SENTIMENT ANALYSIS SU DATASET IMDB

Matteo Galiazzo
Dipartimento di Informatica - Scienza e Ingegneria
Università di Bologna
matteo.galiazzo@studio.unibo.it

January 23, 2025

ABSTRACT

TODO

1 Introduction

This project presents an implementation of sentiment analysis on the IMDB movie reviews dataset using deep learning approaches. The project explores two different Long Short-Term Memory (LSTM) architectures: a standard LSTM and a bidirectional LSTM model. The implementation leverages TensorFlow and Keras for model development and training, with the goal of accurately classifying movie reviews as either positive or negative.

1.1 Recurrent Neural Networks

The problem of using a MLP or CNN to process text is the fact that we have a fixed input. This means that if we use one of this architectures we are constrained by a fixed window for our input. For solving this problem scholars introduced Recurrent Neural Networks (RNNs). In a RNN we have a sequence of words $x_1, ..., x_n$. To process this sequence we inject x_1 in the hidden layer and output o_1. The information of x_2 is processed together with the output of the previous computation. So in principle when you compute the output of x_n you take into consideration all the previous output. This way the sequence length is independent from the network structure.

Two very common memory cells for RNNs are Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU). RNNs have a problem. They can only see the past of the sequence.

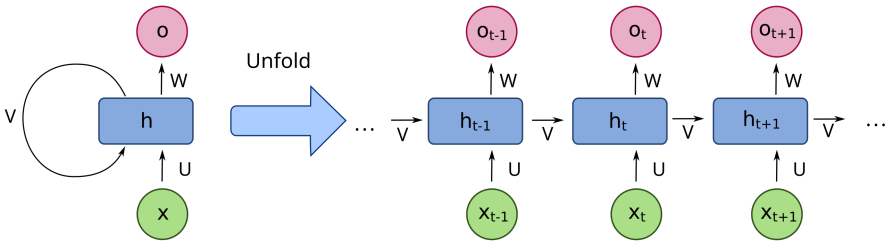


Figure 1: A compressed (left) and unfolded (right) RNN

Abstractly speaking, an RNN is a function f_θ of type $(x_t, h_t) \to (y_t, h_{t+1})$, where:

- x_t: input vector.
- h_t: hidden vector.

- y_t: output vector.
- θ: neural network parameters.

It's a neural netowrk that maps an input x_t into an output y_t, with the hidden vector h_t playing the role of "memory", a partial record of all previous input-output pairs. At each step, it transforms input to an output, and modifies its "memory" to help it to better perform future processing.

The figure 1 may be misleading to many because practical neural netowrk topologies are frequently organized in "layers" and the drawing gives that appearance. However, what appears to be layers are, in fact, different steps in time, "unfolded" to produce the appearance of layers.

1.1.1 Bidirectional RNNs

We can combine two RNNs, one processing the input sequence in one direction, and the other one in the opposite direction This is called a bidirectional RNN, and it's structured as follows:

- The forward RNN processes in one direction: $f_\theta(x_0, h_0) = (y_0, h_1), f_\theta(x_1, h_1) = (y_1, h_2), ...$
- The backward RNN processes in the opposite direction:
  $f'_{\theta'}(x_N, h'_N) = (y'_N, h'_{N-1}), f'_{\theta'}(x_{N-1}, h'_{N-1}) = (y'_{N-1}, h'_{N-2}), ...$

The two output sequences are then concatenated to give the total output: $((y_0, y'_0), (y_1, y'_1), ..., (y_N, y'_N))$. Bidirectional RNN allows the model to process a token both in the context of what came before it and what came after it.

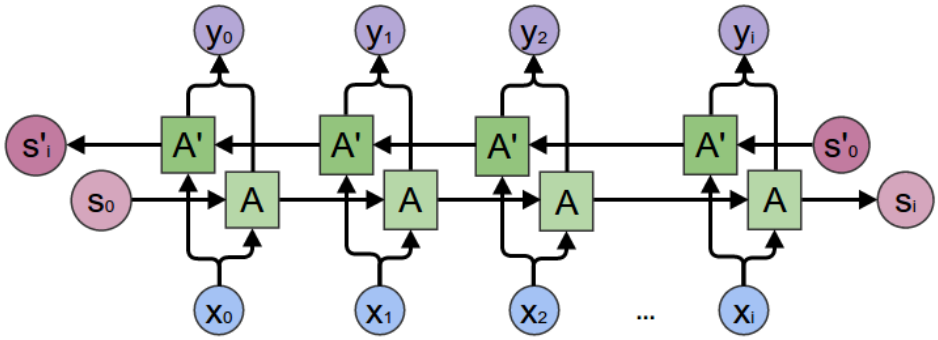


Figure 2: An unfolded bidirectional RNN

1.1.2 LSTM

While Recurrent Neural Networks (RNNs) provide a mechanism for processing sequential data by maintaining a hidden state that captures information from previous steps, they suffer from the vanishing gradient problem. This issue happens when gradients used to update the network's parameters during training diminish exponentially over time, making it difficult for the network to learn long term dependencies. To adress this, Long Short-Term Memory (LSTM) networks were introduced as a specialized variant of RNNs. LSTMs can better capture long-term dependencies by incorporating a more sophisticated memory mechanism and gating structure.

In LSTMs the module, instead of having a single neural network layer, has four parts interacting in a special way.

In the diagram in figure 3, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line fork denotes its content being copied and the copies going to different locations.

The core idea behind LSTMs The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

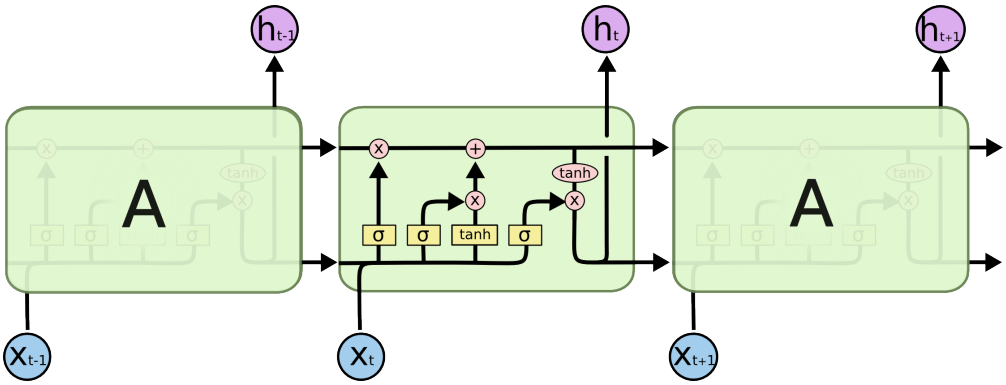
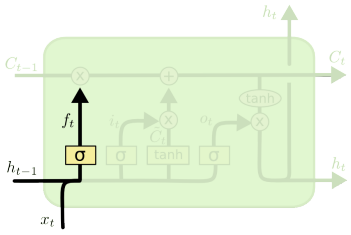


Figure 3: The repeating module in an LSTM

The LSTM does have the ability to remove or add information to the cell state, carefully regulated bu structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through", while a value of one means "let everything through". An LSTM has three of these gates, to protect and control the cell state.

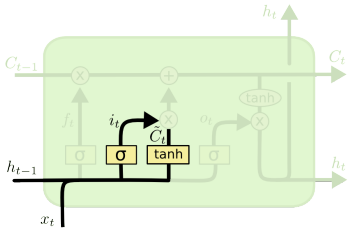
The four LSTM parts The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer". It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "remember this" while a 0 represents "forget this".



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 4: The forget module in an LSTM

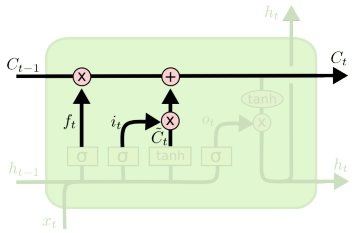
The next step is to decide what new information we're going to store in the cell state. This has two parts. First a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 5: The input module in an LSTM

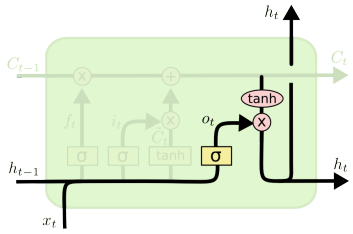
Next, we have to update the old cell state C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it. We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 6: The update module in an LSTM

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell sate we're going to output. Then, we put the cell state through tanh (to push the values between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 7: The output module in an LSTM

2 Technical implementation

2.1 Dataset

The IMDB dataset is a widely used benchmark for sentiment analysis tasks. It contains 50000 movie reviews from IMDB, allowing no more than 30 reviews per movie. The dataset contains an even number of positive and negative reviews, so randomly guessing yields 50% accuracy. Also, neutral reviews are not included in the dataset [1].

2.2 Data preprocessing

The implementation includes several preprocessing steps. The first one is data preprocessing. Data preprocessing is important because raw textual data often contains noise, suck as typographical errors, inconsistent formatting and irrelevant symbols. Preprocessing techniques aim to transform raw text data into a clean format that facilitates effective feature extraction and model training.

In this case I cleaned the dataset by removing: links, html tags, double spaces, punctuation and stopwords. Stopwords are high frequency and low information words such as articles, prepositions, and conjunctions (e.g. "the", "and", "in") that are excluded to reduce noise and improve computational efficiency. The stopwords were removed by using NLTK (Natural Language ToolKit), a suite of libraries and programs for natural language processing written in Python. I also lemmatized the dataset. Lemmatization is the process of reducing words to their root form, known as a lemma. Unlike stemming, which simply chops off word endings, lemmatization considers the context and part of speech of a word to return a valid dictionary form (e.g. "running" is "run" and "better" is "good"). The entire dataset was also converted to lowercase.

We can see the dataset's head in the following code snippet:

- Before the preprocessing:

```
1    0      I rented I AM CURIOUS-YELLOW from my video sto...      0
2    1      "I Am Curious: Yellow" is a risible and preten...      0
3    2      If only to avoid making this type of film in t...      0
4    3      This film was probably inspired by Godard's Ma...      0
5    4      Oh, brother...after hearing about this ridicul...      0
```

```
...                                                    ...    ...
49995  Just got around to seeing Monster Man yesterda...      1
49996  I got this as part of a competition prize. I w...      1
49997  I got Monster Man in a box set of three films ...      1
49998  Five minutes in, i started to feel how naff th...      1
49999  I caught this movie on the Sci-Fi channel rece...      1
```

- After the preprocessing:

```
0      rented curious yellow video store controversy ...      0
1      curious yellow risible pretentious steaming pi...      0
2      avoid making type film future film interesting...      0
3      film probably inspired godard masculin feminin...      0
4      oh brother hearing ridiculous film umpteen yea...      0
...                                                    ...    ...
49995  got around seeing monster man yesterday long w...      1
49996  got part competition prize watched really expe...      1
49997  got monster man box set three film mainly want...      1
49998  five minute started feel naff looking got comp...      1
49999  caught movie sci fi channel recently actually ...      1
```

2.3 Model architectures

References

[1] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.