Brno University of Technology
Faculty of information technology

Course: Soft Computing (SFC)
Project: Long Short-Term Memory (LSTM) Demonstration
Author: Šimon Šmída (xsmida03)
Mail: xsmida03@stud.fit.vutbr.cz
Date: November 27, 2023

1 Introduction

This project illustrates the training and application of Long Short-Term Memory (LSTM) networks within natural language processing (NLP). Specifically, it focuses on the effectiveness of LSTMs in performing a straightforward task: predicting a symbol within a sequence of zeros and ones. This particular task was chosen because of the computational challenges involved in training these networks on more complex tasks using a local machine. This project is still under development for future personal purposes and the latest version can be found on the GitHub repository.

1.1 RNNs and LSTMs in theory

Recurrent Neural Networks (RNNs) are adept at processing sequences by maintaining 'memory' of past inputs. However, they struggle with long sequences due to the 'vanishing gradient' problem, where gradients either become too small or too large to be useful during training.

LSTMs address this by introducing 'gates' in their architecture. These gates allow LSTMs to regulate the information flow, deciding what to retain or discard, thus maintaining longer-term dependencies in data. This capability makes LSTMs particularly effective for tasks requiring an understanding of long sequence contexts, like language modeling or time-series analysis.

2 Implementation

The project consists of the following files:

- `main.py` – main script, entry for the program
- `app.py` – Graphical User Interface of the app
- `progress_window.py` – progress window of the app

- `model.py` – Keras implementation of LSTM model
- `custom_lstm.py` – custom implementation of LSTM model based on A. Karpathy's blog post [1]
- `utils.py` – utility functions and global variables

2.1 LSTM Architecture

The project utilized 2 approaches to creating LSTM models. The first involved custom implementation of the entire LSTM architecture from scratch, this approach was inspired by A. Karpathy's Vanilla RNN implementation [1] and C. Colah in-depth article related specifically to LSTMs [2]. Because of the multiple optimizations made by TensorFlow's Keras API, the second approach of creating an LSTM model was selected as default. The model architecture can be seen below:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(50, return_sequences=True, stateful=True,
                         batch_input_shape=(batch_size, sequence_length, 1)),
    tf.keras.layers.LSTM(50, return_sequences=False, stateful=True),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

The architecture was implemented using TensorFlow's Keras API and comprises a sequential layout with two LSTM layers followed by a Dense layer. The first LSTM layer, with 50 units, maintains statefulness and returns the full sequence, facilitating the capture of temporal dependencies. The second LSTM layer, also with 50 units, is stateful but returns only the last output, suitable for feeding into the subsequent Dense layer. The Dense layer, using a sigmoid activation function, is designed for binary classification, producing a probability output. This configuration is apt for sequence processing tasks where understanding temporal patterns is crucial.

The custom LSTM implementation can be found in the `custom_lstm.py`.

3 Application

3.1 Software and GUI

The project was originally developed on Windows 11, nevertheless, the program should also run on Linux-based systems (tested on VUT FIT `Merlin` server). It was implemented in Python with TensorFlow for model development and PyQt5 for the graphical user interface (GUI). The GUI allows users to interactively input sequences of zeros and ones, train the LSTM model (with an option to pause and continue the process), and view predictions in real time. The main dependencies are listed below:

- `PyQt5` – implementation of the GUI
- `TensorFlow` – utilizing Keras API
- `numpy` – efficient mathematical operations
- `matplotlib` – visualization of the learning process

After installing all the required dependencies (listed above), the application can be executed with `python3 main.py` command in the directory containing all relevant files to the project, listed in Section 2.

3.2 Usage

The application is designed for intuitive use. Users can input a sequence of zeros and ones. This sequence is considered a pattern that the LSTM network attempts to recognize, learn, and then be able to predict the following symbols in the sequence. A user can also initiate the training process, and observe the model's performance and predictions. The GUI (see Fig. 1) displays real-time updates on training progress, loss metrics, and the predictions for the input sequences.

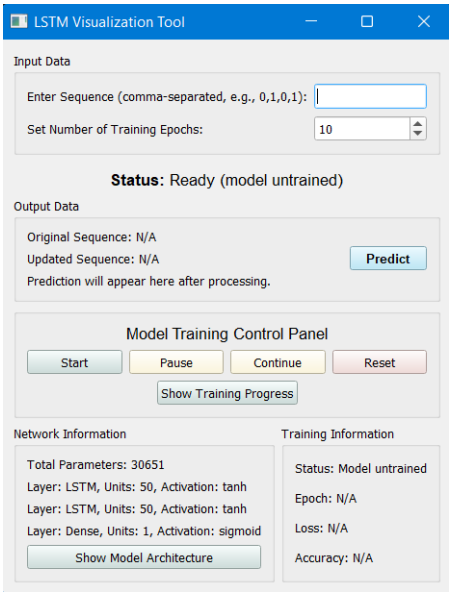

Figure 1: The initial GUI view – Main Window.

As previously mentioned, the input for the LSTM model is an arbitrary sequence of zeros and ones, arbitrarily long. It is considered as a repeating pattern, which the network attempts to learn. A user has also the ability to set the desired number of epochs for the training process of the LSTM model (default is 10).

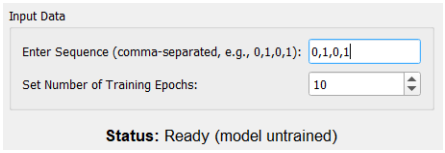

Figure 2: Input group: user can enter a sequence of 0s and 1s.

The GUI displays the real-time status of the program (see Fig. 3), which enables a user to simply see what the system is currently doing. The output of the program is displayed below, and it consists of the original input sequence (entered by a user), and the update of it after a user trained the model and clicked on the predict button. This sequence of steps results in an LSTM network predicting the next symbol in the original sequence, along with the prediction confidence.

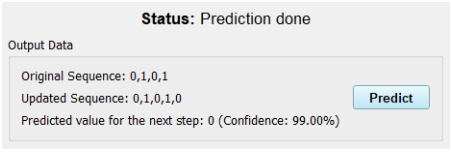

Figure 3: Output group: predicting the next symbol in the sequence.

The training of the LSTM model is handled in the control panel section of the application (see Fig. 4). A user is able to manage the training process himself, and also to display the predictions being learned and the loss curve along with the accuracy of the network.

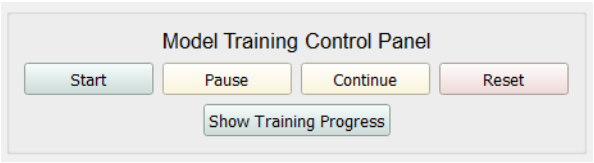

Figure 4: Model Training control panel – user has the ability to start, pause, continue, reset as well as visualize the training process.

An example of the training process visualization is shown in Fig. 5. The window displays the progress bar of the network learning and two plots, the individual predictions learned in time vs. the actual values (left) and the loss curve vs. network accuracy (right).

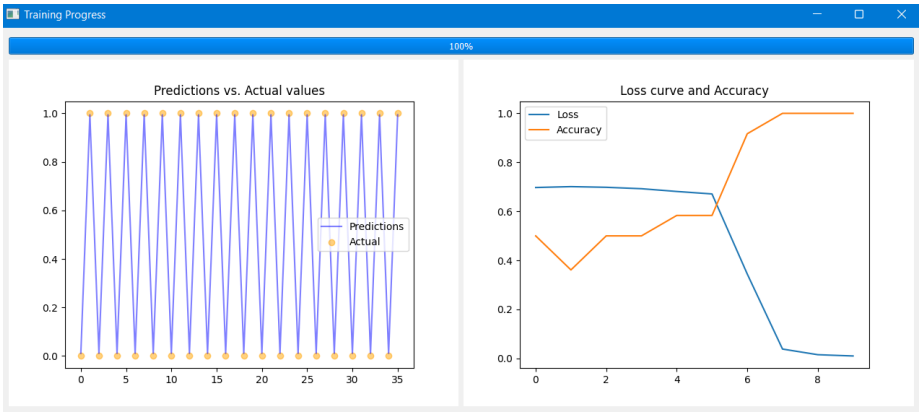

Figure 5: Training process visualized (10 epochs, predicting pattern `0,1,0,1`)

A user has the ability to display the currently used model architecture (see Fig. 6).

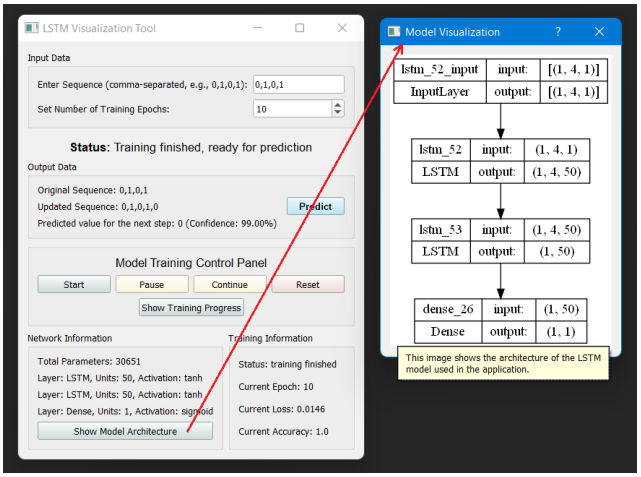

Figure 6: Selected model architecture visualization.

The last section of the Main Window of the application provides information about the selected network (left), and information about the training process itself (right), see Fig. 7.

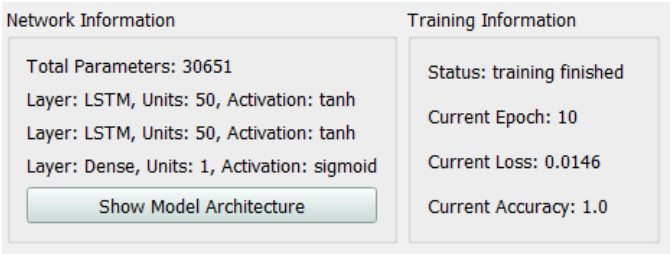

Figure 7: Information about the network – selected network info (left), training info (right).

The visualization of the training process works in real-time. A user, as mentioned, can control the training process (start, pause, continue and reset) and see the values learned by the model at a specific learning epoch, see Fig. 8.

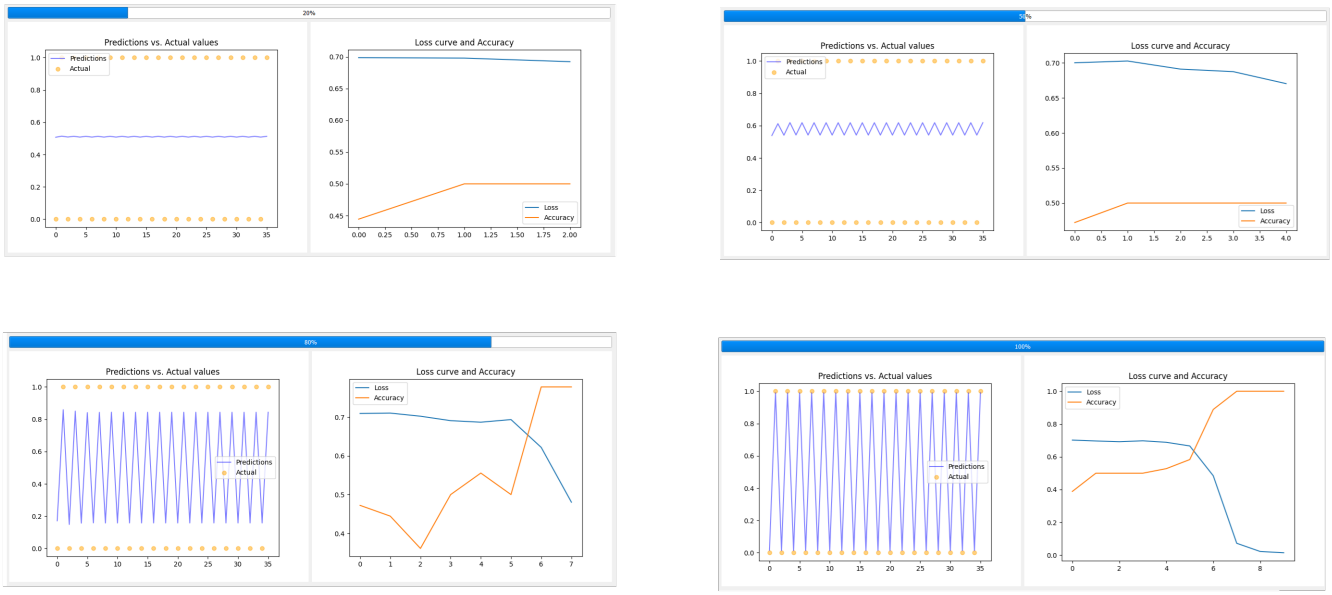

Figure 8: Training process visualization (user can step through the process).

4 Future work

User option to select another model (not necessarily LSTM), and corresponding hyperparameters (learning rate, activation functions, ...). The application shows the inner workings of the network (cell state, hidden state for LSTM), which could help better understand how the LSTM works internally.

References

[1] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. https://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015. Accessed on 27 November 2023.

[2] Christopher Olah. Understanding LSTM networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. Accessed on 27 November 2023.