Text Generation with Different RNN Architectures

Davide Vettore 868855

1. Objective:

The goal of this experiment is to train a Recurrent Neural Network (RNN) and some gated alternatives (Long Short-Term Memory and Gated Recurrent Unit) for single character generation. The trained networks will then be employed to perform text generation.

2. Text data:

The networks will be trained and tested using text data from *'Alice's Adventures in Wonderland'*. By leveraging the rich linguistic patterns and narrative structure present in this text, our models aim to learn and generate coherent and contextually relevant text. The raw, unprocessed text contains a total of 163.918 characters, a pre-processing pipeline is performed in order to make the text easier to work with:

- All characters are converted to lowercase.
- Non-alphanumeric characters are removed.
- The *newline* characters are removed.
- Double spaces (originated by previous operations) are replaced with single spaces.

Sample phrase before performing the pre-processing pipeline:

"She had quite forgotten the Duchess by this time, and was a little startled when she heard her voice close to her ear. "You're thinking about something, my dear, and that makes you forget to talk."

Sample phrase after performing the pre-processing pipeline:

```
she had quite forgotten the duchess by this time and was a little startled when she
heard her voice close to her ear youre thinking about something my dear and that makes
you forget to talk
```

After the pre-processing pipeline is applied to the raw text, we're left with 153,040 characters from the English alphabet.

To train our networks, we segment the processed text into sequences of 100 characters, where the 101st character serves as the target. In other words, each model aims to predict the character immediately following the 100 characters provided as input. The text data is divided into an 80% training subset and a 20% testing subset, resulting in 107.028 training patterns and 30.508 testing patterns, respectively. Additionally, the text data is encoded, with each unique character being mapped to a unique integer.

3. Procedure:

RNNs are specialized networks designed for processing sequential data, aiming to discern patterns through the sequential information they process. The computational graph of a simple recurrent network maps an **input** sequence to an **output** sequence, traversing through a **hidden layer** that evolves at each step. There are three types of connections:

- Input-to-hidden connections, regulated by a weight matrix U.
- Hidden-to-hidden connections, regulated by a weight matrix W.
- Hidden-to-output connections, regulated by a weight matrix V.

The hidden state can be now considered as the memory of the input. At each step, the output relies only on this memory and the input at that timestamp. A notable distinction between RNNs and other FNNs is that RNNs share the same parameters across all steps, making them computationally lighter. Moreover, RNNs break away from the assumption of input independence. To train RNNs, we employ a variant of traditional backpropagation called **backpropagation through time** (BPTT).

Gated RNNs, including both **Long Short-Term Memory** (LSTM) and **Gated Recurrent Unit** (GRU), are proved to be the most effective sequence models in practical applications. While **standard RNNs** consist of repeated modules performing simple non-linear computations, gated RNN modules include multiple operations. The difference is represented in Figure 1.

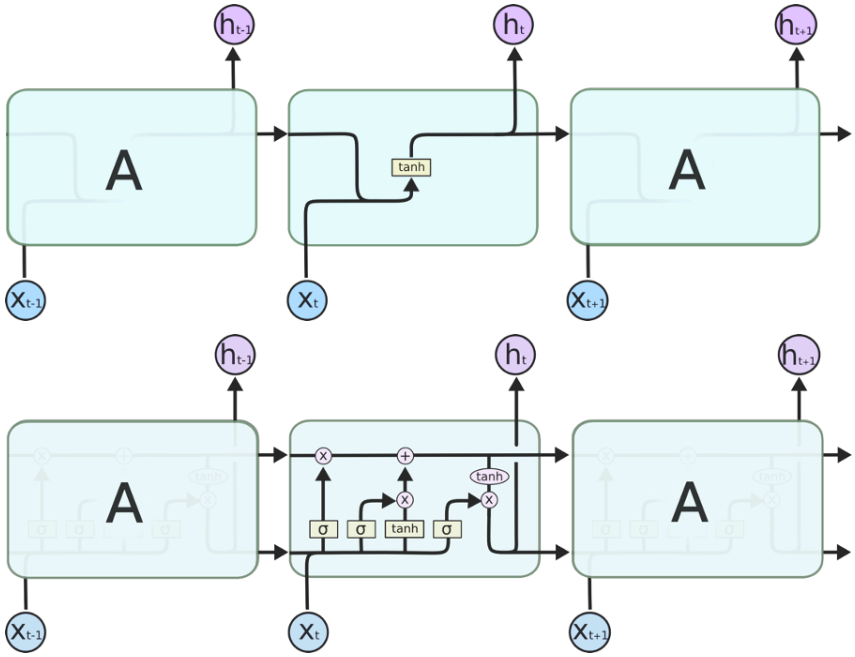

Figure 1: Repeating module of a standard RNN and an LSTM, respectively

For each element in the input sequence, each layer of an LSTM network computes:

- An *input gate*, regulating new information stored in the cell state.
- A *forget gate*, determining retention of the previous cell state.
- A *candidate value* that could contribute to the cell state.
- An *output gate*, which controls the exposure of the cell state as output.
- The *cell state*, housing long-term memory information.
- The *hidden state*, which is the output of the LSTM cell.

Alternatively, Gated Recurrent Units combine the forget and input gates into an *update gate* and eliminate the *cell state*, making the model less parameter-dense and faster.

During the experiment, we'll train the three variations of RNNs mentioned above, while also **fine-tuning parameters** such as the number of features in the hidden layer, the number of stacked RNN layers, and the inclusion of a dropout layer. These adjustments are aimed to increase generation performance.

Training will be conducted using the **Adam** optimizer and evaluated using **cross-entropy loss** as the loss function. We've set the number of epochs to 60, with a patience parameter of 8 epochs. This means that if the loss fails to decrease for 8 consecutive epochs, the training process will be stopped early to prevent overfitting.

4. Results:

Table 1 presents the performance outcomes achieved with the various network architectures. Initially, we trained the three variations with two stacked recurrent layers and without a dropout layer. Retraining the best-performing model from this initial batch with an added dropout layer (where 20% of the outputs from each recurrent layer are dropped) resulted in an increased average validation loss.

Subsequently, the three models were trained with three stacked recurrent layers. Although the addition of a dropout layer did not prove to be beneficial, it was included to manage the increased number of outputs. Performance improved across all three architectures; however, Long Short-Term Memory continued to lead.

Finally, the best model was retrained with the dropout layer removed, resulting in a successful decrease in the average validation loss once again.

Network	Number of stacked layers	Dropout	Best average validation loss	Best accuracy	Epochs before early stoppage
GRU	2	no	2.279	0.393	12
RNN	2	no	2.262	0.396	19
LSTM	2	no	2.239	0.408	15
LSTM	2	0.2	2.268	0.398	14
GRU	3	0.2	2.219	0.413	16
RNN	3	0.2	2.213	0.404	30
LSTM	3	0.2	2.172	0.422	16
LSTM	**3**	**no**	**2.138**	**0.429**	**13**

Table 1: Performances of the different network architectures

We also explored more parameter-dense architectures during the testing phase. However, despite significantly increased computational time, these architectures did not yield better performance. For instance, an LSTM with four stacked recurrent layers, 512 features in the hidden state, and a 20% dropout layer performed slightly worse than the best model obtained previously.

Figure 2 showcases the progression of both the average validation loss and accuracy throughout the training epochs of the best-performing model. While the visualization may suggest that a patience of

eight epochs is excessive, it proved crucial in other executions to prevent premature stoppage over local minima.

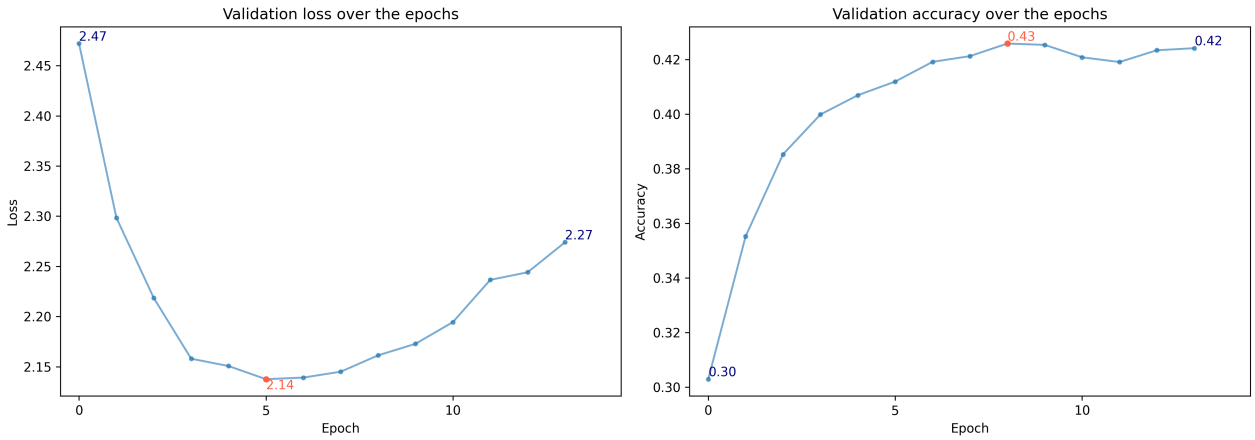

Figure 2: Evolution of the average validation loss and accuracy over the epochs before early stoppage in the LSTM model with 3 layers and no dropout

After completing the training phase, we proceed to generate text using the best model obtained. This involves randomly selecting sequences of 100 characters from the entire processed text (*prompt*), with the model generating the 101st character. The first character of the sequence is subsequently removed, and the new sequence is employed as the new input. This process continues iteratively until 100 new characters are generated.

Example of randomly selected prompt:

```
...oyalties special rules set forth in the general terms of use part of this license apply to copying a...
```

Generation of 100 characters, stopped early because of repetitions:

```
...nd the mock turtle in a mouse it was a cone of the mock turtle in a mouse it was a cone of the
```

It's evident that the generated characters don't form a random sequence; nevertheless, they lack coherence and context. The generation process often stops prematurely due to repetitive phrases. Across 20 randomly selected prompts, the generation typically stops after an average of 94 characters, primarily due to repetitive set of words, averaging around 11.

Furthermore, repeating the generation process with some of the less performing models trained earlier yields even worse results. For instance, the two-layered LSTM with dropout becomes repetitive after an average of 42 generations, primarily due to two-word repetitions. Similarly, the performance of other models is inferior, with all of them terminating prematurely within a range of 42 to 85 generations.

In a final effort to prevent the generation from becoming repetitive, we attempted to replace some of the generated characters (with a probability of 2%, equivalent to an average of 2 characters per entire generation) with a random character. However, while this prevented premature termination, the resulting phrases did not exhibit any improvement in quality, with the generations consistently biased towards the same words.

5. Conclusions:

Firstly, leveraging the rich linguistic patterns and narrative structure of *'Alice's Adventures in Wonderland'*, we pre-process the text to make it suitable for training. Despite pre-processing efforts, the generated text often lacks coherence and context, leading to premature termination due to repetitive phrases across all network configurations explored. Including more text data from diverse sources could have potentially boosted the generation performance of our models, at the expense of increased computational time.

During the training phase, we explored various network architectures, adjusting parameters like layer depth and the inclusion of dropout layers. While stacked layers generally enhanced performance, the addition of dropout layers resulted in inferior results, although with a marginal reduction in computational time. As anticipated, LSTM architectures consistently outperformed alternative models, demonstrating their effectiveness in sequence modeling practical applications.

In an attempt to mitigate repetition and improve generation quality, we experimented with replacing some generated characters with random ones. However, this approach did not yield significant improvements. Despite encountering unsatisfactory generations even with the best model, the experiment provides valuable insights into the capabilities and limitations of RNNs and gated alternatives for text generation.