An analysis of Recurrent Neural Networks architectures for stock price prediction with Julia ®

Irene Simó Muñoz
irene.munoz @ estudiantat.upc.edu

May 2023

Abstract

The analysis of sequential problems with Recurrent Nural Networks (RNN) is a widely explored subject. Despite some of the disadvantages of this tyoe of network such as the exploding or vanishing gradient, it is capable of providing results with great accuracy with proper tuning. This analysis aims to tackle two sources of underperformance of RNNs for time series anaylsis; the determination of the number neurons og the hidden layer and the architecutre of the RNN in terms of input and output size. In line of the state-of-the-art, the experimental side is carried out with Julia, the programming language[a]

[a]Julia is a registered trademark of JuliaHub, Inc.

1 Introduction

In time series analysis it is important to include past information to detect timely patterns. Some popular architectures that do so are Recurrent Neural Networks (RNN). Notable, RNNs suffer from short-term memory problems. It is caused due to vanishing gradient problem. As RNN processes more steps it suffers from vanishing gradient more than other neural network architectures. The gradients exponentially shrink as we backpropagate. A smaller gradient means it will not affect the weight updation. Due to this, the network does not learn the effect of earlier inputs. Thus, causing short-term memory problems.

This study aims to report the effects of hidden layer size and different RNN architectures for time series prediction.

The determination of both the number of hidden layers as well as their size (number of neurons) has been a hot topic of discussion across deep learning. Many publications aim to find mathematical models to fit the optimal number of neurons for specific types of NN, see optimal approaches for Feedforward Neural Network (FNN) (Xu and L. Chen 2008)

One issue within this subject on which there is a consensus is the performance difference

from adding additional hidden layers: the situations in which performance improves with a second (or third, etc.) hidden layer are very few. One hidden layer is sufficient for the large majority of problems (Stathakis 2009) (Ciftcioglu and Turkcan 1992). Furthermore, NNs with more hidden layers are extremely hard to train (LeCun, Bengio, and Hinton 2015) and thus those problems that require more than a hidden layer are considered "non-solvable" by neural networks.

There are some empirically derived rules of thumb; of these, the most commonly relied on is 'the optimal size of the hidden layer is usually between the size of the input and size of the output layers' - non-suitable for some of the architectures explored in this study-. Other ones include some empirically derived formulas (Hagan, Demuth, and Beale 1997), but it is always advised to adapt the neurons to the problem in question.

Julia, the programming language Julia is used for the experiments reported in this analysis via its Flux package (Innes 2018). Julia is best known for scientific computing, machine learning, and data mining applications. In contrast to some other programming languages well-regarded for this matter, Julia employs a just-in-time (JIT) compiler that can generate highly optimized machine code, comparable to low-level languages like C and Fortran. This allows Julia to achieve competitive performance in numerical computations, which are crucial for deep learning tasks.

1.1 Stock price

The used data corresponds to the price history and trading volumes of the fifty stocks in the index NIFTY 50 from NSE (National Stock Exchange) India, extracted from Kaggle[1]. The

[1]https://www.kaggle.com/datasets/rohanrao/nifty50-stock-market-data?resource=download

Figure 1: WVAP evolution in Indian rupees during the studied time period

data spans from 1st January 2000 to 30th April 2021. In particular, the file used is BAJAJFINSV, corresponding to Bajaj Finserv Ltd.; a financial services company based in Pune.
The data used for this study only takes from May 26th, 2008 to November 16th, 2019 - the last reported prices before first China's COVID-19 publicly confirmed case, (World Health Organization 2020) -. This truncation is done to avoid over-informing the predictors with pandemic data, which is not a pattern we desire to include in the analysis.

the target variable to study is the Volume Weighted Average Price (VWAP) - traced in blue in Figure 1 -, a trading benchmark used by traders that gives the average price the stock has traded at throughout the day, based on both volume and price.

The usability of the selected NSE dataset is qualified with the maximum ranking, and quick analysis has shown that no null nor faulty values are contained in the selected date range. Before experimental runs, two main data manipulations are done.

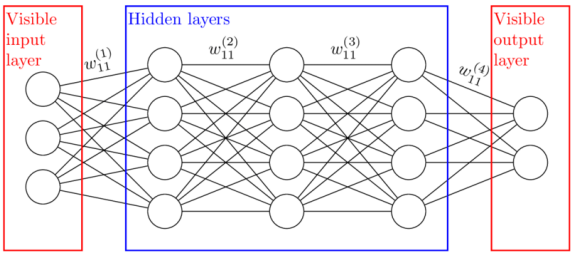

Figure 2: Neural Network structure. Adapted from *Hidden Layer* n.d.

2 Hidden layer size

The number of neurons in a hidden layer (see Figure 2) of a RNN can indeed have an impact on the accuracy of the results.

The choice of the hidden layer size is an important hyperparameter that affects the model's capacity to learn and represent complex patterns in sequential data.

In terms of representation power, increasing the number of neurons in a hidden layer increases the model's capacity to learn and represent complex patterns in the data. A larger hidden layer can capture more intricate temporal dependencies and potentially model the underlying dynamics of the sequential data more accurately. This can lead to improved accuracy, especially if the data contains intricate long-term dependencies.

However, increasing the number of neurons also increases the model's capacity to memorize noise or irrelevant details in the training data.

This can result in overfitting. Moreover, the number of neurons affects the computational complexity of the model. Larger hidden layers require more computational resources, including memory and processing power, to train and evaluate. This translates into more iterations or longer training times to converge and reach optimal performance.

The process of deciding the number of hidden layers and the number of neurons in each hidden layer is unclear and has not been normalized. In this sense, two main lines of work are explored in general; attempts to generalize this process depending on the type of NN and attempts to do it depending on the nature of the problem the network tries to solve.

For both, statistical studies are carried out to find generic guidelines, such as a survey made to resolve the problem of the number of neurons in each hidden layer and the number of hidden layers.

For the former, various works focus on some popular types of networks like BPNNs (Karsoliya 2012) or Feedforward Networks (Teoh, Tan, and Xiang 2006).

For the latter, exploration is done in various problems, fields, and flavors of NN (Krasovsky et al. 2018).

1. Hagan, Demuth, and Beale 1997

$$n = \frac{N}{\alpha(N_x + N_y)} \tag{1}$$

With a value of α typically between 2 and 10.

2. Kanellopoulos and Wilkinson 1997. The upper bound is at most 4 times the nodes in the input layer

3. Hecht-Nielsen 1987

$$n = \frac{N_w}{N_x + N_y} \tag{2}$$

Where N_w is the necessary number of synaptic weights (in the original paper estimated to equations (3), (4), N_x is the input signal dimension and N_y is the output signal dimensionality

$$N_w \geq \frac{N_y Q}{1 + \log_2 Q} \tag{3}$$

$$N_w \leq \left(\frac{N_y Q}{N_x + 1}\right)(N_x + 2N_y + 1) \tag{4}$$

This formulation gives very loose bounds on N_w for the studied dataset, therefore the chosen value has been set to one that guarantees more exploration of the number of neurons search space when the experiments are run with the

Name	Listed	Eq.	Topology
Uni-neuronal	-	-	1:1:1
Hagan's rule	1	(1)	1:100:1
Hush rule	2	-	1:4:1
Hecht-Nielsen	3	(2)	1:40:1

Table 1: Some hidden layer topologies

other topologies.

These rules of thumb are not definitive and should be used as starting points for experimentation. The optimal number of neurons can depend on various factors, including the complexity of the problem, the amount of available data, and the architecture of the network. To try to guide the reader in the selection of a fitting number for the application described in this analysis, experiments are run to test the presented empirical formulations.

2.1 Experimental setup

To report the effect of different hidden layers configurations, the topologies detailed in Table 1 are tested for an else-wise identical RNN.

3 One-to-one vs. One-to-many

The different architectures for RNNs typically serve as boosting tools for the networks depending on the nature of the problem being studied.
For this analysis, One-to-one and One-to-many architectures (see Figure 3) are considered.

One-to-one sequence problems use one value to predict another single one. This type of neural network is known as the Vanilla Neural Network, and it's used for general machine learning problems, which have a single input and a single output. Thus, it is considered the

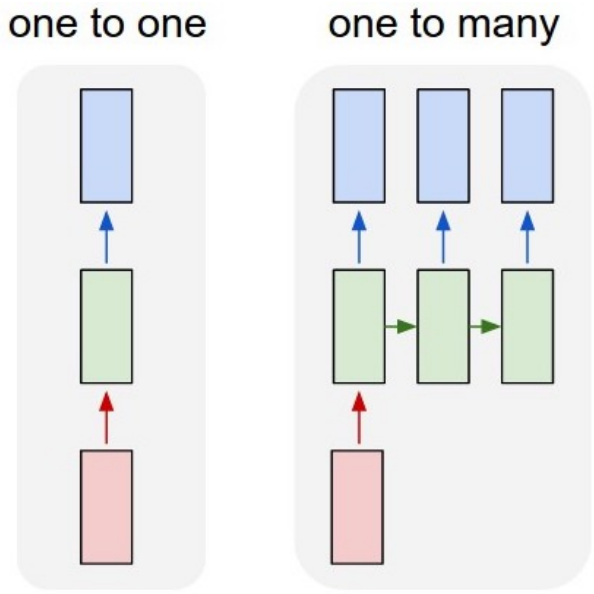

Figure 3: One-to-one vs. One-to-many architectures. Adapted from *The Unreasonable Effectiveness of Recurrent Neural Networks* n.d.

basic architecture and is the most generic one.

Although RNNs are well-suited for time series, these are only a particular case of a greater class of problems for which recurrent neural networks are great performers; sequence problems - this is, problems where the input is known to be gathered with a frequency or to belong to a series -. Sequences can be very intricate and often have patterns undetectable in short instances.

Furthermore, for time series in particular, RNNs are known to suffer from vanishing gradient problems, hence why some RNN variants such as Long Short Term Memory NNs have gained popularity (Manaswi and Manaswi 2018). This adds uncertainty to the predictions of RNNs and allows the error probability to grow as the network keeps accumulating past data to try to make the next time-step prediction if it is allowed to do so.

For this reason, it is believed that for simple RNNs it might be more useful to use the one-to-one architecture since it is simpler and cuts possible error accumulation from the root

	Mean [s]	Min [s]	Max [s]	Runs
Uni-neuronal	130.78 ± 5.89	126.8	132.69	10
Hagan's	132.02 ± 8.34	125.98	135.98	10
Hush	143.88 ± 8.51	134.72	148.76	10
Hecht-Nielsen	197.39 ± 7.48	188.62	202.50	10

Table 2: Time results for different HL configurations

	Mean (%)	Min (%)	Max (%)	Runs
Uni-neuronal	25.97 ± 10.89	14.98	30.23	10
Hush	70.24 ± 4.24	61.79	81.86	10
Hecht-Nielsen	98.70 ± 0.75	97.69	99.32	10
Hagan's	98.15 ± 1.05	89.34	99.75	10

Table 3: Accuracy results for different HL configurations

(Manaswi and Manaswi 2018).

One-to-many sequence problems correspond to scenarios where the input data has a one-time step, and the output contains a vector of multiple values or multiple time steps.

This architecture is commonly used for complex sequence analysis with RNNs; stock prices and high-schoolers grade prediction (Jayne, Lanitis, and Christodoulou 2011) or speech translation (Wang et al. 2022) and generation (Di Gangi, Negri, and Turchi 2019) and(J. Chen et al. 2019).

One of the main advantages of one-to-many architecture is that it allows having a unique source for the forecasting, for instance, a parameter known to have a notably higher impact on the sequence than the rest of the parameters, or one that is an important source of variation. Furthermore, the definition of the relationship between the variables of a multivariate distribution and a single source of variation allows the estimation of the values of multiple variables given the value of the single variable, thus making multivariate forecasting simpler.

3.1 Experimental setup

To test the effect of the architecture of the RNN in the prediction of the dataset, one-to-one, and one-to-many architectures are tested in an identical RNN. For the one-to-many architectures, furthermore, various output sizes are also tested; 2, 4, 8, and 16 in particular. This aims to identify if there exists a trend in regard to the magnitude of the nodes in the output layer.

4 Results

The comparison between the different solutions has been done with the `hyperfine`[2] tool, which provides statistical information of command runtime. The obtained results are presented in Tables 2 to 5, both included and can also be found in the exported files.

4.1 Hidden layer size

Tables 2 and 3 display the results obtained for the different topologies of hidden layers. Notice that the different configurations are sorted in ascending order to reflect any trends in the

[2] An open-source command-line benchmarking tool found in `https://github.com/sharkdp/hyperfine`

	Mean [s]	Min [s]	Max [s]	Runs
O2O	126.82 ± 8.85	110.83	137.70	10
O2M (2)	119.74 ± 7.12	108.51	129.91	10
O2M (4)	126.937 ± 7.38	117.61	135.86	10
O2M (8)	123.40 ± 8.67	115.36	130.72	10
O2M (16)	124.74 ± 8.02	109.27	134.24	10

Table 4: Time results for One-to-one (O2O) and One-to-many (O2M)

	Mean (%)	Min (%)	Max (%)	Runs
O2O	50.72 ± 10.89	21.67	52.8	10
O2M (2)	68.92 ± 7.32	64.49	75.65	10
O2M (4)	91.22 ± 8.22	79.51	99.98	10
O2M (8)	89.49 ± 7.23	79.29	97.9	10
O2M (16)	77.34 ± 4.88	72.99	85.2	10

Table 5: Accuracy results for One-to-one (O2O) and One-to-many (O2M)

behavior; Uni-neuronal (1 neuron), Hush's (4 neurons), Hecht-Nielsen (40 neurons), and Hagan's (100 neurons).

A quick glance at time results in Table 2 highlight a clear pattern; denser hidden layers take more computation time. The standard deviation, however, seems to be similar throughout the different topologies, thus suggesting that the growth in time is steady.

In contrast, taking a look at the accuracies obtained for the different topologies in Table 3, it is clear that the accuracy increases as the number of neurons does, until a certain point at which it stalls - see that the difference between Hech-Nielsen and Hagan's (40 and 100 neurons, respectively), is negligible -.

It is also remarkable the difference in the uncertainty that the RNNs provide; it decreases as the neurons increase. This means that not only the predictions are better, but also they are more consistent and precise. A slight disimprovement of the results in accuracy both in absolute mean terms and in standard deviation can be found in the step from 40 to 100 neurons, although the maximum accuracy reported for 100 is greater than that of 40 neurons. At the cost of a higher computational time, this change in accuracy doesn't seem much relevant.

4.2 Architecture

Tables 4 and 5 display the results obtained for the different architectures tested, sorted in ascending order of output size; from one-to-one (1) to a configuration of one-to-many of 16 output nodes.

Table 4 displays the benchmarked time results for the different architectures. In contrast with the different configurations of the hidden layer, for different output sizes there does not seem to be any notable difference time-wise; neither on the value nor the variance.

However, the accuracy, displayed in Table 5, suffers from drastic changes and is deeply impacted by the output size.
In the first place, the accuracy results for the One-to-one architecture are extremely low; around 25%, and it's not until the hidden layer is a bit dense, with 4 or more neurons, that the

accuracy of the network doesn't reach somewhat acceptable values.
Secondly, once more reasonable values are reached, they seem to peak and start to decrease again as the hidden layer gets denser.
It seems that for RNN and this particular problem, there might be some optimal value around 4 to 8 neurons in the hidden layer which provides the best accuracy.
Finally, the variance of the accuracy decreases as the number of neurons increases. This seems to indicate that the denser the hidden layer, the more precise and consistent the results are, although not always the more accurate the prediction.

5 Conclusions

The results of the experimental part show a very strong response to both the number of neurons of the hidden layer as well as the architecture of the RNN in terms of accuracy. Time-wise, however, only the hidden layer density seems to be impacted.

As more neurons are added, the accuracy of the prediction increases but stalls after some dozens of neurons are already working. This showcases that after a certain threshold value, some hidden layer's neurons are not actively improving the performance of the network; but rahter making it worse since dense layers take more computational time at no accuracy improvement cost.

In terms of the architecture, it seems that one-to-many outperforms one-to-one, which is rather unsurprising taking into account that the latter is a vanilla implementation whilst the latter is more complex and has more capability to manipulate the input data.

In conclusion, for this particular time series problem analyzed with a simple RNN, the optimal configuration seems to be several neurons in line with the Hecht-Nielsen rule and a one-to-many architecture.

References

Hecht-Nielsen, Robert (1987). "Kolmogorov's mapping neural network existence theorem". In: *Proceedings of the international conference on Neural Networks*. Vol. 3. IEEE Press New York, NY, USA, pp. 11–14.

Ciftcioglu, Ozer and Erdinc Turkcan (1992). *Selection of hidden layer nodes in neural networks by statistical tests*. Tech. rep. Netherlands Energy Research Foundation (ECN).

Hagan, Martin T, Howard B Demuth, and Mark Beale (1997). *Neural network design*. PWS Publishing Co., pp. 469–472.

Kanellopoulos, Ioannis and Graeme G Wilkinson (1997). "Strategies and best practice for neural network image classification". In: *International Journal of Remote Sensing* 18.4, pp. 711–725.

Teoh, Eu Jin, Kay Chen Tan, and Cheng Xiang (2006). "Estimating the number of hidden neurons in a feedforward network using the singular value decomposition". In: *IEEE Transactions on Neural Networks* 17.6, pp. 1623–1629.

Xu, Shuxiang and Ling Chen (2008). "A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining". In.

Stathakis, Dimitris (2009). "How many hidden layers and nodes?" In: *International Journal of Remote Sensing* 30.8, pp. 2133–2147.

Jayne, Chrisina, Andreas Lanitis, and Chris Christodoulou (2011). "Neural network methods for one-to-many multi-valued mapping problems". In: *Neural Computing and Applications* 20, pp. 775–785.

Karsoliya, Saurabh (2012). "Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture". In: *International Journal of Engineering Trends and Technology* 3.6, pp. 714–717.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *nature* 521.7553, pp. 436–444.

Innes, Mike (2018). "Flux: Elegant machine learning with Julia". In: *Journal of Open Source Software* 3.25, p. 602.

Krasovsky, Valentin et al. (2018). "Neural network technique when distribution of vehicle component parts on the technological repair routes, taking into account their technical condition". In: *MATEC Web of Conferences*. Vol. 170. EDP Sciences, p. 05011.

Manaswi, Navin Kumar and Navin Kumar Manaswi (2018). "Rnn and lstm". In: *Deep Learning with Applications Using Python: Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras*, pp. 115–126.

Chen, Jiefu et al. (2019). "Learning one-to-many stylised Chinese character transformation and generation by generative adversarial networks". In: *IET Image Processing* 13.14, pp. 2680–2686.

Di Gangi, Mattia A, Matteo Negri, and Marco Turchi (2019). "One-to-many multilingual end-to-end speech translation". In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, pp. 585–592.

World Health Organization (2020). *Listings of WHO's response to COVID-19*. `https://www.who.int/news/item/29-06-2020-covidtimeline`. Accessed: 2023-April-12.

Wang, Ye et al. (2022). "Semantic-aware conditional variational autoencoder for one-to-many dialogue generation". In: *Neural Computing and Applications* 34.16, pp. 13683–13695.

Hidden Layer (n.d.). `https://www.techopedia.com/definition/33264/hidden-layer-neural-networks`. Accessed: 2023-05-28.

The Unreasonable Effectiveness of Recurrent Neural Networks (n.d.). http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Accessed: 2023-05-28.