Multifunctionality in Recurrent neural networks based on LSTMs

Ian Emerik Leko

September 2022

Declaration of Authorship

This is to certify that the work I am submitting is my own and has not been submitted for another degree, either at University College Cork or elsewhere. All external references and sources are clearly acknowledged and identified within the contents. I have read and understood the regulations of University College Cork concerning plagiarism and intellectual property.

Acknowledgements

I would first like to thank both of my supervisors Andrew Keane and Brendan Byrne who's input and guidance and input have been invaluable thought this project. I would also like to thank Brendan for the resources and data provided through TOMRA as this project would not have been possible without it. A special thanks goes to Jim Henry and Kate O'Byrne at TOMRA who have been great helps and fantastic resources while trying to understand the data and systems. A final gratitude goes to my family who have supported me throughout the MSc program.

Summary

Biological neural networks are inherently multifunctional - they can exhibit multiple patterns based on the needs of the environment. In this thesis, we ask ourselves whether we can replicate such behaviour using the LSTM neural networks. Previous work has been established that this effect can be observed in Reservoir Computers, and we extend these results to LSTMs.
Keywords: Long-Short Term Memory; Neural Networks; Dynamical Systems; Limit Cycle
In this report we will do the following

1. Briefly describe the motivation behind the multifunctionality
2. Review the previous work done with Reservoir Computers
3. Explain LSTMs in depth
4. Review modelling of dynamical systems with LSTMs
5. Finally, explore multifunctionality using LSTMs

PART I - Introduction

Motivation

In this section we will describe the concept of multifunctionality and the relevance to the audience.

Biological Perspective of Neural Networks

"A basic tenet of neuroscience is that the ability of the brain to produce complex behaviors such as sensory perception or motor control arises from the interconnection of neurons into networks or circuits." Getting [1989]. Such networks have been a source of inspiration for *artificial neural networks* (ANN). Similarly to a biological neuron, which "...receives multiple signals through the synapses contacting its dendrites and sends a single stream of action potentials out through its axon...", Kriegeskorte and Golan [2019], an *artifical neuron* is a unit that combines multiple inputs and provides a single discernible output. Such units can be combined in networks of arbitrary design. A sketch of a basic ANN is shown in Figure 1. Each layer's units connect to each of the next layer's units, and units themselves first calculate the linear combination of the inputs, and then perform a non-linear *activation* on that linear combination. A network of this model, with at least one *hidden* layer - i.e. a layer in between input and ouput layers, can be shown to be a *universal approximator*, i.e. an algorithm that can approximate arbitrary functions, Scarselli and Chung Tsoi [1998].

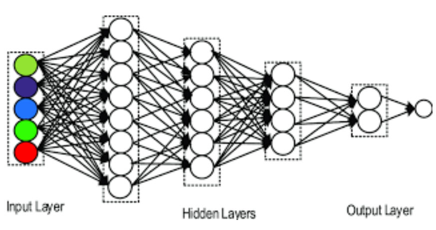


Figure 1: Sketch of an ANN with hidden layers, Dumor and Li [2019]

Biological Perspective of Multifunctionality

We will follow Flynn et al. [2021] in defining the multifunctionality as the networks "... of neurons whose activity patterns can change on the demand of performing a given duty, but synapses remain fixed." In other words, a multifunctional network can perform multiple tasks, without changing its inner structure. Such a definition is biologically motivated: even intuitively, we have a single brain which can both, say, write a poem and drive a bike. To be more specific, for example, Briggman and Kristan [2006] examined the neural activation of a medicinal leach when either crawling or swimming. The key finding was that the motions are driven both by multifunctional and dedicated circuitry. Even more interestingly, the overlap between neurons for the two motions is great: 93% of the neurons that activated when swimming also activated when crawling. One of the proposed explanations is that "...a single network driven at two different frequencies could generate motor patterns with different phase relationships without recruiting any additional neurons." (Briggman and Kristan [2006]). Furthermore, Esch et al. [2002] showed that this switch in the behaviour happens according to the external stimuli, in this case the salinity level.

PART II - Long-Short Term Memory Neural Networks

Introduction

We assume the reader is familiar with the concept of neural networks and the intricancies of ANNs (i.e. the vanilla NNs). In this part we will first introduce and describe recurrent neural networks (RNNs) as following:

1. Motivate development of recurrent neural networks (RNNs)
2. Mathematically describe basic RNNs
3. Motivate development of long-short term memory neural networks (LSTMs)
4. Mathematically describe LSTMs

Then, we will describe the computational representation of such networks via *computational graphs* in the following order:

1. Introduce the idea of computational graphs, forward- & backward-propagation
2. Represent RNNs as a computational graph
3. Represent LSTMs as a computational graph

Finally, we will explore LSTM impementation in `Tensorflow 2.0` and `Keras` packages. This will serve as the foundation for the further talk on multifunctionality in LSTMs.

Recurrent Neural Networks

Motivation

ANNs, while universal function approximators, do not take into account relationship between inputs, most notably the temporal relationship. For example and following Aggarwal [2018], suppose we were to pass the sentences

`The cat chased the mouse` (1)

`The mouse chased the cat` (2)

to an ANN. Each word would be an input and the network would think of the two sentences as being the same. However fine this might be for a simpler task such as classification, it is missing the nuance required for more complicated tasks such as machine translation. Thus for more complicated tasks when working with sequential data, we want to encapsculate that sequentiallity in our model.

History

Recurrent neural networks, as in the neural networks with feedback/self-looping structure have been present in the reasearch since 1980s, for example in Rumelhart et al. [1985] and Jordan [1986] which investigated network structures suitable for sequential data, especially using the recurrent links to provide the network with dynamic memory. Then Elman [1990] published the seminal *Finiding Structure in Time* which provided for a reference RNN model for the time to come. In the next section we will describe the *Elman* model mathematically.

Mathematics

For this section we will describe how a classic, or Elman, RNN operates. Diagram of such an RNN is shown in figure 2. The network has a inner variable, the *hidden state*, denoted by h_0. Usually it is initialised to a vector of 0s. Then it is successively combined with inputs $x_1, \ldots, x_n$ to obtain a new hidden state. Finally, each hidden state can produce an output of its own. Sometimes this is useful, but sometimes we only care about the final output after we have run through all the inputs (for example, in case of classification).
Mathematically, we start with variables

$$h_0, \quad x_1, \quad \ldots, \quad x_n$$

each representing a vector in $\mathbb{R}^d$ for some arbitrary dimension d. Then, at each time step t correspoding to the input x_t we can calculate the new hidden state h_t and the correspoding output y_t as

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1}) \quad (3)$$
$$y_t = W_{hy}h_t \quad (4)$$

where

$$W_{xh}, \quad W_{hh}, \quad W_{hy}$$

are inner parameter matrices that are learnable. σ represents an arbitrary non-linear activation function.

Long Short Term Memory

Motivation & History

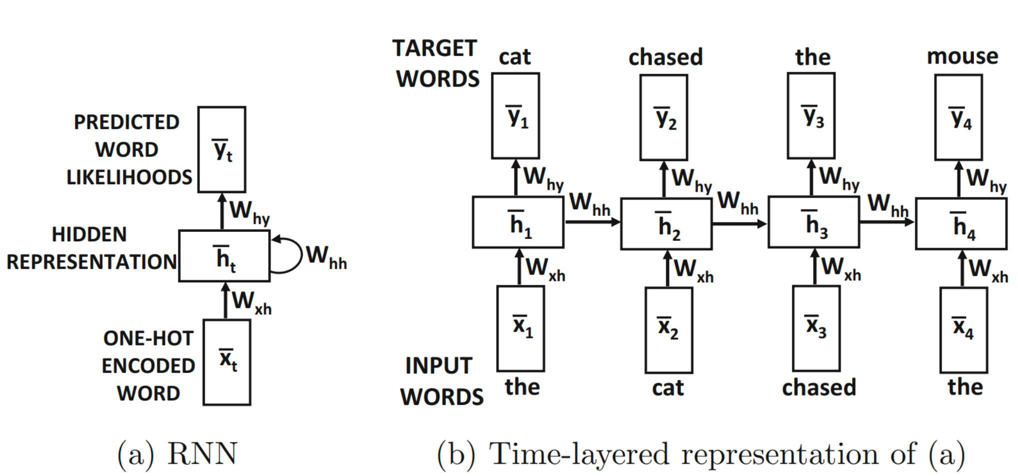
RNNs have been notorious for being hard to train, especially over long sequences with the standard backpropagation techniques for learning: Bengio et al. [1994] provide detailed overview of the problem and rudimentary solutions.Usually the problems crystalise in one of the two ways: gradient vanishing and gradient exploding (see Chapter 7 of Aggarwal [2018] for an overview). Multiple solutions have been proposed, but the most popular one has been the Long Short Term Memory variant introduced by Hochreiter and Schmidhuber [1997]. In fact, "Almost all exciting results based on RNNs have been achieved by LSTM, and thus it has become the focus of deep learning." (Yu et al. [2019]). The "vanilla" LSTM we will describe in detail in the following sections is assumed to be the original LSTM with addition of forget gate and with peephole connections (Van Houdt et al. [2020]). This format has been the upgrade of the original model by one of the inventors in Gers et al. [2000]. The

Mathematics

Figure 3 shows diagram of a vanilla LSTM block. Note that, compared with Elman RNN, we have one more hidden variable, denoted by c (cell state). The key innovation of LSTM is that we control information flow to-and-from the cell state with carefully devised operations.

Again we start with inner states and the input variables

$$c_0, \quad h_0, \quad x_1, \quad \ldots, \quad x_n$$



(a) RNN (b) Time-layered representation of (a)

Figure 2: RNN and its temporal representation (Aggarwal [2018])

where

$$c_0 \in \mathbb{R}^C, \; h_0 \in \mathbb{R}^H, \text{ and } x_i \in \mathbb{R}^X \; \forall i \in \{1, \dots, n\}$$

Following Yu et al. [2019], the formulas for the gates and the hidden states at the time step t, corresponding to the input x_t, are:

$$f_t = \sigma\left(W_{fh}h_{t-1} + W_{fx}x_t + P_f \odot c_{t-1} + b_f\right) \tag{5}$$
$$i_t = \sigma\left(W_{ih}h_{t-1} + W_{ix}x_t + P_i \odot c_{t-1} + b_i\right) \tag{6}$$
$$\tilde{c}_t = \tanh\left(W_{\tilde{c}h}h_{t-1} + W_{\tilde{c}x}x_t + b_{\tilde{c}}\right) \tag{7}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{8}$$
$$o_t = \sigma\left(W_{oh}h_{t-1} + W_{ox}x_t + P_o \odot c_t + b_o\right) \tag{9}$$
$$h_t = o_t \odot \tanh(c_t) \tag{10}$$

where the $\odot$ represents the Hadamard product (the element-wise multiplication). Now we have

$$P_f, \quad P_i, \quad P_o$$

as the peephole weights for the forget, input and the output gate respectively. Similarly to before,

$$W_{fh}, \quad W_{fx}, \quad W_{ih}, \quad W_{ix}, \quad W_{\tilde{c}h}, \quad W_{\tilde{c}x}, \quad W_{oh}, \quad W_{ox}$$

are the trainable inner matrices and

$$b_f, \quad b_i, \quad b_{\tilde{c}}, \quad b_o$$

represent the trainable biases.

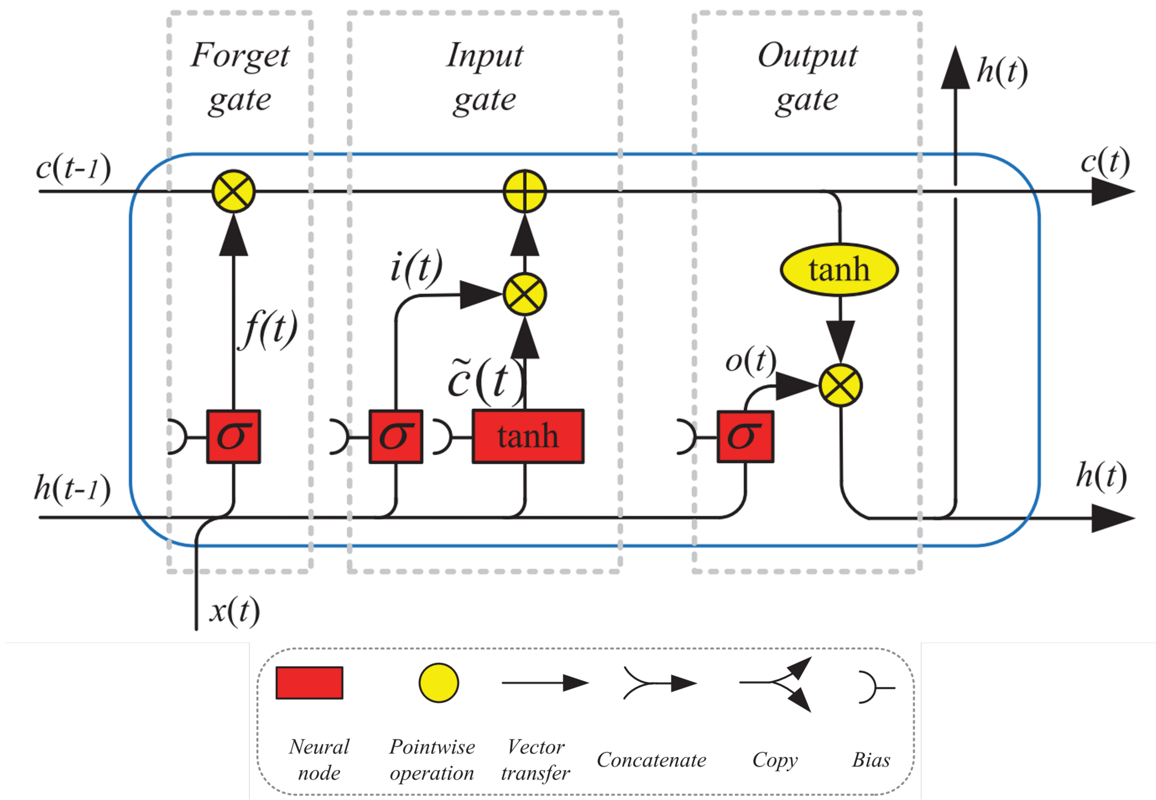


Figure 3: Vanilla LSTM cell (Yu et al. [2019])

LSTMs in `tensorflow`

In this section we describe how the LSTMs are implemented as part of the *Tensorflow 2.0* and *Keras* libraries for the Python programming language. We will discuss how the parameters correspond to the mathematics discussed in the previous sections.

Tensorflow 2.0 & Keras

"TensorFlow is an interface for expressing machine learning algorithms and an implementation for executing such algorithms." (Abadi et al.). On top of `tensorflow`, Keras (Chollet et al. [2015]) provides top-line functionality for quick creation and maintainance of commonly used neural network types, including LSTMs. Let us look at the parameters for the `keras` implementation (whole list is shown in Figure 4). LSTM is incorporated directly in `keras` as a layer found in `tensorflow.keras.layers.LSTM` as seen in the Figure 4.

1. `units` - number of separate LSTM networks that the layer models. It is one of the dimensions of the output
2. `activation` -

```
tf.keras.layers.LSTM(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    time_major=False,
    unroll=False,
    **kwargs
)
```

Figure 4: Keras LSTM class parameters (Team)

PART III - Basics of Dynamical Systems

Introduction

PART IV - Modelling Dynamical Systems using LSTMs

Introduction

Bibliography

References

Peter A. Getting. Emerging principles governing the operation of neural networks. *Annual review of neuroscience*, 12(1):185–204, 1989.

Nikolaus Kriegeskorte and Tal Golan. Neural network models and deep learning. *Current Biology*, 29(7):R231–R236, April 2019. ISSN 0960-9822. DOI:10.1016/j.cub.2019.02.034.

Koffi Dumor and Yao Li. Estimating China's Trade with Its Partner Countries within the Belt and Road Initiative Using Neural Network Analysis. *Sustainability*, 11:1449, March 2019. DOI:10.3390/su11051449.

Franco Scarselli and Ah Chung Tsoi. Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results. *Neural Networks*, 11(1):15–37, January 1998. ISSN 0893-6080. DOI:10.1016/S0893-6080(97)00097-X.

Andrew Flynn, Vassilios A. Tsachouridis, and Andreas Amann. Multifunctionality in a reservoir computer. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(1):013125, 2021.

Kevin L. Briggman and William B. Kristan. Imaging Dedicated and Multifunctional Neural Circuits Generating Distinct Behaviors. *J. Neurosci.*, 26(42):10925–10933, October 2006. ISSN 0270-6474, 1529-2401. DOI:10.1523/JNEUROSCI.3265-06.2006.

Teresa Esch, Karen A. Mesce, and William B. Kristan. Evidence for Sequential Decision Making in the Medicinal Leech. *J. Neurosci.*, 22(24):11045–11054, December 2002. ISSN 0270-6474, 1529-2401. DOI:10.1523/JNEUROSCI.22-24-11045.2002.

Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, Cham, 2018. ISBN 978-3-319-94462-3 978-3-319-94463-0. DOI:10.1007/978-3-319-94463-0.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

M. I. Jordan. Serial order: A parallel distributed processing approach. Technical report, June 1985-March 1986. Technical Report AD-A-173989/5/XAB; ICS-8604, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, May 1986.

Jeffrey L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 1551-6709. DOI:10.1207/s15516709cog1402_1.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994. ISSN 1941-0093. DOI:10.1109/72.279181.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. DOI:10.1162/neco.1997.9.8.1735.

Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, July 2019. ISSN 0899-7667. DOI:10.1162/neco_a_01199.

Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A Review on the Long Short-Term Memory Model. *Artificial Intelligence Review*, 53, December 2020. DOI:10.1007/s10462-020-09838-1.

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural computation*, 12(10):2451–2471, 2000.

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. page 19.

Francois Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

Keras Team. Keras documentation: LSTM layer. https://keras.io/api/layers/recurrent_layers/lstm/.