Report

ILLINOIS INSTITUTE OF TECHNOLOGY

Performance of NN on Sequential Tasks

Feed-Forward vs LTSM vs Transformers Networks

CS 584 - Machine Learning: final project

Student: Paul Legout

Student ID: A20522029

Student: Tristan Leduc

Student ID: A20517874

Fall 2022

2022-11-24

Abstract

RNN are commonly used to analyze sequential data. Analyzing this type of data requires consideration of the order of elements. To that extent, RNN have the ability to memorize small chunks of data and use it to predict the next item in line or to classify time series signals such as electrocardiograms in the medical field. Not only that, but this memory also makes this kind of network suitable for the analysis of language (NLP), in which the analysis of sequences provides information about semantic and context. LSTM are among the most used of this type, but are prone to problems such as vanishing gradients, slow training and they struggle with long sequences. Transformers bring the concept of "Attention" to prevent this kind of problem by focusing on the relevant part. We will see the efficiency of this mechanism.

Keywords: Recurrent neural networks, sentiment classification, weather forecasting

Contents

1. Introduction

1.1 Research questions/hypotheses

This project will solve two different tasks, using several types of neural network models, in order to compare them. The first experiment will be a weather forecasting task, and the second will be a sentiment classification task. We want to see the improvements that LSTMs bring over standard feed-forward models. Then we will see if the transformers, in particular the attention mechanism, can yield even better results.

2. Weather forecasting

2.1 Problem statement

The goal is to predict the hourly temperature, for one or more time period, using several neural networks.

2.2 Implementation details

2.2.1 Data used

The data used for this experiment is the Jena climate dataset, from the Max Planck Institute for Biogeochemistry. It contains more than 420,550 samples and 15 features, such as:

- temperature
- pressure
- wind speed and direction
- humidity
- date and time

70% of the data is used for training, 20% for validation and 10% for testing.

2.2.2 Data pre-processing

Before diving into the implementation of the networks, some pre-processing is required.

First, the wind direction is given in degrees. However, angles do not make a good model input. Indeed, 0° and 360° should be close to each other, but the model will not get that. Moreover, the wind speed and wind direction are decoupled, but the wind direction should not matter if the wind is not blowing. Thus, we will convert the wind direction / velocity into a vector wind by projecting into a two dimensional Cartesian coordinate system. The conversion is as follows:

$$w_x = w_v * cos(w_d) \tag{1}$$
$$w_y = w_v * sin(w_d) \tag{2}$$

where w_d is the wind direction, in radian, w_v is the wind speed,
and w_x, w_y are the coordinates of the wind, with respect to x and y.
The wind speed and direction features are then replaced by w_x and w_y in the dataset, and given as input to the models.

Then, the date and time is a string, so we have to convert it to seconds. Since converting date to seconds returns the elapsed time between the given datetime and

the epoch time (00:00:00 UTC on 1 January 1970), it is always increasing. For the model to make sense of this variable, we have to modify it, to display the periodicity (per day, per year). Considering the current date and time, the conversion is as follows:

$$day = 24 * 60 * 60 \tag{3}$$
$$year = (365.2425) * day \tag{4}$$
$$day_{sin} = sin(\frac{2\pi}{day}) \tag{5}$$
$$day_{cos} = cos(\frac{2\pi}{day}) \tag{6}$$
$$year_{sin} = sin(\frac{2\pi}{year}) \tag{7}$$
$$year_{cos} = cos(\frac{2\pi}{year}) \tag{8}$$
$$\tag{9}$$

Then, the datetime feature is replaced by these four and given as input to the models.

2.2.3 Windowing

In order to feed the model with input / output for training, a windowing class must be implemented. From the raw data, it creates a window of n inputs and m targets as illustrated 1

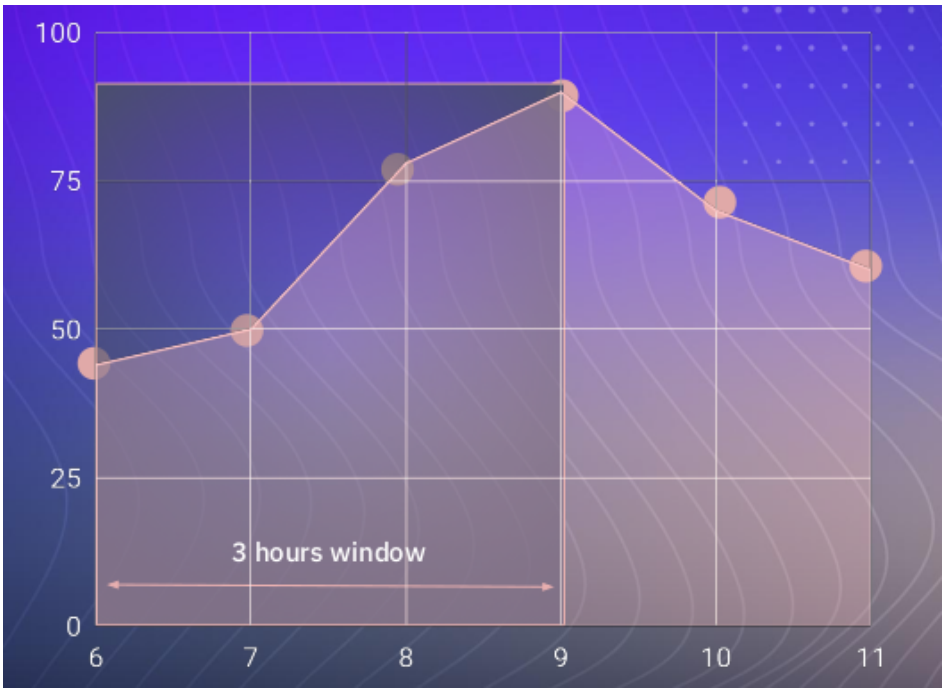

Figure 1: Time series windowing

2.2.4 Model creation

To fairly compare the different models, they have approximately the same number of trainable weights. The last Dense layer takes n units, where n is the number of target values. Here, we can see the prediction is 1 hour ahead. The models are the same for multi outputs, solely the number of units of the last layer changes.

The Adam optimizer is used for the three models, with default parameters.
The ReLu activation function is used for each layer, except the output layer and the LSTM layers, which use the default activation function (tanh).

Feed-Forward

Layer (type)	Output Shape	Param #
flatten_5 (Flatten)	(None, 95)	0
dense_32 (Dense)	(None, 32)	3072
dense_33 (Dense)	(None, 32)	1056
dense_34 (Dense)	(None, 1)	33
reshape_7 (Reshape)	(None, 1, 1)	0

Total params: 4,161
Trainable params: 4,161
Non-trainable params: 0

Figure 2: Feed forward architecture

LSTM

Layer (type)	Output Shape	Param #
lstm_73 (LSTM)	(None, 5, 32)	6656
lstm_74 (LSTM)	(None, 2)	280
dense_459 (Dense)	(None, 1)	3
reshape_115 (Reshape)	(None, 1, 1)	0

Total params: 6,939
Trainable params: 6,939
Non-trainable params: 0

Figure 3: LSTM architecture

Transformer

Layer (type)	Output Shape	Param #
input_219 (InputLayer)	[(None, 5, 19)]	0
transformer_9 (Transformer)	(None, 5, 19)	3851
flatten_102 (Flatten)	(None, 95)	0
dense_505 (Dense)	(None, 32)	3072
dense_506 (Dense)	(None, 1)	33
reshape_125 (Reshape)	(None, 1, 1)	0

Total params: 6,956
Trainable params: 6,956
Non-trainable params: 0

Figure 4: Transformer architecture

The implemented Transformer contains 1 encoder block and 1 decoder block. It is possible to stack multiple encoder / decoder blocks into one encoder / decoder for more complex architectures.

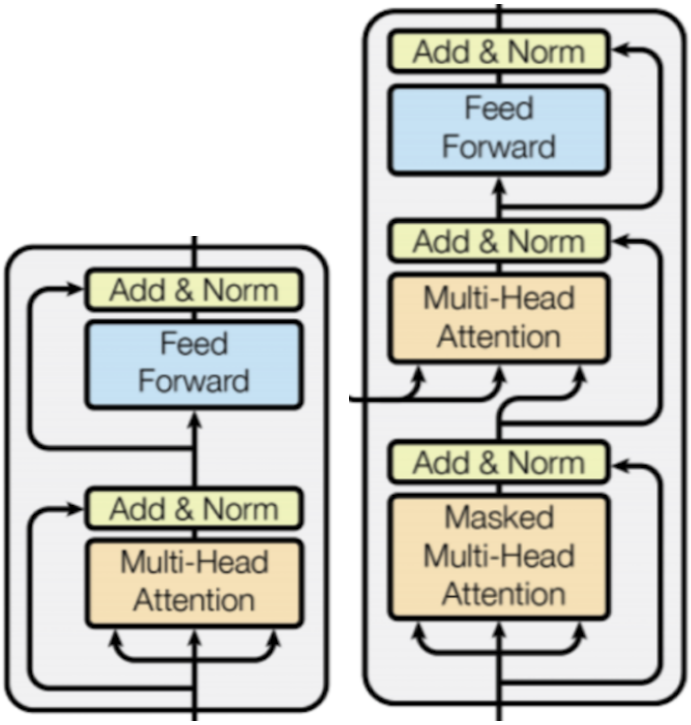

Figure 5: Encoder block (left), Decoder block (Right)

A ”classic” Transformer, as defined in the paper ”Attention is all you need”, is a sequence to sequence model, which means the encoder takes the input and creates

latent variables. The decoder takes the target, shifted by n steps as input, and the latent variables from the encoder as context. The decoder learns to predict the target, based on the target shifted by n and the output (state) of the encoder.

The implemented Transformer here, is not a sequence to sequence model. The encoder is the same, however the decoder takes the output of the encoder as input, and the input of the encoder as context.

2.3 Results and discussion

The models were first evaluated with a window of 5 inputs, to predict 1 hour ahead. Then, a new window with 15 inputs, to predict 5 hours ahead.

2.3.1 Window 5 inputs, 1 output

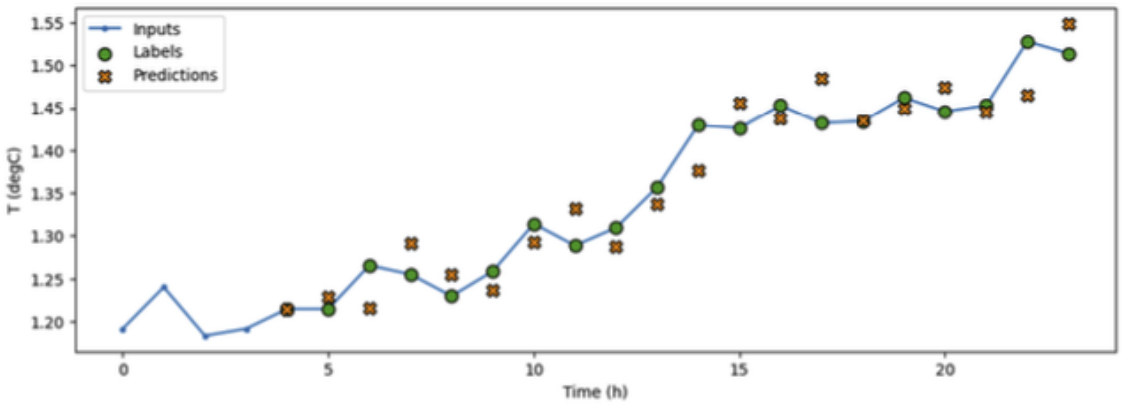

Figure 6: Feed-Forward

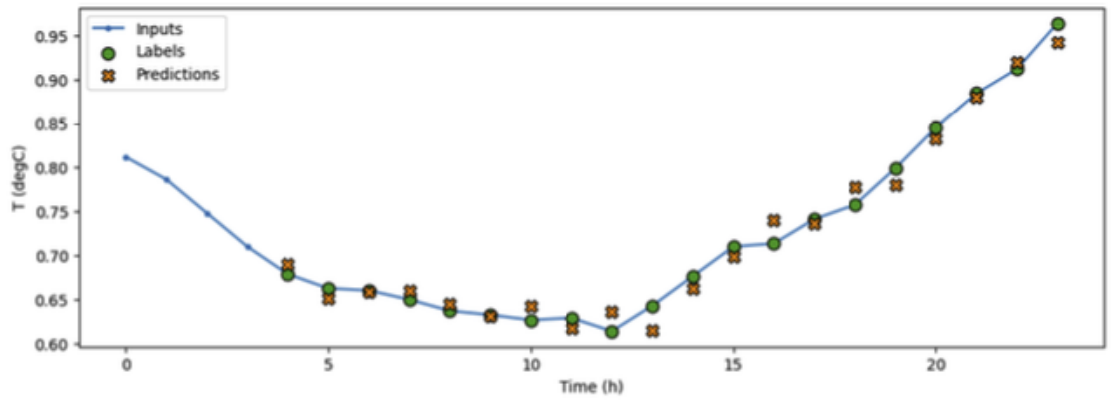

Figure 7: LSTM

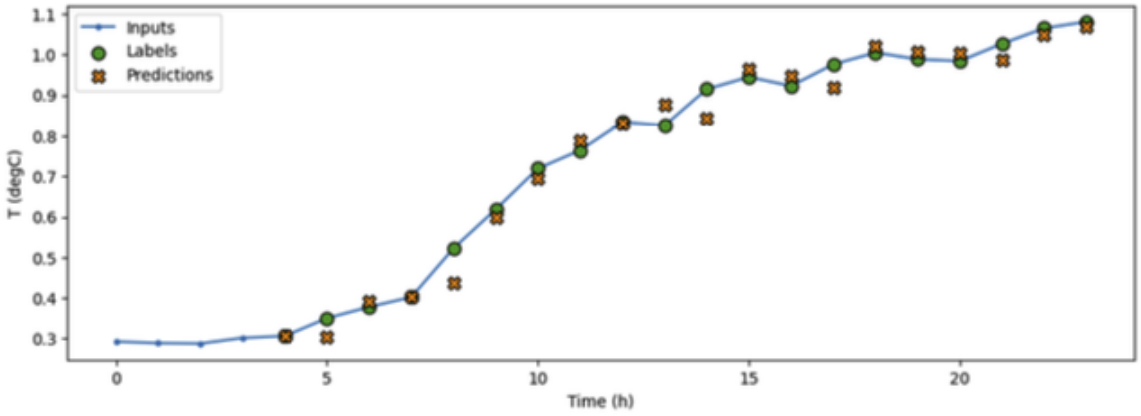

Figure 8: Transformer

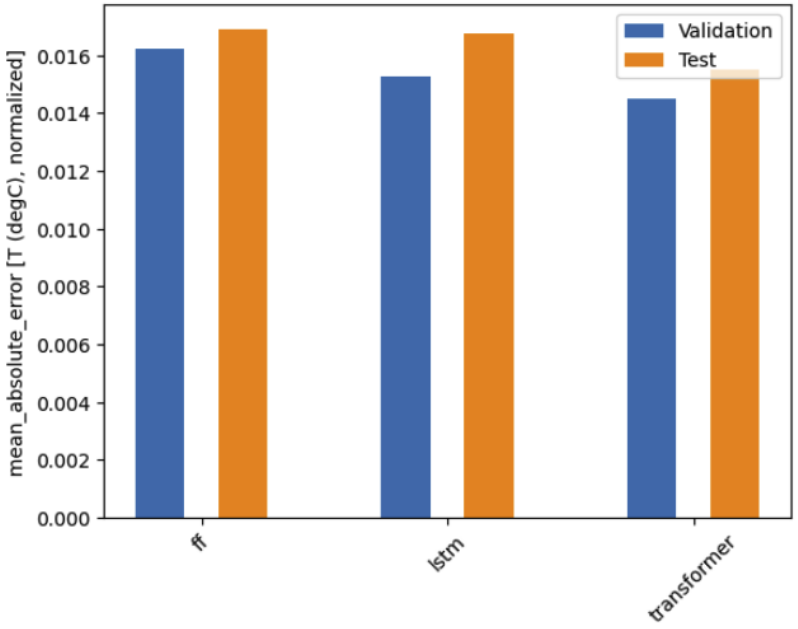

Figure 9: Performances on small window

$$- - - - - - - FF - - - - - - - \tag{10}$$
$$Validation : 0.01621 \tag{11}$$
$$Test : 0.0169 \tag{12}$$
$$- - - - - - - LSTM - - - - - - - \tag{13}$$
$$Validation : 0.01527 \tag{14}$$
$$Test : 0.01676 \tag{15}$$
$$- - - - - - - TRANSFORMER - - - - - - - \tag{16}$$
$$Validation : 0.01449 \tag{17}$$
$$Test : 0.01551 \tag{18}$$

Not surprisingly, the LSTM model performs better than the Feed-forward network. The performances of the Transformer are impressive compared to the LSTM. It highlights the power of attention.

2.3.2 Window 15 inputs, 5 outputs

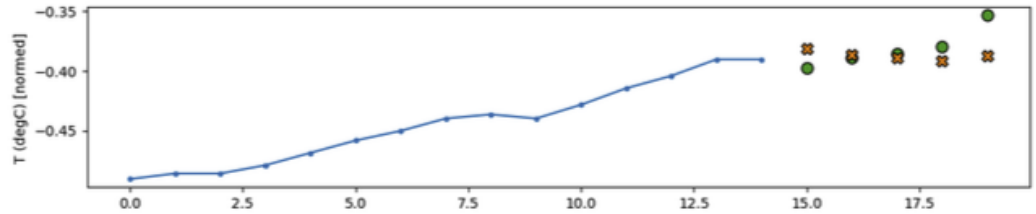

Figure 10: Feed-Forward

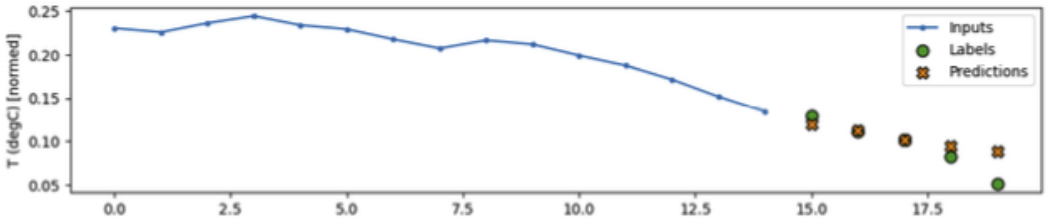

Figure 11: LSTM

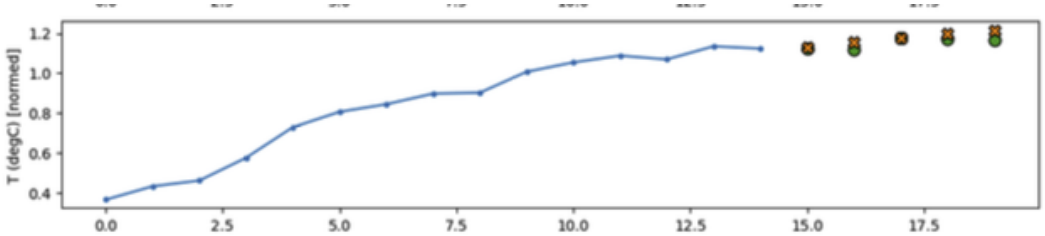

Figure 12: Transformer

$$------FF------ \tag{19}$$
$$Validation : 0.03508 \tag{20}$$
$$Test : 0.0361 \tag{21}$$
$$------LSTM------ \tag{22}$$
$$Validation : 0.03364 \tag{23}$$
$$Test : 0.03478 \tag{24}$$
$$------TRANSFORMER------ \tag{25}$$
$$Validation : 0.03313 \tag{26}$$
$$Test : 0.03458 \tag{27}$$

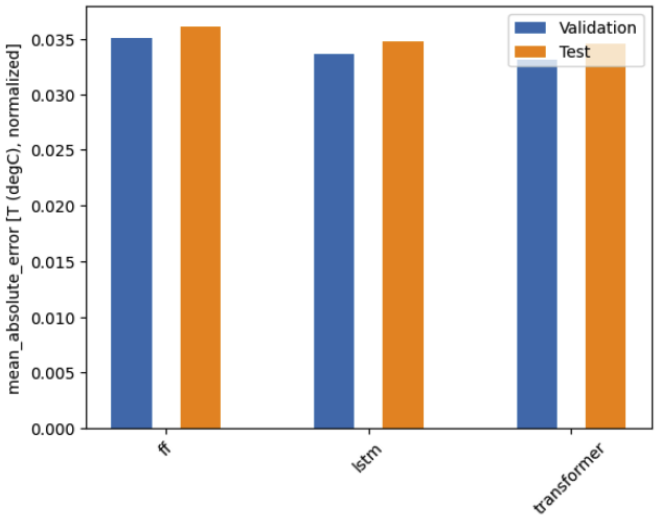

Figure 13: Performances on large window

Multiple outputs models do not perform great. The benefits of attention do not really shine. Even though the LSTM model performs better than the Feed-forward, and the Transformer better than the LSTM, it is not as distinct as the results with one output.
Auto-aggressive models could get better performances. Given the performances of the Transformer model on 1 output, an auto-aggressive Transformer could get very decent results.

3. Sentiment analysis

3.1 Problem statement

The goal of this experiment is to predict the sentiment associated with sentences. Each review used as training data is either positive or negative, hence this is a binary classification.

3.2 Implementation details

3.2.1 Data used

Three datasets containing reviews are used for the model training and testing: - A set of Amazon reviews which contains 1000 records - A set of reviews from the Yelp website which contains 1000 reviews - A set of reviews from Imdb that contains 1000 reviews

To put things into the context: Imdb is a movie database allowing users to rate the films they have seen. Yelp is a website that publishes crowd sourced reviews on various services.

3.2.2 Data pre-processing

All reviews are csv files. Each record contains both a review and the associated label (either 0 or 1). The first step performed was to get all reviews in one list and their associated labels in another list. Then, as shown below, a wordcloud is generated to figure out the importance of each word in reviews. A wordcloud is a visual representation of words where each word size is proportional to its prevalence in the reviews.

As we can see on the wordcloud "good","great" and "movie" are among the most frequently used words in the reviews.

Figure 14: Wordcloud corresponding to the reviews

Then, a normalization of the words is necessary. All words in the list are split and capital letters are replaced by regular letters. Besides, all special characters are removed using the re module of python,that handles regular expressions. The next step consists in getting the POS tags of all words, that is, to match each of the words with a tag indicating their tense, nature or genre. Adding this kind of tag allows the

model to get additional information on the the underlying structure of reviews. For this experiment, we choose to only consider four tags indicating if the word either is a noun, an adjective, a verb or an adverb. The library used to get those tags is NLTK, a widely used natural language processing toolkit for Python. Ultimately, POS tags are used to lemmatize words in each reviews. Practically, a lemma is the base form of a word, that removes inflectional endings and allows for a morphological analysis of the words. Words are therefore replaced by their corresponding lemmas and added to a list.

Finally, sequences of lemmas need to be turned into numerical sequences. In order to perform this transformation, a tokenizer is used. It creates a dictionnary keeping the most frequently used words and replaces each word in the reviews by its index in the dictionary.

Then, sequences are padded to have the same length. Finally, sequences are put in an array.

When it comes to the transformer model, the tokenization is performed in a slightly different way: The tokenizer is directly imported from the transformers library.

3.2.3 Data split

Following the data normalization and lemmatization, the data is divided into the training and testing sets. As a matter of fact, the training set represents 80% of the data and the testing is made of the remaining 20% of data.

3.2.4 Model creation

Three models are compared to tackle the sentinent prediction task: - A standard feed forward network - A model using an LSTM layer - A tranformer model based on the BERT architecture

Let's describe the architecture of each model in further details:

Architecture of the standard feed forward model:

- Input layer: dense layer with 32 units and ReLU activation
- Dropout layer with 20% dropout rate
- Batch normalization layer
- Output layer: Dense layer with one unit and Sigmoid activation

Architecture of the LSTM model:

- Input layer
- embedding layer
- dense layer with 32 units
- dropout
- batch normalization layer

- output layer: dense layer with 1 unit and sigmoid activation

Architecture of the BERT model: BERT models appeared recently in the field of transformers and had huge impact on NLP. It's a type of transformer using multiple attention layers. The transformer used is a smaller version of the BERT model called the distilBERT, that is 40% smaller than a traditional BERT based model.

3.3 Results and discussion

3.3.1 Feed forward model

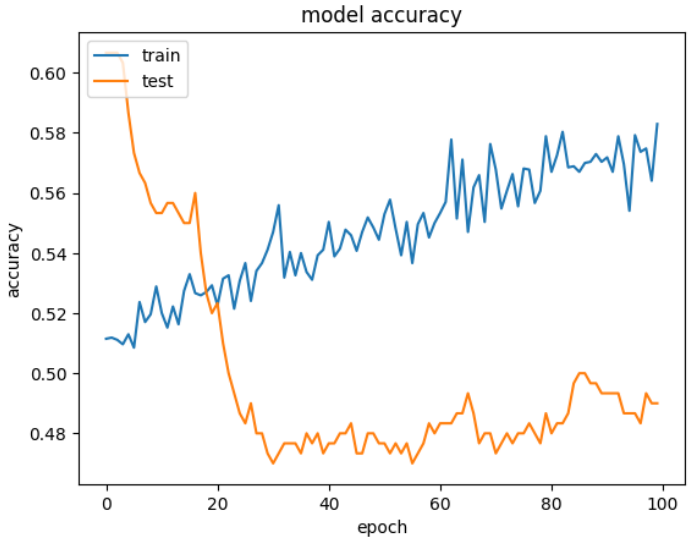

Figure 15: Feed forward training plot accuracy

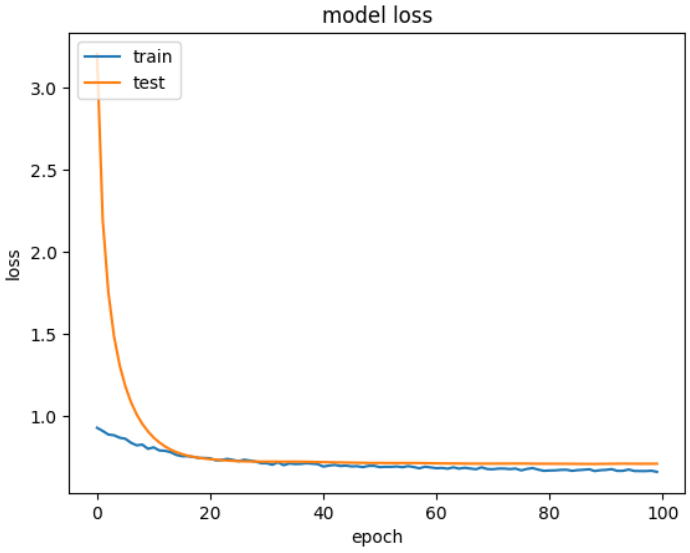

Figure 16: Feed forward training plot loss

As seen on the performance plot, the feed forward model performs poorly on the sentiment analysis task. The model only reaches 58% of test accuracy and overfits quickly as reflected by the validation loss increasing along the epochs.

3.3.2 LSTM model

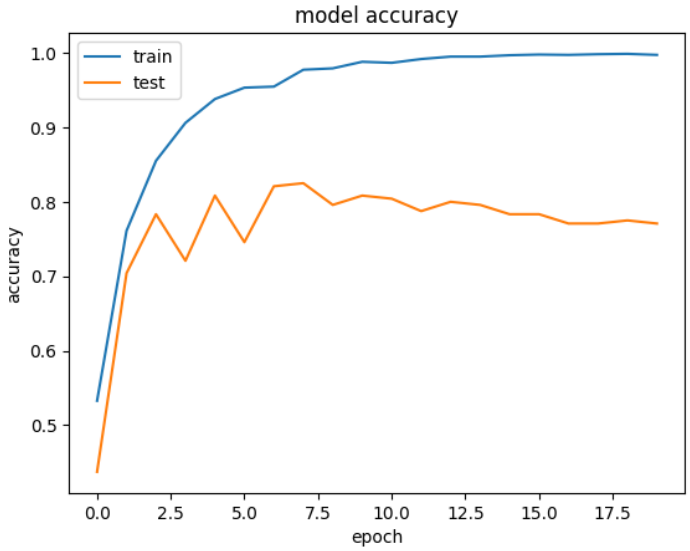

Figure 17: LSTM training plot accuracy

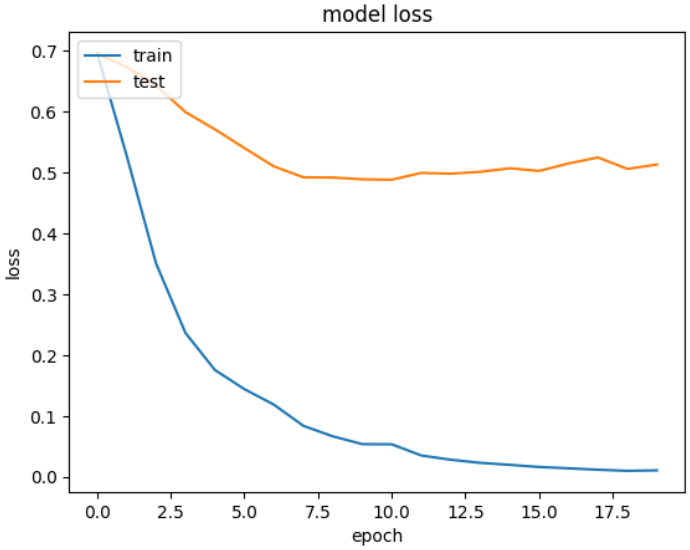

Figure 18: LSTM training plot loss

As for the LSTM model, it reaches 75% accuracy on the test set and doesn't overfit. However, the accuracy doesn't seem to go over 80%.

3.3.3 Transformer model

When it comes to the distilBERT model, we notice that it reaches a test accuracy of 95% after only three epochs. However, the model still seems to slightly overfit. Overall, this model outperforms both the feedforward model and LSTM model.

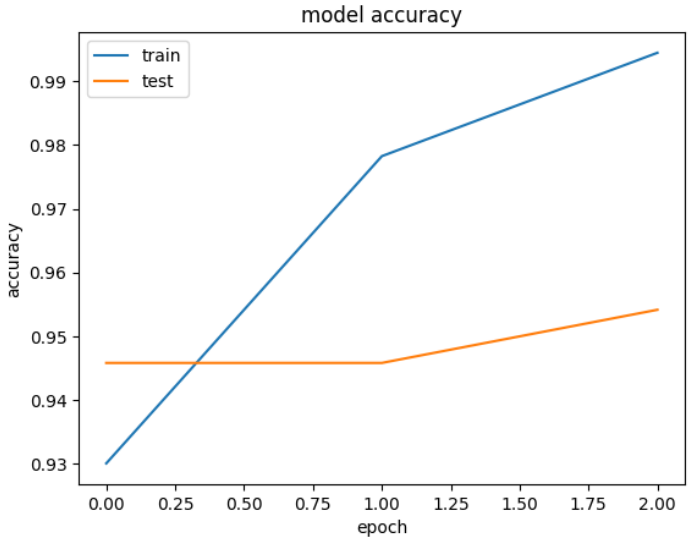

Figure 19: Transformer training plot accuracy

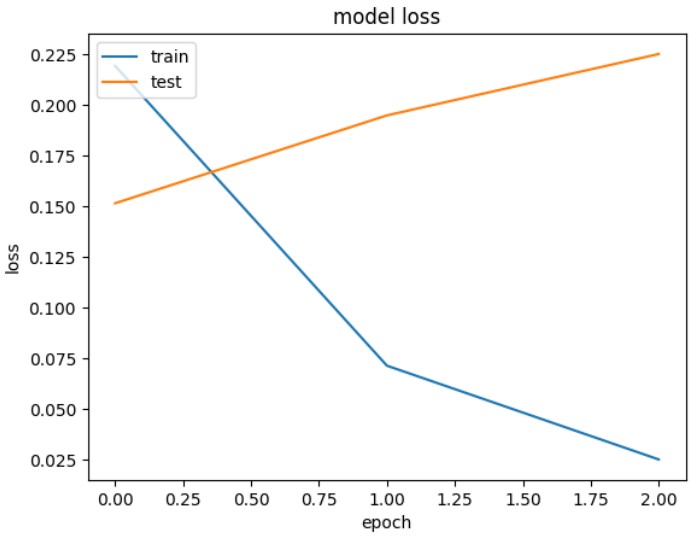

Figure 20: Transformer training plot loss

4. Conclusion

To wrap it up clearly, we have demonstrated through theses two experiments that tasks involving sequences are better handled by LSTM and transformers models than with feed-forward neural networks. On the sentiment prediction task, the performance improvement provided by the transformer model is significant. As far as the weather prediction experiment is concerned, the transformer outperforms the LSTM and feed-forward model, even if the performance gap isn't as significant. Overall, we see that the attention mechanism is a big leap forward on sequences based tasks like time-series prediction and natural language processing.
However, it is worth noticing that the Transformer model is harder to train. Nevertheless, it is parallelizable, contrary to LSTM models.

References

[1] Samarth Agrawal. Sentiment analysis using lstm. https://towardsdatascience.com/sentiment-analysis-using-lstm-step-by-step-50d074f09948.

[2] TensorFlow documentation. Neural machine translation with a transformer and keras. https://www.tensorflow.org/text/tutorials/transformer.

[3] TensorFlow documentation. Time series forecasting. https://www.tensorflow.org/tutorials/structured_data/time_series#setup.

[4] Max Planck Institute. Jena climate dataset. https://www.bgc-jena.mpg.de/wetter/.

[5] Damien Sileo. Understanding bert transformer: Attention isn't all you need. https://medium.com/synapse-dev/understanding-bert-transformer-attention-isnt-all-you-need-5839ebd396db.

[1] [5] [4] [3] [2]