SENTIMENT ANALYSIS SU DATASET IMDB

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

January 29, 2025

ABSTRACT

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.

Contents

1 Introduction **2**
1.1 Recurrent Neural Networks 2
1.1.1 Bidirectional RNNs 2
1.1.2 LSTM 2

2 Technical implementation **4**
2.1 Dataset 4
2.2 Data preprocessing 5
2.3 Model architectures 6
2.3.1 LSTM model 6
2.3.2 Bidirectional LSTM model 6
2.4 Hyperparameters selection 6
2.5 Model training 7

3 Results and analysis **7**
3.1 Training results 7
3.1.1 LSTM 7
3.1.2 Bidirectional LSTM 8

1 Introduction

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.

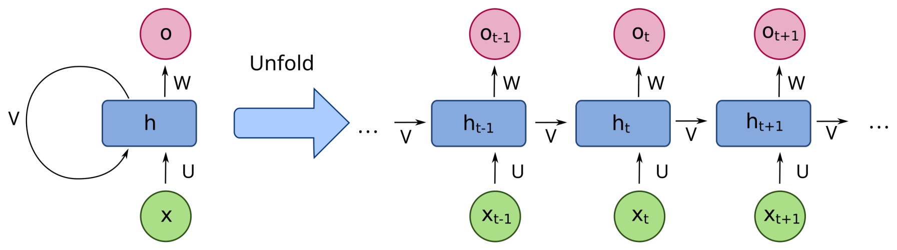


Figure 1: A compressed (left) and unfolded (right) RNN

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.

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

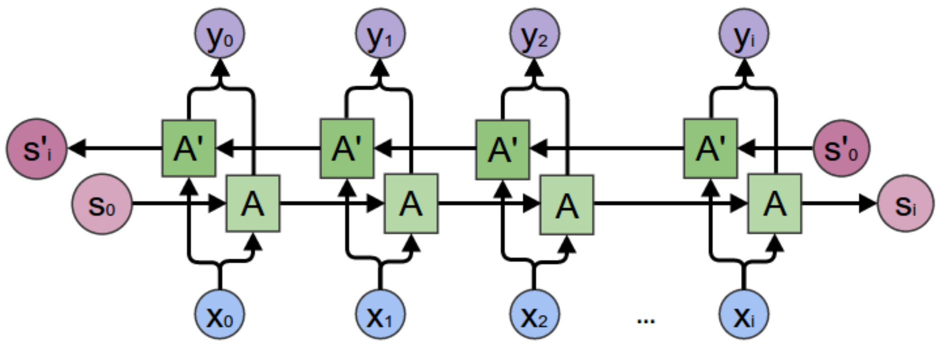


Figure 2: An unfolded bidirectional RNN

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.

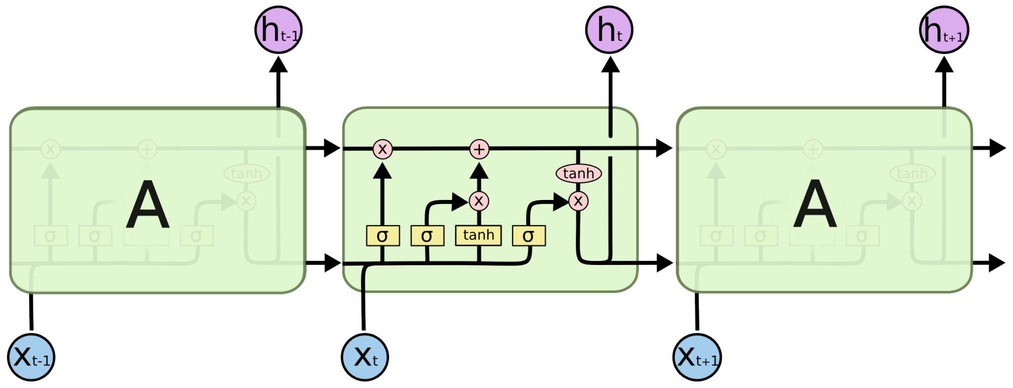


Figure 3: The repeating module in an LSTM

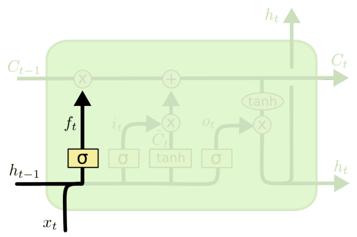
In the diagram in figure 3, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line fork denotes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

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.

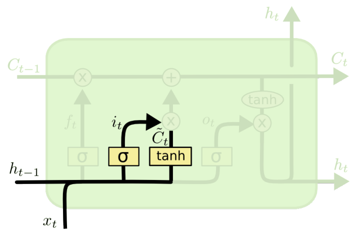
The four LSTM parts The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer". It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.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

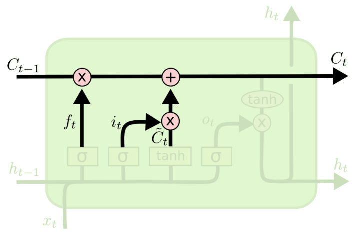
The next step is to decide what new information we're going to store in the cell state. This has two parts. First a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.



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

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.



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

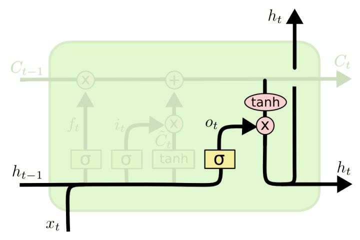
Figure 6: The update module in an LSTM

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.

2 Technical implementation

2.1 Dataset

IMDB is a popular website that collects movie reviews. The IMDB dataset is a widely used benchmark for sentiment analysis tasks; it contains 50000 movie reviews from the website, 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. Only highly polarized reviews are considered. A negative review has a score ≤ 4 out of 10, and a positive review has a score ≥ 7 out of 10. Neutral reviews are not included in the dataset. [1].



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

Figure 7: The output module in an LSTM

2.2 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:

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

2.3.1 LSTM model

The proposed model employs a neural network architecture with three core layers, combining embedding, recurrent processing, and final classification. We will now see the model and go through each one of its layers:

```
model = Sequential()
model.add(Embedding(input_dim=TOKENIZER_MAX_WORDS, output_dim=EMBED_DIM))
model.add(LSTM(LSTM_OUT, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation="sigmoid"))
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```

Embedding layer This layer's purpose is to convert integer-encoded word indices into dense vector representations. The layer learns semantic relationships between words during training. Instead of having a long, sparse vector, each word is mapped to a dense vector of fixed size. These vectors are not binary, they contain real numbers that the model learns during training. Embeddings are powerful because they use vector spaces, so they can capture intricate relationships between words. For example, in a well trained embedding space, the vector for "king" minus "man" plus "woman" will point to queen. Embeddings provide a way for the models to understand and relate to the data in a more sophisticated way compared to, for example, one hot encoding [2].

LSTM layer We have already seen in 1.1.2 how an LSTM works. We know that the LSTM layer processes sequential information while maintaining memory of long-range dependencies. We add the LSTM layer in the network together with a dropout and recurrent dropout. Dropout is a regularization technique (a process that converts the answer of a problem to a simpler one) for reducing overfitting in artificial neural networks by preventing complex co-adaptations on training data. Dropout refers to randomly "dropping out", or omitting, units (both hidden and visible) during the training process of a neural network. Dropout is the equivalent of training several smaller networks on the same task. The final model is like an ensemble of smaller networks, reducing variance and providing more robust predictions. Dropout applies dropout to the non-recurrent connections of the LSTM layer. Recurrent dropout applies dropout to the recurrent connections (hidden statet transitions) within the LSTM [3].

Dense classification layer This layer serves as the final classification step. The dense layer is a fully connected layer that takes the output from the LSTM and maps it to a single scalar value. The LSTM typically returns a sequence of hidden states or a final hidden state. This dense layer "summarizes" that information into a single value. The sigmoid activation function squashes the output into the range $[0, 1]$, interpreting it as a probability of the sentiment.

Model compilation The model compilation step configures the model for training by defining how it learns (the optimizer), what to optimize for (loss function) and how to evaluate the performance (evaluation metric). The optimizer is Adam (Adaptive Moment Estimation), an advanced gradient descent alogrithm that adapts learning rates during training. It combines the benefits of momentum (accelerates updates in consistent gradient directions) and adaptive leraning rates (adjusts rates for each parameter individually). The loss function is the binary cross-entropy. The purpose of the loss function is to measure the difference between predicted probabilities and true labels. It quantifies how well the model performs. The evaluation metric is the accuracy (the percentage of correct predictions), I didn't choose another metric like F1-score because the dataset is balanced.

2.3.2 Bidirectional LSTM model

The bidirectional LSTM model is basically identical to the LSTM network, but it adds a bidirectional layer. The main difference is that while the standard LSTM processes the sequence in one direction, so it can only capture the past context, while the bidirectional LSTM processes the sequence in both directions, so it can capture both past and future context. The bidirectional LSTM is more complex, with roughly double the parameters due to the two LSTMs.

```
model = Sequential()
model.add(Embedding(input_dim=TOKENIZER_MAX_WORDS, output_dim=EMBED_DIM))
model.add(Bidirectional(LSTM(LSTM_OUT, dropout=0.2, recurrent_dropout=0.2)))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

2.4 Hyperparameters selection

The network has both fixed and experimented hyperparameters. The fixed ones are:

- `VALIDATION_SPLIT = 0.2`: a small portion (20% of the training data) is sufficient to evaluate the model's generalization performance without significantly reducing the training set size.
- `TOKENIZER_MAX_WORDS = 10000`: Since adults that are native english speakers tend to have a vocabulary of 15000 to 30000 words I expect to be able to extract the positive/negative sentiment with two thirds of the tipical vocabulary [4].

The hyperparameters where I experimented are:

- `BATCH_SIZE`: the batch size is variated from 64 to 128. The batch size defines the number of samples that will be propagated thorugh the network. Using a batch size that is smaller than the number of the samples is useful because it requires less memory since you train the network using fewer samples, and because typically the networks train faster with mini-batches. That's because we update the weights after each propagation [5].
- `REVIEW_MAX_LENGTH`: the maximum review length is variated from 250 to 500 tokens. A longer sequence allows the model to analyze broader textual patterns. Longer sequences require more memory but reduce the truncation of input text. I chose to test both to determine if longer sequences improve sentiment detection. If 500 improves accuracy, longer sequences are better, if 250 performs similarly, it saves computational resources.

This settings challenge the common absumption in deep learning that "bigger is better", that means that a bigger model will yield better results.

2.5 Model training

The model training starts with k-fold cross validation. Cross validation is a resampling technique used to evaluate machine learning models. It provides a statistically more reliable estimation of the performance than a single training/test split. This happens because if we do a single training we could get a particularly easy/hard test set, and so we would not come up with the best estimate possible of the models ability to learn and predict.

We want to use all the data. So if we use an 80/20 split we are going to do 5-fold cross validation by training the model 5 times on 80% of the data and testing it on the remaining 20%. We ensure that each data point ends up in the 20% test set exactly once. We've therefore used every data point we have to contribute to an understanding of how well our model performs the task of learning from some data and predicting some new data [6].

During model training, since I chose 5 trainng epochs as an arbitrary number, I also used ModelCheckpoint. This is done or several reasons, in fact using a model checkpoint:

- Prevents loss of progress since it saves the model at specified intervals. In fact if training crashes or is interrupted, it can be resumed from the last saved checkpoint instead of starting over.
- Saves the "best" model automatically since it only saves the model when the validation loss improves. This is a good thing because it avoids saving suboptimal models, and saves disk space since it overwrites previous checkpoints if the current model has a better validation loss than the last one.
- Mitigates overfitting since by monitoring validation loss even if the model starts to overfit after 3 epochs it won't save the final model, that would have degraded performance compared to the one which stopped at 3 epochs.

This has been done both for the LSTM and for the bidirectional LSTM.

3 Results and analysis

3.1 Training results

3.1.1 LSTM

As we already pointed out in 2.4, the LSTM has been trained two times: one with batch size of 64 and maximum review length of 250 and one with batch size of 128 and maximum review length of 500. These are the training results for the different models and folds:

Fold	Accuracy	Precision	Recall	F1-score
Fold 1	0.8710	0.8782	0.8710	0.8704
Fold 2	0.8667	0.8698	0.8667	0.8664
Fold 3	0.8680	0.8694	0.8680	0.8679
Fold 4	0.8603	0.8660	0.8603	0.8598
Fold 5	0.8650	0.8652	0.8650	0.8650
Mean ($\pm$ std)	0.866 $\pm$ 0.00396	0.870 $\pm$ 0.00515	0.866 $\pm$ 0.00396	0.866 $\pm$ 0.00395

Table 1: LSTM with batch size 64 and review length 250

Fold	Accuracy	Precision	Recall	F1-score
Fold 1	0.8621	0.8721	0.8621	0.8612
Fold 2	0.8484	0.8531	0.8484	0.8479
Fold 3	0.8540	0.8641	0.8540	0.8530
Fold 4	0.8673	0.8685	0.8673	0.8672
Fold 5	0.8701	0.8701	0.8701	0.8701
Mean ($\pm$ std)	0.860 $\pm$ 0.00908	0.866 $\pm$ 0.00757	0.860 $\pm$ 0.00908	0.860 $\pm$ 0.00936

Table 2: LSTM with batch size 128 and review length 500

3.1.2 Bidirectional LSTM

The bidirectional LSTM has also been trained in the same two configurations as the standard LSTM: one with batch size of 64 and maximum review length of 250 and one with batch size of 128 and maximum review length of 500. These are the training results for the different models and folds:

Fold	Accuracy	Precision	Recall	F1-score
Fold 1	0.8729	0.8760	0.8729	0.8726
Fold 2	0.8725	0.8744	0.8725	0.8723
Fold 3	0.8637	0.8637	0.8637	0.8637
Fold 4	0.8589	0.8590	0.8589	0.8589
Fold 5	0.8762	0.8763	0.8762	0.8762
Mean ($\pm$ std)	0.869 $\pm$ 0.00723	0.870 $\pm$ 0.00799	0.869 $\pm$ 0.00723	0.869 $\pm$ 0.00717

Table 3: Bidirectional LSTM with batch size 64 and review length 250

Fold	Accuracy	Precision	Recall	F1-score
Fold 1	0.8746	0.8780	0.8746	0.8743
Fold 2	0.8465	0.8482	0.8465	0.8463
Fold 3	0.8675	0.8700	0.8675	0.8673
Fold 4	0.8508	0.8629	0.8508	0.8495
Fold 5	0.8728	0.8728	0.8728	0.8728
Mean ($\pm$ std)	0.862 $\pm$ 0.01295	0.866 $\pm$ 0.01153	0.862 $\pm$ 0.01295	0.862 $\pm$ 0.01322

Table 4: Bidirectional LSTM with batch size 128 and review length 500

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.

[2] Amit Yadav. How Does Embedding Layer Work? — medium.com. https://medium.com/biased-algorithms/how-does-embedding-layer-work-eafb1c721302. [Accessed 28-01-2025].

[3] Dilution (neural networks) - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Dilution_(neural_networks). [Accessed 28-01-2025].

[4] Lesson Nine GmbH. Does Your Vocabulary Size Matter? — babbel.com. https://www.babbel.com/en/magazine/does-vocabulary-size-matter. [Accessed 28-01-2025].

[5] What is batch size in neural network? — stats.stackexchange.com. https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network. [Accessed 29-01-2025].

[6] How to choose a predictive model after k-fold cross-validation? — stats.stackexchange.com. https://stats.stackexchange.com/questions/52274/how-to-choose-a-predictive-model-after-k-fold-cross-validation. [Accessed 29-01-2025].