cited 12 Dec 2017]. Available: https://en.wikipedia.org/wiki/Perplexity#cite_ref-1

13. keras [Internet]. Github; Available: https://github.com/keras-team/keras