Methodology for Predictive calculator

Anton Mukin

June 2025

0.1 Brief Summary of the Project

This Matura project investigates and evaluates the arithmetic capabilities of different neural networks.
The project began with a literature review to generate a hypothesis regarding the weaknesses of neural networks in performing simple arithmetic. This literature review was submitted as the Zwischenprodukt, alongside a proof-of-concept notebook featuring a comparison of Feed-forward Neural Networks (FNNs) of different sizes.
The next step was to build a Recurrent Neural Network (RNN) and similar attention-based RNNs to investigate their arithmetic capabilities and compare them to those of the FNN using a benchmark.
The benchmark's baseline was defined to be the performance of a basic FNN's performance on different, but roughly still similar arithmetic tasks.
Afterwards, the same was done for the transformer type of neural network model. Here, their exact functionality was thuroughly investigated, because of their unique architectures.
Lastly a simillar workflow was repeated for some bigger, pre-trained models.
And finally all the findings were collected and evaluated as a whole.

0.2 Introduction to this document

The goal of this document is assisting reproducability and showing how the findings discussed in the other document have been obtained.
All of the code written for this project is available in the github reporsitory:
`https://github.com/AntonStantan/matura`
In this project all of the code is written in Python-notebooks (Jupyter Lab).
The preffered library used was tensorflow keras.
Most of the models were trained locally on a Nvidia GPU device: *Nvidia Jetson Orin Nano Super Developer Kit*

Contents

1 Feed-forward Neural Networks (FNN)

This section refers to the /FNN directory in the github reporsitory.
There are two notebooks here: FNN1 and FNN2.

1.1 Train and Test data

Defining the train data; The decision was made to only use subtraction and addition. The arithmetic expressions were defined to consist of two operators (+ or -) and three integers $[-5, 5]$
The reason for these definitions are: + and - are the 2 simplest operators. And the decision to introduce the model to 3 numbers, in place of the more commonly used 2, was made in hopes of simplifying the transition to more numbers later on.
e.g., the first three entries are:

$$1 - -2 + 3 \qquad -2 + -3 - -5 \qquad 3 - 1 - 0$$

In total there are 1907 expressions like this in the train data.

The test data is also defined in a slightly unconventional manner. It is split up into three categories.

- Inside of the number range: Expressions just like in the train data but not inside of train data.
- Outside of the number range: The same expressions, but with numbers in the ranges $[-8, -5]$ and $[5, 8]$ e.g.,

  $$-6 + 6 + 8 \qquad -7 + 6 + 7 \qquad 5 - -6 - 6$$

- Longer expressions: Expressions of different lengths. Specifically, there are between 2 and 8 numbers inside of the $[-5, 5]$ range. e.g.,

  $$-5 + 1 \qquad 3 + -1 + 4 + -2 + -4 \qquad -2 + 1 + 5 + -5 - -2 + -1 - 2 - 5$$

1.2 Drop-Out

When introducing a Drop-Out with an industry standard value of 0.3; contrary to the expectation of reducing over-fitting, which in some way is present, as the models aren't able to generalize beyond the training range, this has a negative effect on the model. The MSEs of models with Drop-out are higher then, the ones of the previous models without drop-out.

2 Recurrent Neural Network (RNN)

2.1 Introduction

Recurrent Neural Networks (RNNs) work similar to FNNs with one key difference: There is a vector called the hidden-state. This vector contains information about previous inputs. The hidden-state of the previous time-step, in addition to the input of the current time-step, is fed into a model which computes the hidden-state of the present time-step. The output of each time-step is calculated by feeding the respective hiddenstate to a model.

Numerical Visualization:
Let:

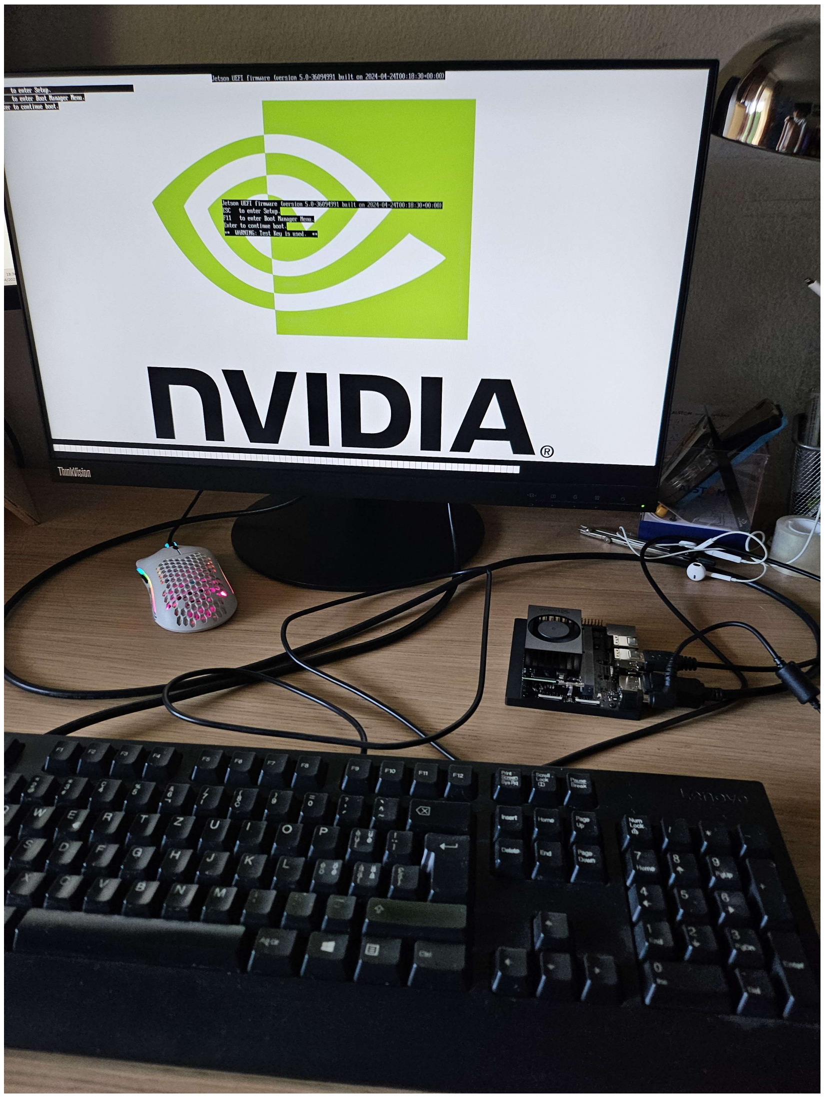
- x_t: Input at time step t
- h_t: Hidden state at time step t
- y_t: Output at time step t
- W_{xh}: Weight matrix connecting input to hidden state
- W_{hh}: Weight matrix connecting previous hidden state to current hidden state (recurrent weights)
- W_{hy}: Weight matrix connecting hidden state to output
- b_h: Bias vector for the hidden layer
- b_y: Bias vector for the output layer
- σ: Activation function (commonly tanh or ReLU for the hidden state)
- σ_{out}: Activation function for the output (e.g. softmax for classification, or linear in our case of regression)

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = \sigma_{out}(W_{hy}h_t + b_y)$$

Useful sources for the creation of the first RNN prototype:

3 attention



```
Jetson UEFI firmware (version 5.0-36094991 built on 2024-04-24T00:18:30+00:00)
ESC   to enter Setup.
F11   to enter Boot Manager Menu.
Enter to continue boot.
**  WARNING: Test Key is used.  **
```

Figure 1: You can see the aformentioned Nvidia Jetson device booting up.