# Paper

* Andrej Karpathy, Justin Johnson, Li Fei-Fei. “Visualizing and Understanding Recurrent Networks”.

# Abstract

In our project, we tried to reproduce most of the paper’s figures (elaborated in part 04) using our own implementation of the paper’s Recurrent Neural Networks with the help of ‘PyTorch’.  
We managed to reproduce the test set cross-entropy loss table for all models (figure 1), graphic cell and gate visualization (figure 2 & 3) and lastly, venn diagram of three studied networks (figure 4).

SUCSESS OR FAIL: TBD.

# Introduction

The paper “Visualizing and Understanding Recurrent Networks” tries to shed a light on the source of performance and the limitations of Recurrent Neural Networks (RNNs) along with their variants (LSTM and GRU). In order to analyze these models, the researchers used character-level language models (LM’s) as a testbed. In particular, the experiments revealed the existence of interpretable cells that keep track of long-range dependencies. They compared their analysis against finite horizon n-gram models which supposedly lack long range structural dependencies. The paper presents those results and suggests areas for further study in the subject.

# Methodology

The researchers didn’t supply any of their source code for the networks or their analysis which meant the we needed to write it on our own. In order to mimic the experiments, we did use most of the parameters that were mentioned in the paper (part 3.3) and added some of our own to achieve best results and overcome technical difficulties.

In addition, we only used the enormous book “war and peace” by Leo Tolstoy as a dataset since the Linux Kernel source code has overblown since the paper was written and was too big for our machine.

We implemented the paper’s experiments using Python with two main frameworks, ‘PyTorch’ and ‘PyTorch-Lightning’.

For the training procedure we used ‘PyTorch-Lightning’ and for each RNN architecture we used ‘PyTorch’s own implementation. We will explore LSTM, GRU and RNN each with hidden size of 32, 64, 128, 256 with 1 to 3 layers. The hyper parameters for the training phase are mostly the same as the paper.

The hyper parameters we used:

* 50 Epochs
* Batch size of 100
* Sequence length of 100
* Dataset splits percentages of (80, 10, 10) for (training, validation, test)
* Learning rate of with gamma equals 0.95 and learning rate decay of 10.
* We decided to omit truncated back propagation because the training phase with it took too long and with basic hardware it just wasn’t feasible. On the other hand, we added gradient norm clip of size 5 to overcome the vanishing gradient problem (“normalizing” the gradient to be of norm 5 during backpropagation).

The analysis part of our project consists of two parts:

* Network simulation with gates and cells extraction: implement our own “forward pass” with additional gate extraction.
* Visualization methods for graphic data presentation.

We trained the networks using “Google Collaboratory” environment with the free GPU

hardware supplied by them. The analysis part ran on our CPU-only machine.

# Experiments

Firstly, we trained each model, layer and hidden size on the training set of the “War and Peace” dataset. For each model, when the training phase has ended, we saved the network locally in order to produce the following analysis.

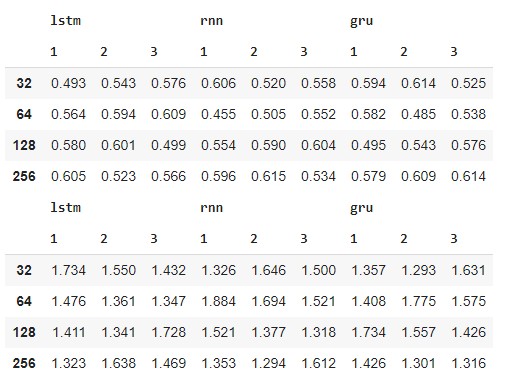
Figure 1: Accuracy (top table) and Cross-entropy loss (bottom table) for the test set on each model. Models in each row have a nearly equal number of parameters. We took each saved network and ran the test set on the network and kept the mean loss and mean accuracy.

Figure 2: Cell Visualization – Helps us to visualize cells with interpretable activations in our “War and Peace” LSTMs and GRUs where the text color corresponds to , where -1 is red and +1 is blue. This experiment enables us to identify whether our networks can detect high-level patterns. We used a web interface supplied by “[huanghao-code](https://github.com/huanghao-code)” to showcase cell activation values for a selected cell in the entire network on a small section of the test set.

Figure 3: Saturation plots for 3-layered LSTM and GRU models. For the LSTM figure, the plots correspond to forget, input and output gates. Each circle in the figure is a gate in the LSTM and its position is determined by the fraction of times it’s left or right saturated. We consider a gate to be left or right saturated if its activation is less than 0.1 or higher than 0.9 respectively. For GRU figure, the plots correspond to update and reset gates.

Figure 4: Venn diagram – overlap between test set errors between our best RNN, GRU and LSTM networks. We loaded the three networks and for each one we saved its correct examples in a set data structure. Then, we compared each network’s set against the others in order to seek for similarities between the three.

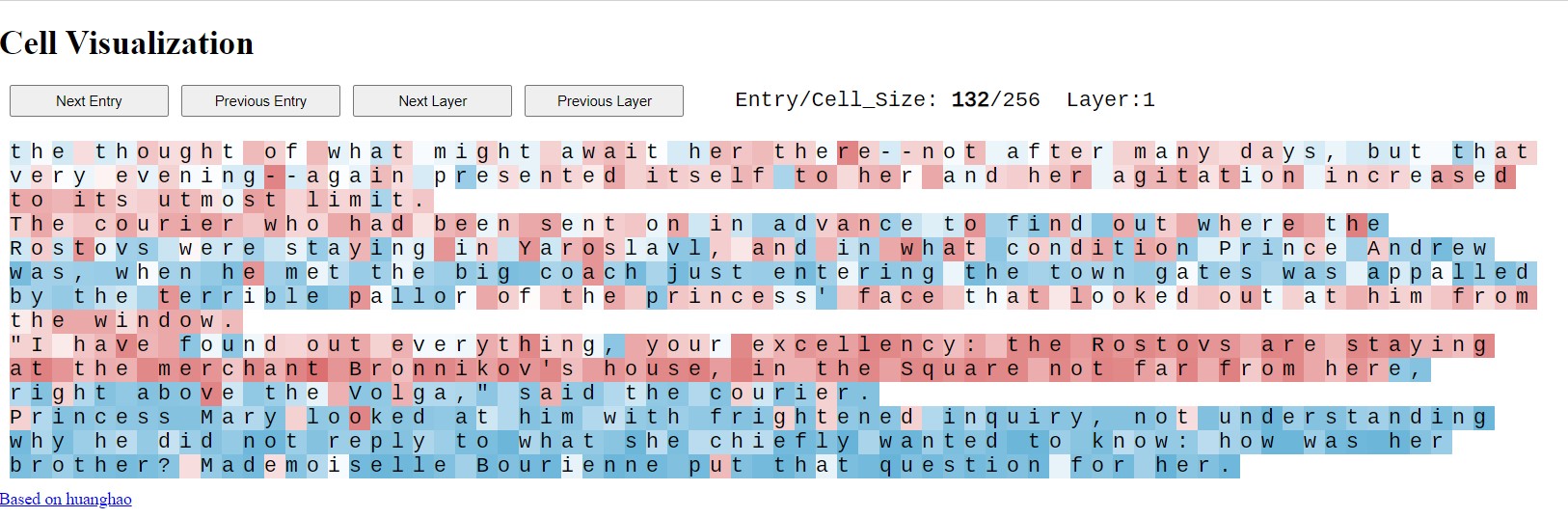
# Results



Figure

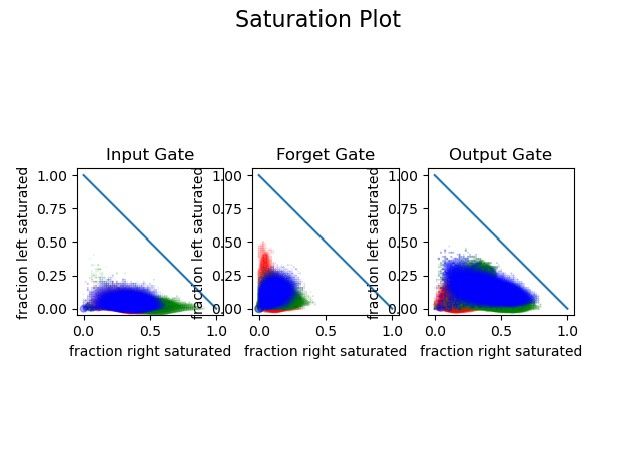
Our figure 1 reproduction attempt resulted in mixed results. We got better cross-entropy loss on LSTM models with hidden size of 64 and on RNN models with hidden size of 256 but on the others, we mostly got higher loss. In our project, we used the same parameters as the researchers detailed. This fact leads us to the conclusion that the difference between the results lays on other parameters that were not specified or different models’ implementation.

As in the paper, we also got to the conclusion that depth of at least two improves the results, highly beneficial and usually between depth 2 and 3 the results are mixed. Furthermore, in most cases the LSTM models outperform the RNN models.



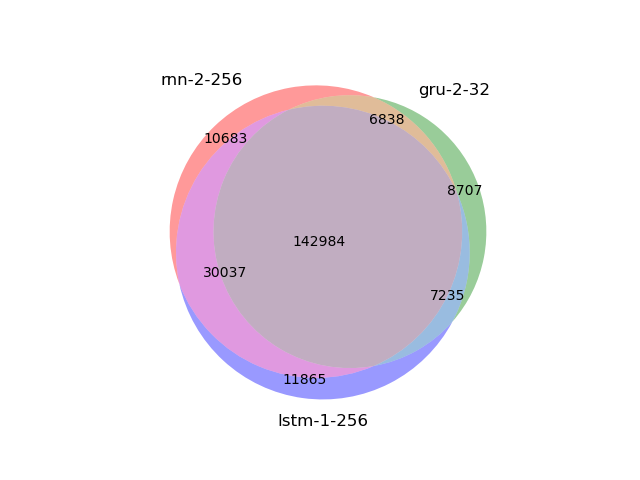
Figure

Blah blah blah



Figure

Blah blah blah



Figure

Blah blah blah