Artificial Intelligence and Robotics

Neural Network Project

Realized by: Nicolò Dentale 1868948, Stefano Iervese 2057309

Academic Year: 2022-2023

Introduction

We present a comparative analysis of the performance of classical RNN networks, including Vanilla, GRU and LSTM models, versus the novel architecture proposed by Michael Rotman and Lior Wolf named Shuffling RNN.
We adressed both regression as the main problem and classification problem as experiment.
The regression dataset used in the study consists of a multivariate time series related to wheather conditions. Specifically, the time series consisted of four variables: humidity, pressure, temperature, and wind speed. The classification dataset is a text collection of movie reviews labelled as positive or negative.

1 Method Description

The main issue that limits the performance of traditional recurrent neural network models is the exploding or vanishing gradient. While GRU and LSTM utilize a gating mechanism to stabilize the gradient flow between subsequent hidden states, SRNN identifies the source of the problem in the matrix multiplication over the previously seen hidden states. Since the first input of a sequence is processed more times than later ones there appeare to be a gap in the way each time step influences the network weights during training. Consider the framework for a Vanilla RNN:

$$h_t = \sigma(W_1 h_{t-1} + W_2 x_t) = \sigma(z_t) \tag{1}$$

where σ is a nonlinear activation function.
Using Backpropagation through time we obtain

$$\frac{\partial L}{\partial h_t} = \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial W_1} = \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} \sigma'(z_t) h_{t-1} \tag{2}$$

Depending on the maximal eigenvalue of W_1, the repeated multiplication by W_1 in $\frac{\partial L}{\partial h_t}$ may lead to exponential growth or decay in the gradient. SRNN focuses on constraining the weight matrix W_1 to be orthogonal. The method benefits from this property, since the successive applications of W_1 do not increase nor decrease the norm of the hidden state vector, such that the gradients with respect to the hidden state do not suffer from an exponential growth or decay. Therefore, the method can be applied without the use of any gradient clipping schemes.
The SRNN architecture consists of two hidden-state processing components, the learned

network b that is comprised of fully connected layers, and a fixed permutation matrix W. At each time step, the layer receives two input signals: the hidden state of the previous time step, h_{t1} and the input at the current time step x_t.

$$h_t = \sigma(W_p h_{t-1} + b(x_t)) \equiv \sigma(z_t) \tag{3}$$

Network b is given by the sum of a Multilayer Perceprtron and a single affine layer with a sigmoid activation function.

$$b(x_t) = f_r(x_t) \odot \sigma(W_s x_t + b_s) \tag{4}$$

The output of the network at time step t is obtained by using a single affine layer.

$$o_t = s(h_t) \tag{5}$$

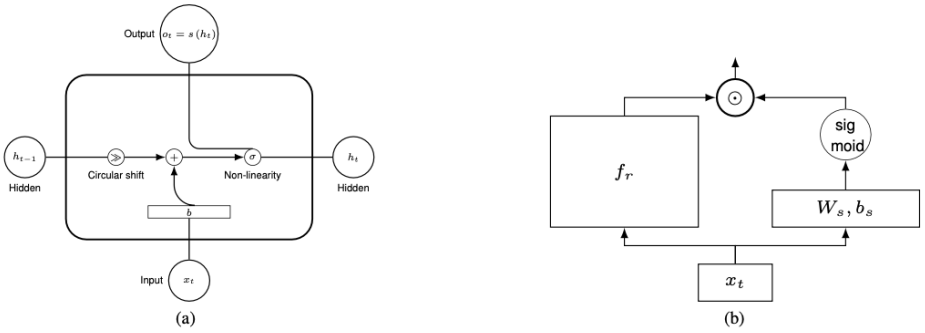

Figure 1.1: SRRN framework

As further experiment we have used the SRNN for a text binary classification problem.

2 Code instructions

The project has been built using the following libraries: *Pythorch, Scikit Learn, Pandas, Matplotlib, Numpy, Torchtext.* Please install all the libraries listed before running. To start the program you have to be in one of the 2 folders in the project, tiping `cd ./Classification` or `cd ./Regression`. In the folder simply type '`python main.py`'.

The primary function of the project accepts three command line arguments

- -t: to test the selected model with the chosen hyperparameters
- -i: to train all the neural networks to plot our experiments

- -r: to load a previously trained model

If type **'-t'** you can use more arguments:

* '-m' for the model you want to train, followed by the initial letter of the possible models: `rlsg`
* '-s' to save the model
* '-l' to select the learning rate
* '-p' to plot the loss
* '-e' to select the number of epochs
* '-b' to select the batch size
* '-d' to choose the directory of the dataset
* '-h' to select the hidden size

To study a time series, it is common to extract a sequence of values from the series and try to predict the last item of the sequence. This process is then repeated by sliding the sequence window by one step until the end of the time series is reached. This preprocessing is made by the class Data in the file *utils.py*. The minibatch extraction is made through a dataloader with shuffling enabled. We have chosen to use the MSE and the BCE loss functions (one for regression and the other for classification) and the Adam optimizer in order to control gradient explosion. The four Neural Networks have been implemented by scratch. When not declared the following default hyperparameters will be used: hidden size = 128, batch size = 32, epochs = 100, sequence length = 100.

Experiments

We have conducted some experiments to test the effect of different hyperparameters. It is possible to train the four models varying the sequence lenght in the range 50,100,200,400 and the hidden size in the range 64,128,256,512. The SRNN has been tested with different shift values between 0 and 3.

As a conclusive test we have applied the SRNN in a text binary classification problem. The dataset consists of movie reviews labelled as positive or negatives. For every review a vocabulary associate each word to a token which are then passed through the network sequentially. As for regression we executed experiments varying sequence length and shift value in SRNN for the classification problem.

3 Results

The empirical findings of the project are consistent with the theoretical expectations displaying improved outcomes using SRNN.

Regression

GRU Model needs longer training: in early trials the network gave the appearence of not functioning properly until training time has been extended. On the other hand SRNN performs better on shorter training. LSTM proves itself to be a competitive alternative.

Hidden Size Variation

LSTM shows the best performance with short hidden size but as it increaeses the model is progressively outperformed by SRNN.

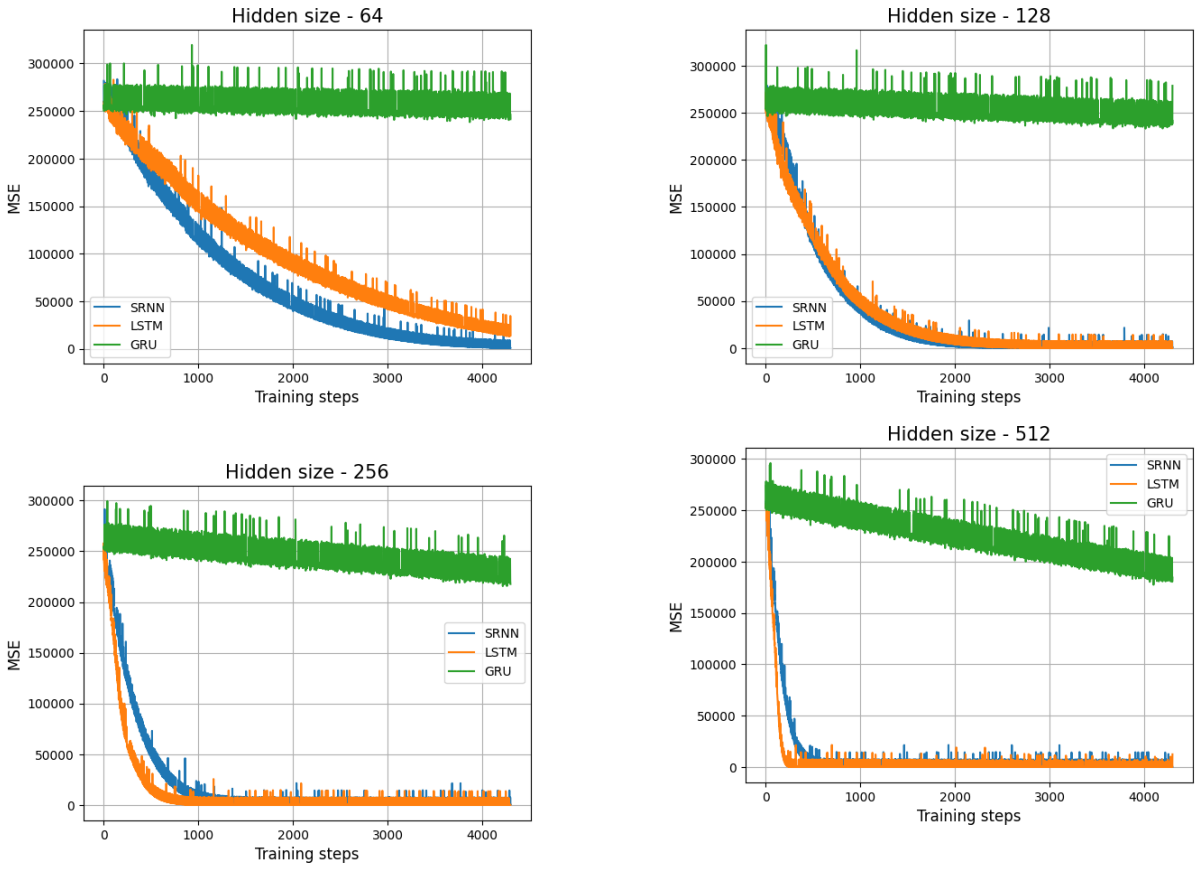

Figure 3.1: Hidden size comparison in regression

Sequence Length Variation

An opposite trend is displayed varying the sequence length. The longest the sequence the more is preferrable to use LSTM over SRNN. In both experiments GRU shows shortcomes in comparison with LSTM and SRNN.

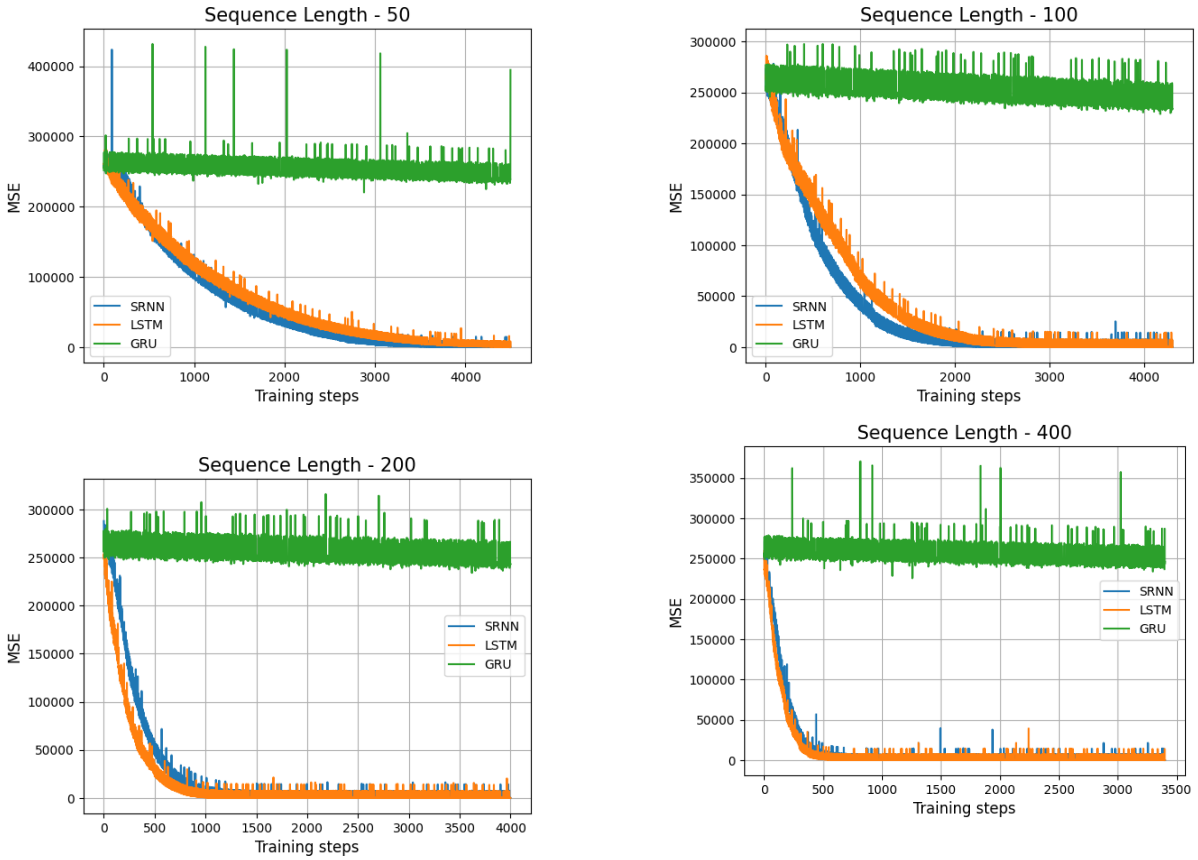

Figure 3.2: Sequence Length comparison in regression

Shift Variation

Regarding different shifts of the permutation matrix we acknowledged a direct proportionality between the shift size and training time. In the long term big shifts lead to more advantegeous accomplishments.

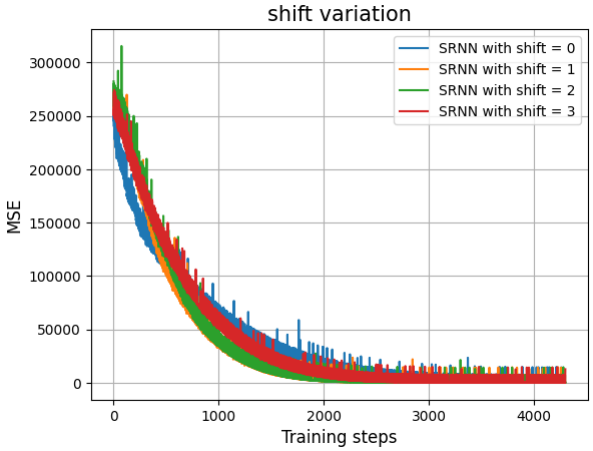

Figure 3.3: Shift variation comparison for SRNN in regression

Classification

Hidden Size Variation

In relation to the classification problem LSTM model exhibits its superiority in handling longer sequences in comparison to other models. Its accuracy depicts an exponential decay, making it a suitable solution for classification. SRNN may be chosen for fast training but this is unlikely to happen in real applications.

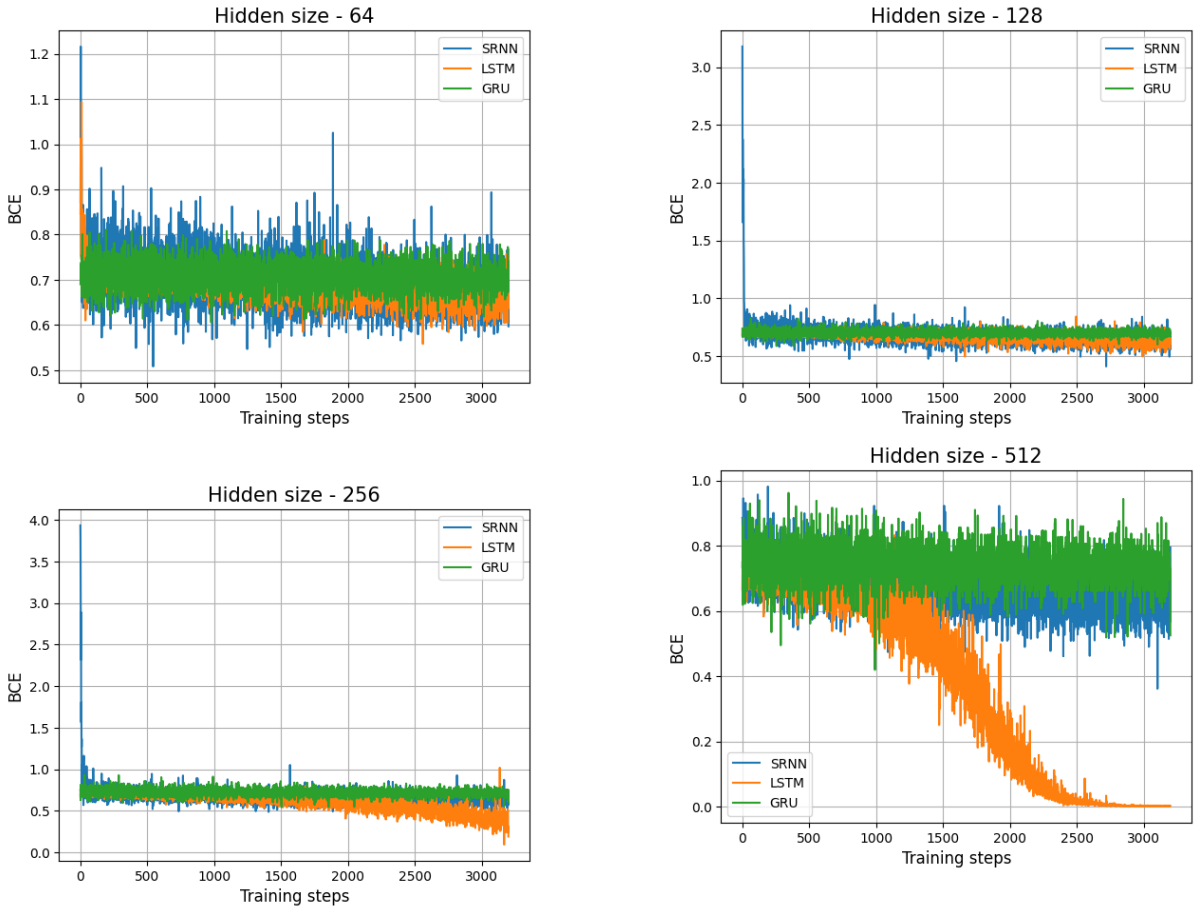

Figure 3.4: Hidden size comparison in classification problem

Shift Variation

Despite big shifts seems to strength SRNN, in the long the long term perfromance in classification using different values are comparable.

Figure 3.5: Shift variation comparison for SRNN in classification

References

[1] Rotman, M., Wolf, L. (2018). Shuffling recurrent neural networks.

[2] DailyDehliClimate.csv from `https://www.kaggle.com/`

[3] IMDB_Dataset.csv from from `https://www.kaggle.com/`