Recurrent neural networks for newsgroup documents classification

Angel Lopez Manriquez **Jonas Rudi**

Abstract

Recurrent neural networks have been one of the most extensively applied methods to text processing. In comparison to bag-of-words approaches they are able to exploit the sequential information encoded in the text. We implemented Recurrent Neural Networks with Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) Layers that classify documents from the ”20 newsgroups” dataset. We describe our architecture, the methods we applied to preprocess the data and discuss our results in comparison to different benchmarks. Our best model uses LSTM cells

In short, what was done was:

- Expressing the texts as tensors, removing useless words.
- Use of word embedding.
- Define the architecture of a recurrent neural network.
- Validate the model.

1 Description of the problem

1.1 Multinomial Classification

In machine learning, multi-class or multinomial classification is the problem of classifying instances into one of three or more classes, as opposed to binary classification, where an instance can only be classified into one of two classes.

While some classification algorithms naturally permit the use of more than two classes, others are by nature binary algorithms; these can, however, be turned into multinomial classifiers by a variety of strategies.

Multi-class classification should not be confused with multi-label classification, where multiple labels are to be predicted for each instance.

1.2 The 20 Newsgroups data set

The dataset we used to train our model is called ”20 Newsgroups”. It consists of approximately $20,000$ documents with a structure similar to that of an email. They can have headers, footers, quotes and additional information next to the content itself[1]. A newsgroup is a type of forum within the Usenet, a precursor to the World Wide Web[2]. Every document resembles a post in a certain newsgroup, which we use as its class. There are 20 different groups (1.2), which all occur around $1,000$ times. Some of the groups are closely related, like ”rec.autos” and ”rec.motorcycles”, while others have not much in common, such as ”comp.graphics” and ”alt.atheism”.
The data was originally gathered by Ken Lang and can be downloaded from many sources[3]. For the sake of simplicity, we used the SciKit-Learn library[4] do directly retrieve the dataset split into a train and a test set with quotes from previous replies already removed.

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.scrypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Table 1: List of classes in the dataset

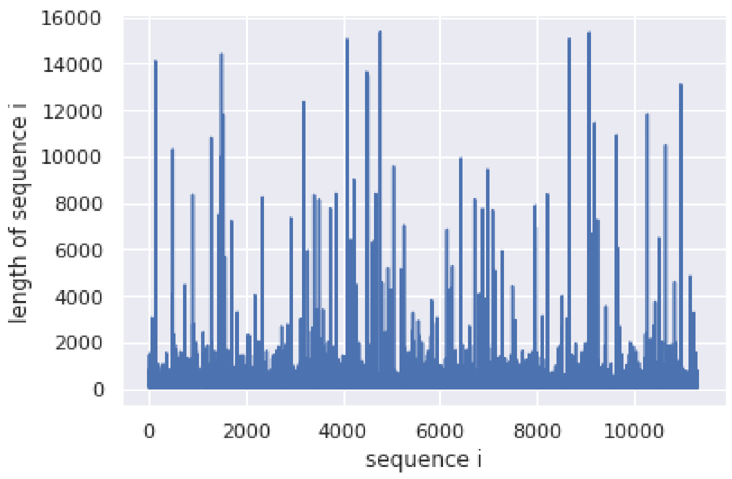
2 Description of our approach

We organized the implementation of the project as follows:

2.1 Preprocessing

As usual, we need to transform inputs to numerical tensors. To do so, we tokenize each string, then we express the tokens as sequences. After this transformation we loss some information about data such as punctuation, sentences surrounded by parenthesis, typos and so one, it is fine.

Even though recurrent neural network does not require a fixed we length it is a good idea to fix it for performances purposes. To figure out the suitable length of the sequence we compute the mean μ and the standard deviation σ from the length of each sequence. We realize $\mu = 302.52$ but $\sigma = 723.12$, which indicate us setting the length as μ might not be a good idea, for speed purposes we fix the sequence to 100.



We keep the most common 100 words from the sequence. We use gloval vectors for word representation embeddings, for each word we obtain its representation as 100 vector. This save time to train the model and improve accuracy.

2.2 Architecture of the model

For comparison purposes we tried to solve the multinomial classification task with four models, for the fist one we use a LSTM, the second one a GRU. The remaining ones are as the previous but using recurrent dropout. There are few differences with these ones but the most important is the time to fit the model, as reviewed in lectures, GRU is faster to learn than LSTM with similar results.

3 Implementation

As usual, we used the python programming language to implement the tasks. We used the `fetch_20newsgroups` function from the scikit learn library module to obtain the dataset, in fact, we have the posibility to remove the headers, footers and so one, but again, we had to do it from scratch for educational purposes.

```python
from sklearn.datasets import fetch_20newsgroups


twenty_train = fetch_20newsgroups(subset='train', shuffle=True)  # obtain
↪  train dataset
target_names = twenty_train.target_names  # class label name
texts = twenty_train.data  # list: strings, each element is an email
target = twenty_train.target  # list: int, each integer associated to
↪  target_names
```

For preprocessing the data we used the `Tokenizer` and `pad_sequence` from `Keras`, as we discussed before, we tokenize the texts and fix the length of the sequence. Then we used GloVe[5](gloval vectors for word representation) in order to build our *embedding matrix*.

```python
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequence


def format_data(texts, target, vocab_size=20000, max_length=100):
    tokenizer = Tokenizer(num_words=vocab_size) # Setup tokenize
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts) # Generate sequences
    data = pad_sequences(sequences, maxlen=max_length)
    labels = to_categorical(np.asarray(target))
    return data, labels
```

Once we obtained the embeeding matrix we define the architecture of the model using the high level library **Keras**. We used the `Sequential` model as usual and the layers `LSTM, Dense, Activation, Embedding and GRU`[1].

```python
from keras.models import Sequential
from keras.layers import LSTM, Dense, Activation, Embedding, GRU


def Model(rnn_layer='gru', recurrent_dropout=0.0, nclasses=20):
  ''' Define the model. According to rnn_layer parameter we define a
  ↪  disntinct rnn layer.'''
  model = Sequential()  # instantiate the classic sequential model
  model.add(Embedding(vocab_size,    # vocabulary size
                      embedding_dim,    # embedding dimension
                      input_length=max_length,
                      weights=[embedding_matrix],    # pass the weights
                      ↪  obtained previously from the Glove File
                      trainable=False))  # we don't need to train it
                      ↪  again
  if rnn_layer == 'gru': model.add(GRU(128,
  ↪  recurrent_dropout=recurrent_dropout))
  else: model.add(LSTM(128, recurrent_dropout=recurrent_dropout))
  model.add(Dense(nclasses))
  model.add(Activation('softmax'))
  return model  # return the model if needed
```

[1]For more details in the implementation you could visit the notebook https://github.com/Ang3lino/mlnn/blob/master/projects/3/20newsFinalBoss.ipynb

4 Results

For validation we plot the accuracy at each epoch, letting us know the model is improving significantly at each time 2. After we use the confusion matrix. For example let us consider the GRU with no recurrent dropout as an example.

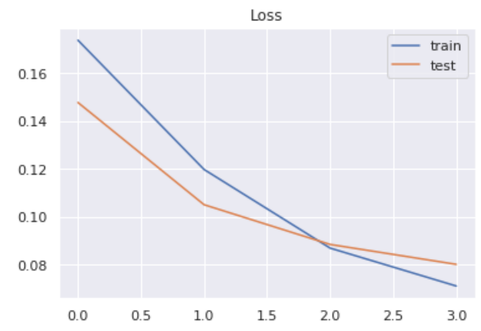


Figure 1: Loss value of our best model for each epoch

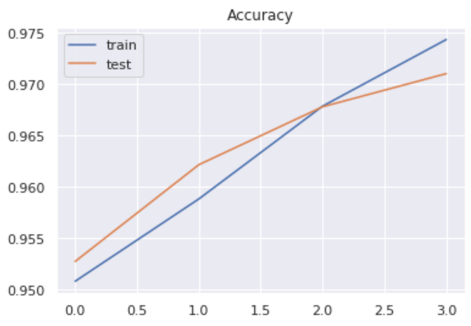
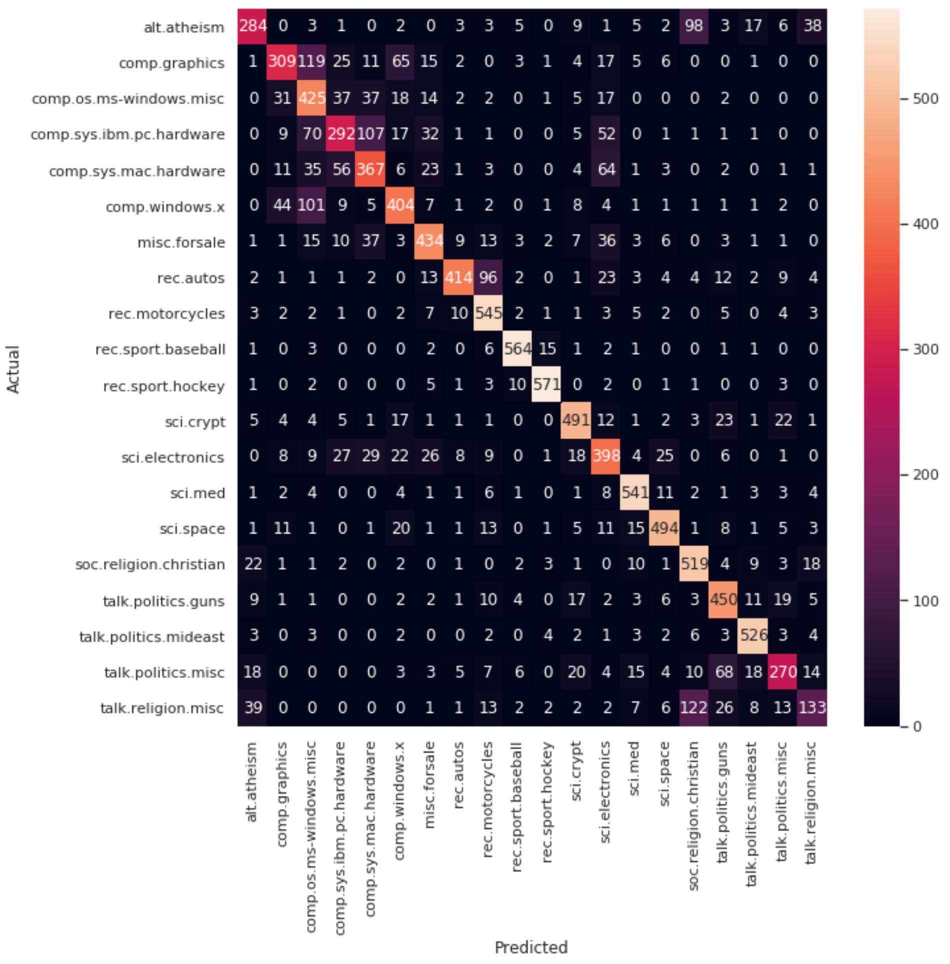


Figure 2: Accuracy value of our best model for each epoch



	alt.atheism	comp.graphics	comp.os.ms-windows.misc	comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	comp.windows.x	misc.forsale	rec.autos	rec.motorcycles	rec.sport.baseball	rec.sport.hockey	sci.crypt	sci.electronics	sci.med	sci.space	soc.religion.christian	talk.politics.guns	talk.politics.mideast	talk.politics.misc	talk.religion.misc
alt.atheism	284	0	3	1	0	2	0	3	3	5	0	9	1	5	2	98	3	17	6	38
comp.graphics	1	309	119	25	11	65	15	2	0	3	1	4	17	5	6	0	0	1	0	0
comp.os.ms-windows.misc	0	31	425	37	37	18	14	2	2	0	1	5	17	0	0	0	2	0	0	0
comp.sys.ibm.pc.hardware	0	9	70	292	107	17	32	1	1	0	0	5	52	0	1	1	1	1	0	0
comp.sys.mac.hardware	0	11	35	56	367	6	23	1	3	0	0	4	64	1	3	0	2	0	1	1
comp.windows.x	0	44	101	9	5	404	7	1	2	0	1	8	4	1	1	1	1	1	2	0
misc.forsale	1	1	15	10	37	3	434	9	13	3	2	7	36	3	6	0	3	1	1	0
rec.autos	2	1	1	1	2	0	13	414	96	2	0	1	23	3	4	4	12	2	9	4
rec.motorcycles	3	2	2	1	0	2	7	10	545	2	1	1	3	5	2	0	5	0	4	3
rec.sport.baseball	1	0	3	0	0	0	2	0	6	564	15	1	2	1	0	0	1	1	0	0
rec.sport.hockey	1	0	2	0	0	0	5	1	3	10	571	0	2	0	1	1	0	0	3	0
sci.crypt	5	4	4	5	1	17	1	1	1	0	0	491	12	1	2	3	23	1	22	1
sci.electronics	0	8	9	27	29	22	26	8	9	0	1	18	398	4	25	0	6	0	1	0
sci.med	1	2	4	0	0	4	1	1	6	1	0	1	8	541	11	2	1	3	3	4
sci.space	1	11	1	0	1	20	1	1	13	0	1	5	11	15	494	1	8	1	5	3
soc.religion.christian	22	1	1	2	0	2	0	1	0	2	3	1	0	10	1	519	4	9	3	18
talk.politics.guns	9	1	1	0	0	2	2	1	10	4	0	17	2	3	6	3	450	11	19	5
talk.politics.mideast	3	0	3	0	0	2	0	0	2	0	4	2	1	3	2	6	3	526	3	4
talk.politics.misc	18	0	0	0	0	3	3	5	7	6	0	20	4	15	4	10	68	18	270	14
talk.religion.misc	39	0	0	0	0	0	1	1	13	2	2	2	2	7	6	122	26	8	13	133

Figure 3: Confusion Matrix for our best model and the test data

Model/ Dropout	LSTM	GRU
0.0	0.944	0.939
0.2	0.940	0.936

Table 2: Accuracy Scores for all tested models

Model/ Dropout	LSTM	GRU
0.0	0.623	0.741
0.2	0.618	0.713

Table 3: F1-Scores for all tested models

Model/ Dropout	LSTM	GRU
0.0	153.44	115.31
0.2	149.35	118.1

Table 4: Time to train the model in seconds with four epochs

Before analyzing our result, we tried to find out, what other people achieved without an RNN architecture. We found accuracy values of 83.3% using Naive-Bayes and 91.6% using Support Vector Machines. If we take these results as our baseline, we have achieved a significantly better result, with our top model reaching an accuracy of 0.944% on the test data. At the same time, our model did not take very long to train, or at least not much longer than a non-RNN apporach would have taken.

5 Conclusions

We have successfully used Recurrent neural networks with word embeddings for the task of multi-class text classification for the ”20 newsgroups” data set. Also, we reviewed, how using RNN is one of the most effective techniques to solving problems where the order of the data is relevant. It is important to emphasize the significance of improving the current paradigms. For example, this problem could have hardly be solved with a simple recurrent neural network since the sequence of the model, as we watched in the plot, could be high, having the problem of the vanishing gradient problem[6].

References

[1] Wikipedia. Usenet newsgroup — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Usenet%20newsgroup&oldid=919862919, 2019. [Online; accessed 15-December-2019].

[2] Usenet.com. What is Usenet? https://www.usenet.com/what-is-usenet/, 2019. [Online; accessed 15-December-2019].

[3] Ken Lang. Newsweeder: Learning to filter netnews. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=1734CA431737ECA95C2419DF4BE29276?doi=10.1.1.22.6286&rep=rep1&type=pdf, 2010.

[4] 20 newsgroups dataset from scikit-learn. https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[6] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.