Recurrent Deep Learning Models and its applications

Ngai Ho Wang

February 8, 2025

Contents

1 Introduction

With the rise of the Generative Artificial Intelligence, the development of AI has already made remarkable strides in processing sequential data. In understanding and producing sequential data. It has applications ranging from Natural Language Processing (NLP) to music composition to video generation. Especially NLP, has emerged as a pivotal field in artificial intelligence, enable machines to understand, interpret and generate in human readable format. Some famous Artificial Intellience assistence for example, Siri, Alexa and Bixby have shown the possibility. Everyone can communicate with those machines, which make the reasonable response back to user.

Recurrent Neural Networks (RNNs) have been a foundational architecture in this domain, Unlike the traditional Artificial Neural Network, RNNs do not treat each input independently, RNNs handle each input by considering the information from previous inputs. Conceptually this architecture able to retain the information. Thus, this architecture is suitable for handling sequential data. Unfortunately, early RNNs had limitation in training of networks over long sequence. vanishing and exploding gradient problems significantly affect the training process of RNN (Bengio et al., 1994). Eliminating many practical applications of RNNs. After that, (Hochreiter & Schmidhuber, 1997) introduced Long Short-Term Memory (LSTM) networks and are responsible for the breakthrough in how to solve these challenges. Specificized gating mechanisms were introduced in LSTMs to regulate the flow of the information, minimize the vanishing gradient problem and learn the long-term dependencies. This advanced made RNNs much more performant on tasks like a language modeling, machine translation and speech recognition tasks.

Further improvements were achieved with Gated Recurrent Units (GRUs) by (Cho et al., 2014) which diminished the LSTM architecture's complexity, but still provided the same performance. GRUs performed comparably but used fewer parameters, making it computationally and more tractably trainable.

Since the craze of AI has been revived by generative AI, natural language processing to time series prediction and speech recognition have once again aroused people's interest in RNN. This report aims to:

- Explore the theoretical foundations of recurrent deep learning models.
- Investigate their diverse applications in solving sequential data tasks.
- Analyze their performance, strengths, and inherent limitations.

2 Background and Literature Review

2.1 Evolution of Deep Learning and Recurrent Neural Networks

In the past few decades, thank to the rapidly development of technology, the computing resource has a incredible increase. Thus, substantially deep learning architecture have improved, from simple architectures, which only able to capture simple information from data to sophisticated models that are able to learn complex, abstract representations. This was before the early neural networks like perceptrons and multilayer perceptrons (MLPs) laid the footwork of neural computation that first came in the picture, but were burdened by the lack of ability to model sequential dependencies. This however imposed a limit on the feed forward paradigm, which prompted the development of recurrent neural networks (RNNs) that extend the old stalactite of feed forward paradigm with cyclic connections. Through these connection, RNNs are capable to keep a hidden state that represents information over time steps thereby effectively capture temporal dynamics. RNNs have been a decisive step in the evolution of deep learning, as they are able to do tasks that require memory of previous events, including problems of natural language processing and time series modeling. Despite that, early RNNs models suffered from serious problems for example, vanishing gradients and exploding gradients, which prevented these RNN models from learning long ranged dependencies. This stimulated the building of more refined architectures intended to side step these obstacles.

2.2 Literature Review

2.2.1 Backpropagation Through Time

BPTT is one of the most important algorithms used for training RNNs. Dating back to the original effort to expand the typical backpropagation algorithm, BPTT has been formulated to handle the difficulties of temporal sequences that are inherent in sequential data (Werbos, 1990). This algorithm allows RNNs in learning sequence dependent data by unfold the network over time steps and then updating weights matrix through the gradient of loss function with respect to the variable (Rumelhart et al., 1986).

Conceptual Framework of BPTT

BPTT works based on the technique of treating an RNN as a deep feedforward network for across multiple time steps. In the forward pass, the RNN, like other artificial neuronal network, applies operation over the data input in sequence, bringing changes in its own state variables at every time step, depending on the input and the previous state of its general working state or hidden state. This sequential processing produces outputs and stores the internal states of the network in any period (Werbos, 1990).
This unfolds the RNN to construct a traditional Feedforward Neural Network where we can apply backpropagation through time. Below is the conceptual idea of BPTT in RNN.

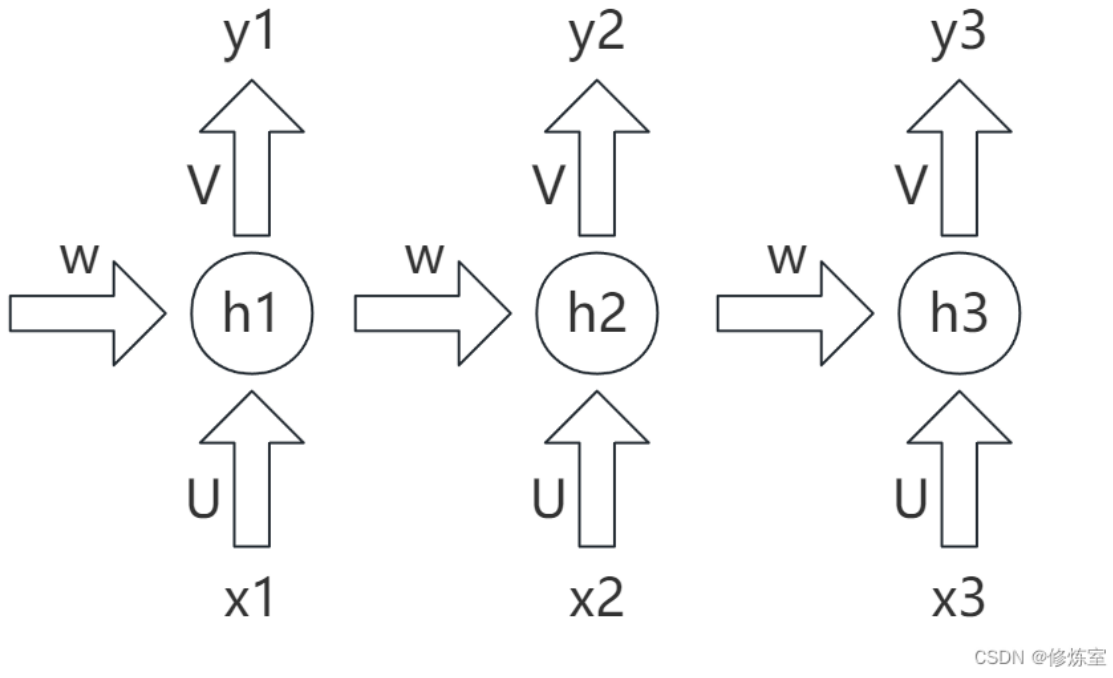


Figure 1: Unfolded RNN

Notation	Meaning	Dimension
U	Weight matrix for input to hidden state	$input\ size \times hidden\ unites$
W	Weight matrix for hidden to hidden state	$hidden\ units \times hidden\ unites$
V	Weight matrix for hidden state to output state	$hidden\ units \times number\ of\ class$
x_t	Input vector at time t	$input\ size \times 1$
h_t	Hidden state output at time t	$hidden\ units \times 1$
b_h	Bias term for hidden state	$hidden\ units \times 1$
b_y	Bias term for output state	$number\ of\ class \times 1$
$\hat{o}_y$	Output at time t	$number\ of\ class \times 1$
$\hat{y}_t$	Output at time t	$hidden\ units \times 1$
$\mathcal{L}$	Loss at time t	$scalar$

Table 1: Unfolded RNN

Forward Pass

During the forward pass, the RNN processes the input sequence sequentially, computing hidden states and output at each timestep:

$$h_t = f(U^T x_t + W^T h_{t-1} + b_h) \tag{1}$$

$$\hat{y}_t = f(V^T h_t + b_y) \tag{2}$$

Computing the loss function

Assuming the loss is computed only at the final timestep t:

$$\mathcal{L}_t = L(y_t, \hat{y}_t) \tag{3}$$

In order to do backpropagation through time to tune the parameters in RNN, we need to calculate the partial derivative of loss function $\mathcal{L}$ with respect to the differently parameters.

Backward pass using the chain rule

Using the chain rule for computing the gradient.
Partial derivative of loss function $\mathcal{L}$ with respect to W (hidden to hidden state) at time 2.

$$\frac{\partial \mathcal{L}_2}{\partial W} = \frac{\partial \mathcal{L}_2}{\partial y_2} \cdot \frac{\partial y_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W} \tag{4}$$

By mathematic induction

$$\frac{\partial \mathcal{L}_t}{\partial W} = \frac{\partial \mathcal{L}_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} \cdot (\sum_{i=1}^{t} \frac{\partial h_t}{\partial h_i} \cdot \frac{\partial h_i}{\partial W}) \tag{5}$$

Where

$$\frac{\partial h_t}{\partial h_i} = \prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \tag{6}$$

Partial derivative of loss function $\mathcal{L}$ with respect to U (input to hidden state) at time 2.

$$\frac{\partial \mathcal{L}_2}{\partial U} = \frac{\partial \mathcal{L}_2}{\partial y_2} \cdot \frac{\partial y_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial U} \tag{7}$$

By mathematic induction

$$\frac{\partial \mathcal{L}_t}{\partial U} = \frac{\partial \mathcal{L}_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} \cdot (\sum_{i=1}^{t} \frac{\partial h_t}{\partial h_i} \cdot \frac{\partial h_i}{\partial U}) \tag{8}$$

Where

$$\frac{\partial h_t}{\partial h_i} = \prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \tag{9}$$

Partial derivative of loss function $\mathcal{L}$ with respect to V (hidden to output state) at time 2.

$$\frac{\partial \mathcal{L}_2}{\partial V} = \frac{\partial \mathcal{L}_2}{\partial y_2} \cdot \frac{\partial y_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial V} \tag{10}$$

By mathematic induction

$$\frac{\partial \mathcal{L}_t}{\partial V} = \frac{\partial \mathcal{L}_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} \cdot (\sum_{i=1}^{t} \frac{\partial h_t}{\partial h_i} \cdot \frac{\partial h_i}{\partial V}) \tag{11}$$

Where

$$\frac{\partial h_t}{\partial h_i} = \prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \tag{12}$$

Partial derivative of loss function $\mathcal{L}$ with respect to b_h (bias term in hidden state) at time 2.

$$\frac{\partial \mathcal{L}_2}{\partial b_b} = \frac{\partial \mathcal{L}_2}{\partial y_2} \cdot \frac{\partial y_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial b_h} \tag{13}$$

By mathematic induction

$$\frac{\partial \mathcal{L}_t}{\partial b_h} = \frac{\partial \mathcal{L}_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} \cdot (\sum_{i=1}^{t} \frac{\partial h_t}{\partial h_i} \cdot \frac{\partial h_i}{\partial b_h}) \tag{14}$$

Where

$$\frac{\partial h_t}{\partial h_i} = \prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \tag{15}$$

Partial derivative of loss function $\mathcal{L}$ with respect to b_y (bias term in output state) at time 2.

$$\frac{\partial \mathcal{L}_2}{\partial b_y} = \frac{\partial \mathcal{L}_2}{\partial y_2} \cdot \frac{\partial y_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial b_y} \tag{16}$$

By mathematic induction

$$\frac{\partial \mathcal{L}_t}{\partial b_y} = \frac{\partial \mathcal{L}_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} \cdot (\sum_{i=1}^{t} \frac{\partial h_t}{\partial h_i} \cdot \frac{\partial h_i}{\partial b_y}) \tag{17}$$

Where

$$\frac{\partial h_t}{\partial h_i} = \prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \tag{18}$$

parameters updates

$$W \leftarrow W - \alpha \frac{\partial \mathcal{L}}{\partial W} \tag{19}$$

$$U \leftarrow U - \alpha \frac{\partial \mathcal{L}}{\partial U} \tag{20}$$

$$V \leftarrow V - \alpha \frac{\partial \mathcal{L}}{\partial V} \tag{21}$$

$$b_h \leftarrow b_h - \alpha \frac{\partial \mathcal{L}}{\partial b_h} \tag{22}$$

$$b_y \leftarrow b_y - \alpha \frac{\partial \mathcal{L}}{\partial b_y} \tag{23}$$

Pseudocode of BPTT (Wikipedia, 2023)

Algorithm 1 Backpropagation Through Time (BPTT)

Input:
  Sequence of input data $\{x_1, x_2, \ldots, x_T\}$
  Sequence of target outputs $\{y_1, y_2, \ldots, y_T\}$
  Learning rate η
  Number of time steps to unroll N
Initialize: Model parameters θ, hidden state $h_0 = 0$
Forward Pass:
for $t = 1$ to T **do**
  Compute hidden state: $h_t = f(h_{t-1}, x_t; \theta)$
  Compute output: $\hat{y}_t = g(h_t; \theta)$
  Compute loss for time step t: $L_t = \mathcal{L}(\hat{y}_t, y_t)$
end for
Backward Pass (BPTT):
Set total loss: $L = \sum_{t=1}^{T} L_t$
for $t = T$ down to 1 **do**
  Compute gradient of loss with respect to output: $\frac{\partial L_t}{\partial \hat{y}_t}$
  Backpropagate through output layer to obtain: $\frac{\partial L_t}{\partial h_t}$
  Accumulate gradients for parameters: $\frac{\partial L}{\partial \theta}$
  for $k = 1$ to N **do**
    Backpropagate through time for N steps:
    Compute gradient contribution from step $t - k$: $\frac{\partial L_t}{\partial h_{t-k}}$
  end for
end for
Update Parameters:
$\theta = \theta - \eta \cdot \frac{\partial L}{\partial \theta}$
Output: Updated parameters θ

2.2.2 Activation Function

Activation functions, particularly the sigmoid function, are fundamental components of recurrent neural networks (RNNs). They transform input data into output data. A key property of these functions is their differentiability. Differentiability is crucial for the backpropagation through time (BPTT) algorithm, enabling the application of the chain rule during training.

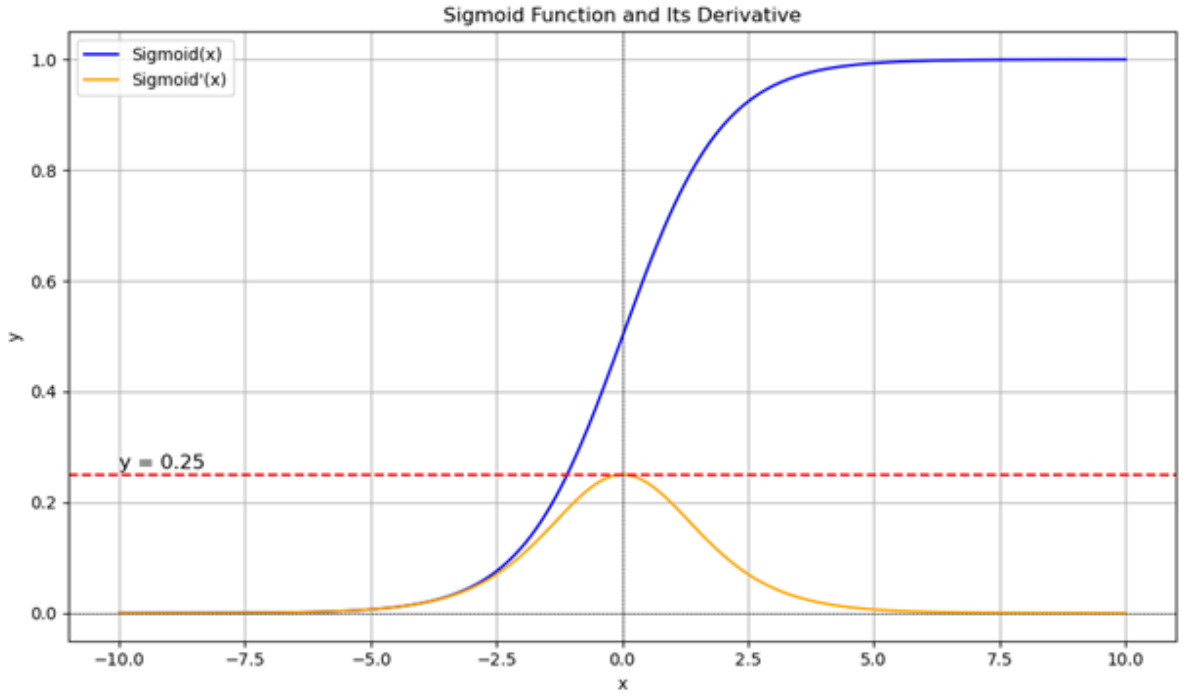
Sigmoid activation function

The main role of the sigmoid activation function is to normalize candidate values and convert the cell state to a hidden state when performing cell state updates. It limits the output between [0,1] because it has a smooth gradient, which is important for discovering long-range dependencies.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{24}$$

$$Sigmoid'(x) = Sigmoid(x)(1 - Sigmoid(x)) \tag{25}$$

Below is the sigmoid function and its derivative.



$Domain(Sigmoid(x)) = \mathbb{R}, \quad Codomain(Sigmoid(x)) = (0, 1)$
$Domain(Sigmoid'(x)) = \mathbb{R}, \quad Codomain(Sigmoid'(x)) = [0, 0.5]$

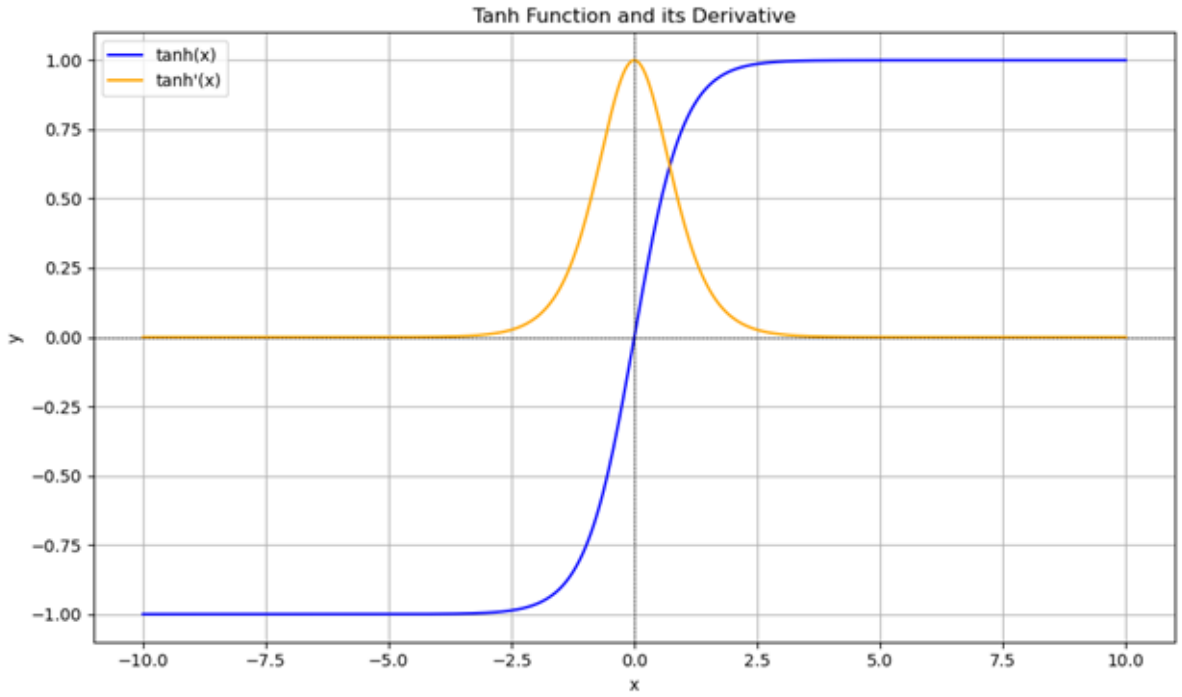
Hyperbolic tangent activation function

The main role of the hyperbolic tangent (tanh) activation function is to normalize candidate values and convert the cell state to a hidden state when performing cell state updates. It limits the output between [-1,1] because it has a stable gradient, which is important for discovering long-range dependencies.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{26}$$

$$tanh'(x) = 1 - tanh^2(x) \tag{27}$$

Below is the Hyperbolic tangent activation function and its derivative.



$Domain(tanh(x)) = \mathbb{R}, \quad Codomain(tanh(x)) = [-1, 1]$
$Domain(tanh'(x)) = \mathbb{R}, \quad Codomain(tanh'(x)) = [0, 1]$

Gradient vanishing and gradient exploring

When training the RNN, BPTT was used to update the weight matrix. As the number of time steps increase, the problem of gradient instability of often encountered, and this problem is gradient vanishing and gradient explored (Bengio et al., 1994).

Vanishing Gradients

Generally, sigmoid activation function is used commonly in RNNs, has a maximum derivative of 0.25. When doing BPTT in long time steps, this multiplication results in exponentially diminishing gradients as the sequence length increases. Consequently, the shallow neural receive very small gradient updates, making it difficult to adjust the parameters effectively. This leads to the model struggling to learn long time dependencies.

Exploding Gradients

When we are doing the feedforward and get super large value computed by loss function. Then when updating the parameters. The updates to the weights will also be large. Resulting in higher loss and larger gradients in the next iterations. This will lead to exploding gradients.

We have introduce the backpropagation through time. This is the method to update the parameters in RNNs. When calculating, for example, the partial derivative of loss function with respect to W. Assume the time t go to infinity large. We will get this term. $\prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$, and it will lead to exponential problem. if $\frac{\partial h_j}{\partial h_{j-1}} > 1$. Then the product of all term will increase exponentially, then exploding gradients occur. On the contrary, if $\frac{\partial h_j}{\partial h_{j-1}} < 1$. Then the result will decrease exponentially, then vanishing gradients occur.