TEXT GENERATION WITH RECURSIVE NEURAL NETWORKS - ASSIGNMENT 7

Mirko Morello
920601
m.morello11@campus.unimib.it

Andrea Borghesi
916202
a.borghesi1@campus.unimib.it

May 11, 2024

1 INTRODUCTION

In this assignment, we explore and compare the performance of different RNN architectures, including classic RNNs, LSTMs, and GRUs, on a character-level language modeling task. We aim to understand the strengths and limitations of these architectures and investigate how they perform on a specific dataset and task.

2 TECHNOLOGIES USED

2.1 Recursive Neural Networks

Recurrent Neural Networks (RNNs) have emerged as a powerful class of neural network architectures for modeling sequential data, such as text, speech, and time series data. Unlike traditional feedforward neural networks, RNNs are designed to process inputs sequentially, allowing them to capture and model the temporal dependencies and patterns present in sequential data.
The core component of an RNN is the recurrent cell, which takes the current input and the previous hidden state as inputs and produces the current hidden state and output. This recurrent structure enables RNNs to maintain a memory of past inputs, making them well-suited for tasks that require capturing long-range dependencies.

2.2 Gating mechanisms

The classic RNN architecture suffers from vanishing and exploding gradient problems, which can make it challenging to learn long-range dependencies effectively. To address these issues, variants of RNNs, such as Long Short-Term Memory (LSTM)[2] and Gated Recurrent Unit (GRU)[1], have been proposed. LSTMs and GRUs introduce gating mechanisms that control the flow of information through the recurrent cell, allowing them to remember or forget information from previous time steps selectively. These gating mechanisms help mitigate the vanishing and exploding gradient problems, enabling these architectures to capture long-range dependencies more effectively than classic RNNs.

3 DATA

Our models have been trained on the raw text of Alice in Wonderland by Lewis Caroll. It has been cleaned from punctuation, upper case letters, and numbers. To further prepare it we divided the text into bite-size sequences of 100 characters each to be fed into the network.

4 APPROACH

The approach methodology is very standard. After the data has been properly pre-processed, we define the architectures. Three main architectures have been proposed, each using different types of recurrent layers. Each architecture is rather simple to implement thanks to the abstraction layer provided by PyTorch. Each implementation will be as shown in Table 1. After the recurrent layers, a simple fully connected layer passes the output of the recurrent layer to a single output node.

Layer	Param #
Recursive Layer: 1-1	number of params
Linear: 1-2	27,675

Table 1: Network Summary

4.1 Architecture Details

We implemented four different RNN architectures: a basic RNN, a Gated Recurrent Unit (GRU), a Long Short-Term Memory (LSTM), and a Bidirectional LSTM (BiLSTM).

For the basic RNN, we used 512 hidden units and 6 stacked layers, with a dropout of 20%, totaling 8,401,920 parameters. Although our hardware resources allowed for more complex architecture, we experimentally observed that it quickly worsened in performance. The vanishing gradient problem might justify such behaviour, a well-known limitation of basic RNNs when dealing with long-range dependencies in sequential data.

For the GRU, we used 512 hidden units, 6 stacked layers, and a dropout of 20%, totaling 25,233,435 parameters. GRUs are known to capture long-term dependencies more effectively than basic RNNs, while having fewer parameters compared to LSTMs, allowing for a good balance between complexity and performance.

The LSTM architecture consisted of 512 hidden units, 6 stacked layers, and a dropout rate of 20%, totaling 2,903,835 parameters. Although it is easy to get a very complex network by incrementing the number of hidden units, we saw that increasing the depth (number of stacked layers) was more effective in capturing long-range dependencies in our text data. Exactly like the other two architectures, we then implemented a bidirectional configuration to capture context both from past and future inputs. By still using 512 hidden units, 6 stacked layers, and a dropout rate of 20%, we raised the total parameters to 33,635,355, the bidirectional configuration enabled the network to incorporate context from both directions, potentially enhancing its ability to capture patterns in the text data.

4.2 Metrics and Validation

Since this is a classification model where we have to predict the next letter, the metric used to calculate the loss was cross-entropy.
If we try to understand the semantic meaning of cross-entropy, in this specific case, we conclude that it tries to measure how well we're able to predict how the book should continue.
Since creating a model that can rewrite word by word Alice in Wonderland from a portion of it (that it has never seen) is not a feasible task, we can expect the validation's cross-entropy to never reach a low value just by changing the type of gating mechanism, so this isn't the only metric we're going to look at when evaluating our models, but it will be the one used for training.
One of the handy insights the cross-entropy can still give us is the bias-variance tradeoff. We can monitor our training and know if our models are learning letter by letter the entirety of Alice in Wonderland book from the cross-entropy of the training set, and how much it is learning general English vocabulary and grammar.
What we're interested in, it's a middle ground between these tradeoffs, we want to have emergent behaviour where our network can generate sensible English words with a large vocabulary.
For this reason, we decided to take a look at the uniqueness of the words generated by our models in 100 sentences of 350 characters in comparison to the normal uniqueness of english words found empirically in 100 sentences of 350 words extracted at random from our text.
One last metric, was to count the number of actual english words in the same pool of the aforementioned sentences, the rationale being that a robust model has learned actual english words. A good model is one that, given the same temperature as other models, is less prone to generate inexistent words due to the fact that, when in the middle of generating a word, it always keeps a distinct preference for letters that may end up creating an actual sensible english word.

5 Results

As shown in Figure 1, our network is big enough to show how RNN train-loss decreases much slower than the other architectures, this can be due the problem of the vanishing gradient. Such difference is rather signficant from LSTM and GRU. By comparing LSTM and GRU, GRU clearly

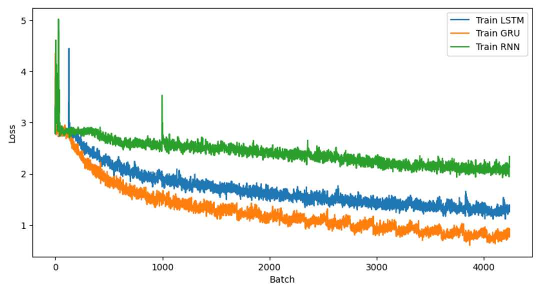

Figure 1: Train loss during 20 epochs calculated at every step

converges faster although it is a simpler architecture. This behaviour can be explained by the bias-variance tradeoff. Although LSTM is more complex and it might be able to represent harder distributions, it is more difficult to get the right configuration of weights to perfectly represent it. On the other hand GRU, by being less complex, it has fewer choices of weights configurations, and therefore it is able to converge faster, although probably to a higher point with respect to LSTM.

Another remark can be the english word rateo discussed

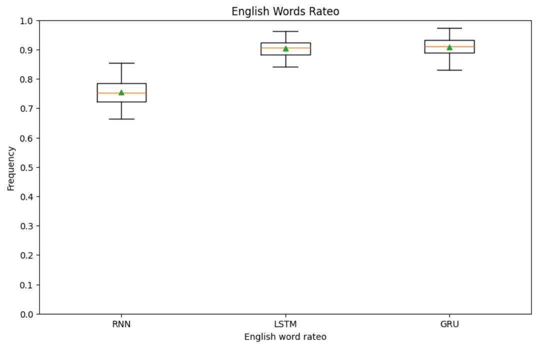

Figure 2: English word rateo of all the models

in Section 4.2. As shown in Figure 2, GRU and LSTM scores the best, although GRU, as we will show in the next section, has learned better English grammar than LSTM, and we do not have a metric to represent it. As last but not least, Figure 3 shows the unique words rateo per produced sequence. We also added the rateo of sequences taken from Alice in Wonderland to compare it to the generated data from our models. As we can clearly see, GRU again scores the best, showing a more extensive vocabulary than the other models.

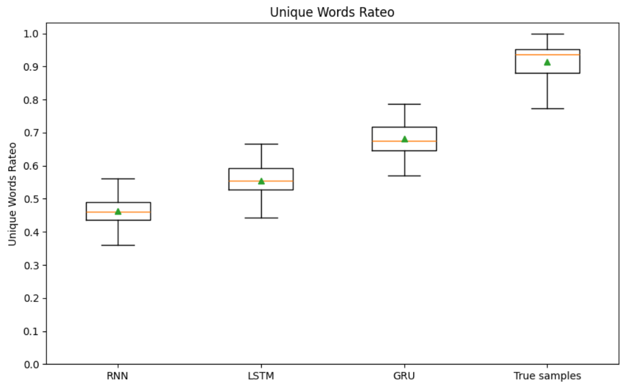

Figure 3: Boxplot describing the rateo of unique english words, with empirically observed uniqueness in non-generated text for 100 sentences of 350 letters each.

5.1 Sample Text Generation

Input sequence:

```
e her look up in a great hurry an enormous
puppy was looking down at her with large
round eyes and
```

Generated text:

1. Basic RNN

```
caso an w w fe s e she it the the cu
alicu tay say id t t ou the thor the
the horel the the the fu the on in the
the hery the fav the ray he be s du
aserow a t w asge the the tasy dou it
all it a t it alicll the thart the the
the t h and the the she and whe yhe
sting shey the the wav uhe fr the cou
well the she she sas the what the she
yhe nou alice she whe the said so thou
asing her wes and ses itte whe frow a
fere the wo and the wo the a t h the a
t id the low the she a w s wem the she
i w
```

2. GRU

```
 shoes understand it as all said the
caterpillar alice flanced rather a
great crowd as soon as she could not
to see the caterpillar and then a great
crash as if she went on we had thinking
about for it would bend i cay as soon
as she could not in the well id not
look up go on a crimson tely without
looking for it you know it was said the
caterpillar alice thought the duchess
i make you are old father william seem
them alice had been in a pleased and
those twallowing one said alice in a
sorro
```

3. LSTM

```
he sone of the same the cat and the
mock turtle in a little said alice
remarked the dormouse said alice and
the jurpos way the bestainly that saw
in the words a pinute into that i soon
and the was that all her so the exen
```

```
in the same the pight way the had not
like and the tong and the silence to
the procers would have and the first
was said alice began and the same said
the door the chimney and the others the
others and she officer it as well make
you might happen to ask she court and
the sight
```

5.2 Generation analysis

Upon analyzing the outcomes of various neural network architectures, a recurring pattern emerges wherein certain models seem to enter into a perpetual loop of word generation. This behavior can be understood by envisioning a graph $G = (V, E)$, where V encompasses all possible states generated by input/output sequences of 100 characters, hence $V = |27^{100}|$. Each vertex in this graph represents a unique state of the system, where the outgoing edges correspond to the probabilities of transitioning to the subsequent character based on the network's predictions, therefore for each vertex we have 27 outgoing edges, hence $E = |27^{101}|$. As the network responds with a non-uniform probability distribution over possible next characters, a random walk on the graph G occurs, driven by these probabilities. Consequently, regardless of the starting state, the walker tends to gravitate toward the most probable subgraphs, perpetuating the recurrence of certain word sequences. As our goal is to have an output that makes sense in the English language, we aim to stay away from this phenomenon or postpone it as much as possible. However, introducing a temperature parameter and selecting the next character using the probability vector rather than always picking the highest probability character, allos for a temporary workaround of this issue, as it can help to explore a wider range of possibilities and potentially sidestep the convergence towards specific subgraphs.

6 CONCLUSIONS

The best performing approach in our experiments has been the GRU architecture, that showed a wider vocabulary and better understanding of the English grammar, while being a lighter architecture than the second runner LSTM.

REFERENCES

[1] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.