Project 3

Edvard B. Rørnes[*] and Isak O. Rukan[†]
Institute of Physics, University of Oslo,
0371 Oslo, Norway

(Dated: December 4, 2024)

Abstracting very cool

CONTENTS

1. INTRODUCTION

2. THEORY

2.1. Gravitational Waves

2.1.1.

2.2. Recurrent Neural Networks

The Recurrent Neural Networks (RNNs) are a class of neural networks specifically designed to handle sequential data or data with temporal dependencies. Unlike traditional FeedForward Neural Networks, RNNs are capable of "remembering" information from previous time steps. This is done through the so called 'hidden state', which acts as a form of memory by retaining information about prior computations. The hidden state is essentially an array of data that is updated at each time step based on the input data and the previous hidden state. Alhough this enables RNN to access the temporal dependencies of the data at hand, it greatly increases the commputation time compared to that of the FFNN. The standard RNN consists of only one hidden layer, but it is certainly possible to have more than one hidden layer. In fact, this is commonly referred to as the stacked RNN (SRNN), and we will arrive at this neural network further down. However, firstly, we present the structure and general algorithm for the RNN.

2.2.1. Structure

The RNN processes input sequentially, with information flowing step-by-step from the input to the output. This is done with the introduction of a hidden state h_t, where the subscript denotes at time t. The network can be summarized by the following two equations [**?**]:

$$h_t = \sigma^{(h)}\left(W_{hx}X_t + W_{hh}h_{t-1} + b_h\right), \tag{1a}$$

$$\tilde{y}_t = \sigma^{(\text{out})}\left(W_{yh}h_t + b_y\right). \tag{1b}$$

Here, σ_h and σ_{out} is the activation function for the hidden layer and the output layer respectively. W_{xh} is the weight from input to hidden layer, W_{hh} the hidden layer, W_{yh} the output layer and $\tilde{y}$ the output of the RNN. Let now t be divided into a discrete set of times $(t_i)_{i\in N}$. Substituting (1a) into itself recursively leads to a formula for computing h_{t_n}:

$$\begin{aligned} h_{t_n} = \sigma^{(h)}\Bigg(& W_{hx}X_{t_n} + W_{hh}\sigma_h\Bigg(W_{hx}X_{t_{n-1}} \\ &+ W_{hh}\sigma_h\left(\cdots + b_h\right) + b_h\Bigg) + b_h\Bigg) \end{aligned} \tag{2}$$

This shows that the hidden state at time t_n is dependent on the input X_t for $t \in [0, t_n]$, i.e. all previous times.

2.2.2. General Algorithm

Consider some general data output y, of shape (N, p_{out}) and some data input X, of shape (N, p_{in}), where N corresponds to the total amount of time points, and p_{out}, p_{in} the dimension of the output and input, respectively. Generally, X could correspond to a quite large sampling frequency in time, making the computation of the hidden state h_t in (2) computationally demanding. One typical way of dealing with this is to split the data into 'windows' of size N_W in time. These windows should

[*] e.b.rornes@fys.uio.no
[†] Insert Email

generally overlap, such that no temporal dependencies across windows are left out.

Splitting the data into windows, we define the hidden state for window n as:

$$\begin{aligned} h_n &= \sigma^{(h)}\left(W_{hx}X_n + W_{hh}h_{n-1} + b_h\right) \\ &\equiv \sigma^{(h)}(z_n) \end{aligned} \tag{3}$$

where X_n is the n-th window.

2.2.3. Backpropagation Through Time

The error between y and the predicted output $\tilde{y}$, is given by some chosen loss function $L(y, \tilde{y})$,

$$L(y, \tilde{y}) = \frac{1}{N}\sum_{n=1}^{N} l(y_n, \tilde{y}_n), \tag{4}$$

where l is some error-metric. For some learning rate η, the standard update rule for the weights and biases is given by:

$$W \leftarrow W - \eta\frac{\partial L}{\partial W},\ b \leftarrow b - \eta\frac{\partial L}{\partial b}. \tag{5a}$$

This transformation may be extended using optimization methods aimed at handling exploding gradient, faster convergence, avoiding local minimas, etc. We covered three of these optimization methods in [**?**]; the root mean squared propagation (RMSprop), the adaptive gradient (AdaGrad) and the adaptive moment estimation (Adam).

Compared to FFNN, computing the gradient of L with respect to the weights leads to a somewhat more complicated expression. Consider now the partial derivative of the loss function with respect to the weight W (W_{hx} or W_{hh}):

$$\begin{aligned} \frac{\partial L}{\partial W} &= \frac{1}{N}\sum_{n=1}^{N}\frac{\partial l(y_n, \tilde{y}_n)}{\partial W} \\ &= \frac{1}{N}\sum_{n=1}^{N}\frac{\partial l(y_n, \tilde{y}_n)}{\partial \tilde{y}_n}\frac{\partial \tilde{y}_n}{\partial z_n}\frac{\partial z_n}{\partial W}, \end{aligned} \tag{6}$$

where ($\odot$ represents the Hadamard product)

$$\begin{aligned} \frac{\partial z_n}{\partial W} &= \frac{\partial}{\partial W}\left(W_{hx}X_n + W_{hh}h_{n-1} + b_n\right) \\ &= X_n^T \odot \frac{\partial W_{hx}}{\partial W} + h_{n-1}^T \odot \frac{\partial W_{hh}}{\partial W} + W_{hh} \odot \frac{\partial h_{n-1}}{\partial W}, \end{aligned} \tag{7}$$

and

$$\frac{\partial h_{n-1}}{\partial W} = \sigma_h'(z_{n-1}) \odot \left(X_n^T \odot \frac{\partial W_{hx}}{\partial W} + h_{n-1}^T \odot \frac{\partial h_{n-2}}{\partial W}\right), \tag{8}$$

meaning that

$$\begin{aligned} \frac{\partial z_n}{\partial W} &= X_n^T \odot \frac{\partial W_{hx}}{\partial W} + h_{n-1}^T \odot \frac{\partial W_{hh}}{\partial W} \\ &\quad + W_{hh} \odot \left(\sigma'(z_{n-1}) \odot \left(X_n^T \odot \frac{\partial W_{hx}}{\partial W}\right) \right.\\ &\quad \left. + h_{n-1}^T \odot \left(\sigma'(z_{n-2}) \odot \left(X_n^T \odot \frac{\partial W_{hx}}{\partial W} + \dots\right)\right)\right). \end{aligned} \tag{9}$$

There is the option of computing this recursively. However, the more common approach is to instead 'unfold' the RNN through time. This is known as the method of backpropagation through time (BTT). Here, we define the quantities

$$\delta_n^{\text{out}} \equiv \frac{\partial l(y_n, \tilde{y}_n)}{\partial \tilde{y}_n}\frac{\partial \tilde{y}_n}{\partial z_n} = \frac{\partial l(y_n, \tilde{y}_n)}{\partial \tilde{y}_n}\sigma_{\text{out}}'(z_n), \tag{10a}$$

$$\delta_N \equiv \delta_n^{\text{out}}\frac{\partial l(y_N, \tilde{y}_N)}{\partial \tilde{y}_N}, \tag{10b}$$

$$\delta_n \equiv \left(\delta_n^{\text{out}}W_{hx}^T + \delta_{n+1}W_{hh}^T\right) \odot \sigma_h'(z_n). \tag{10c}$$

This makes it possible to iterate backwards, starting from the last time window N, and the gradients of the weights and biases are then given by

$$\frac{\partial L}{\partial W_{hx}} = \frac{1}{N}\sum_{n=1}^{N}\delta_n^{\text{out}}X_n^T, \tag{11a}$$

$$\frac{\partial L}{\partial W_{hy}} = \frac{1}{N}\sum_{n=1}^{N}\delta_n^{\text{out}}h_n^T, \tag{11b}$$

$$\frac{\partial L}{\partial W_{hh}} = \frac{1}{N}\sum_{n=1}^{N}\delta_n h_{n-1}^T, \tag{11c}$$

$$\frac{\partial L}{\partial b_y} = \frac{1}{N}\sum_{n=1}^{N}\delta_n^{\text{out}}, \tag{11d}$$

$$\frac{\partial L}{\partial b_h} = \frac{1}{N}\sum_{n=1}^{N}\delta_n. \tag{11e}$$

The dependency for each error term δ_n on 'future' error terms leads to a much greater computation time, compared to that of FFNN. Every gradient computation need an additional propagation through all time-windows. This can lead to gradients blowing up due to only (relatively) minor errors. However, there are multiple ways of resolving this issue. Perhaps the most obvious one is to simply truncate the amount of terms in the algorithm summerized in (10a), commonly referred to as 'truncated backpropagation through time' (see e.g. [**?**]). Apart from that it is an actual simplification of (**??**), it has the immediate consequence of ignoring long-term dependencies of the data, which in some cases is just the type of information you do not want your model to train on.

Implementing the stacked RNN is then done by essentially creating a hidden state for each 'stack' of RNN.

The output of the stacked RNN is computed by feeding the hidden states to each other in succession, starting from the first hidden layer. The hidden states in some time window n are given by

$$h_n^l = \begin{cases} \sigma^{(h)}\left(W_{hx}^1 X_n + W_{hh}^1 h_{n-1}^1 + b_h^1\right), & l = 1, \\ \sigma^{(h)}\left(W_{hx}^l h_n^{l-1} + W_{hh}^l h_{n-1}^l + b_h^l\right), & l \geq 2, \end{cases} \tag{12}$$

and the output of the stacked RNN in time window n as

$$\tilde{y}_n = \sigma^{(\mathrm{out})}\left(W_{yh} h_n^L + b_y\right). \tag{13}$$

Here, the dimensions are $W_{hx}^l \in \mathbb{R}^{d_l \times d_{l-1}}$, $W_{hh}^l \in \mathbb{R}^{d_l \times d_l}$, with d_l being the dimension of the l-th hidden state, l_0 the dimension of the input and l^L the dimension of the output. The BTT algorithm for a stacked RNN takes on the same form, except that we now have L hidden states.

2.2.4. Gradient Clipping

A common method for dealing with exploding gradients, is the method of gradient clipping (see e.g. [**?**]). This method prevents checks whether the magnitude of the gradient is moving past a certain threshold. If this is true it truncates the current gradient. This can be summarized as:

$$\nabla L \rightarrow \frac{\epsilon}{||\nabla L||} \nabla L \text{ if } ||\nabla L|| > \epsilon. \tag{14}$$

3. IMPLEMENTATION

4. DISCUSSION

5. CONCLUSION

Test bib [**?**]