Recurrent Deep Learning Models and its applications

Ngai Ho Wang

February 9, 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.

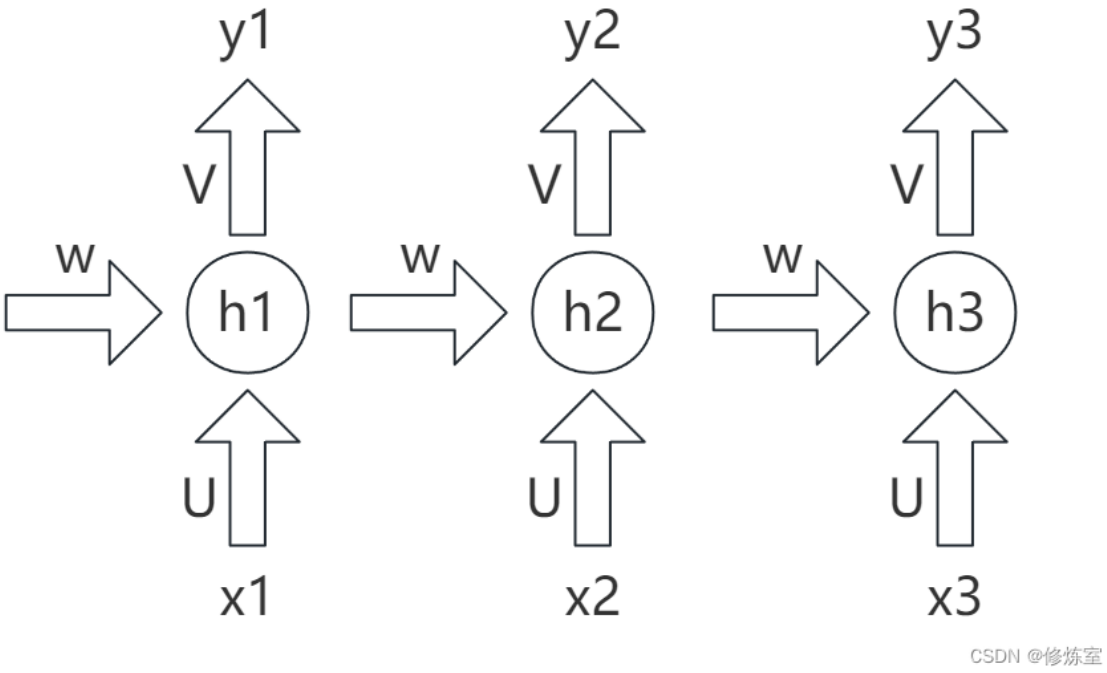


Figure 1: Unfolded RNN

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.

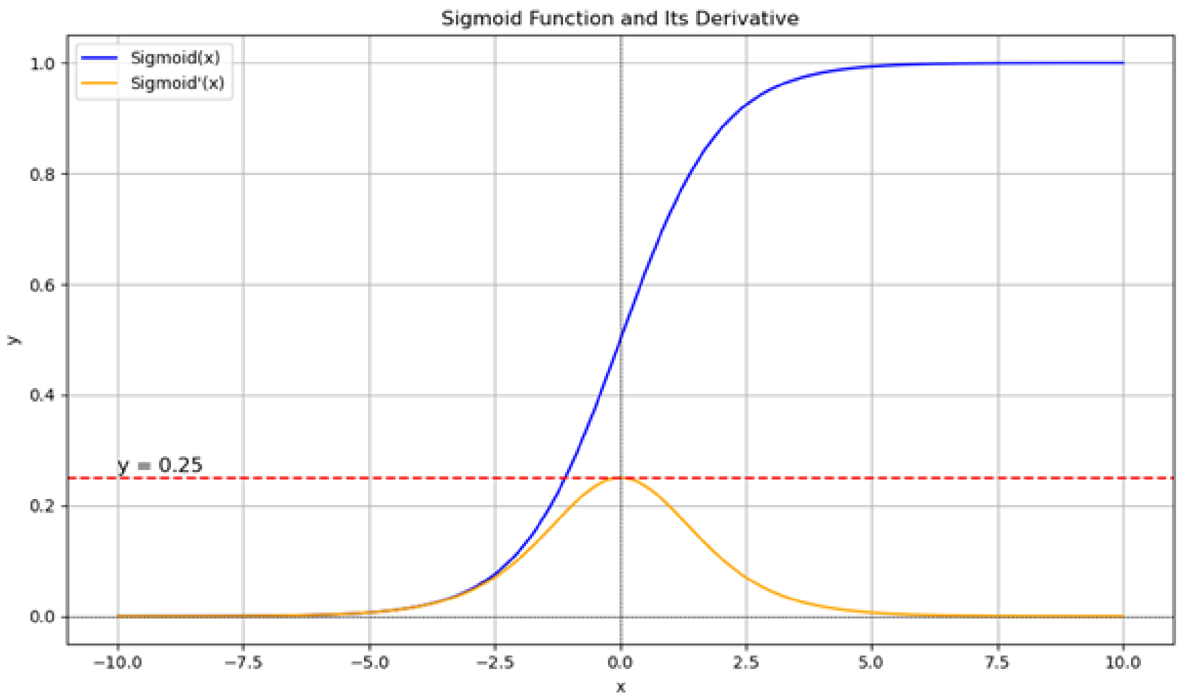
Sigmoid activation function

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.



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

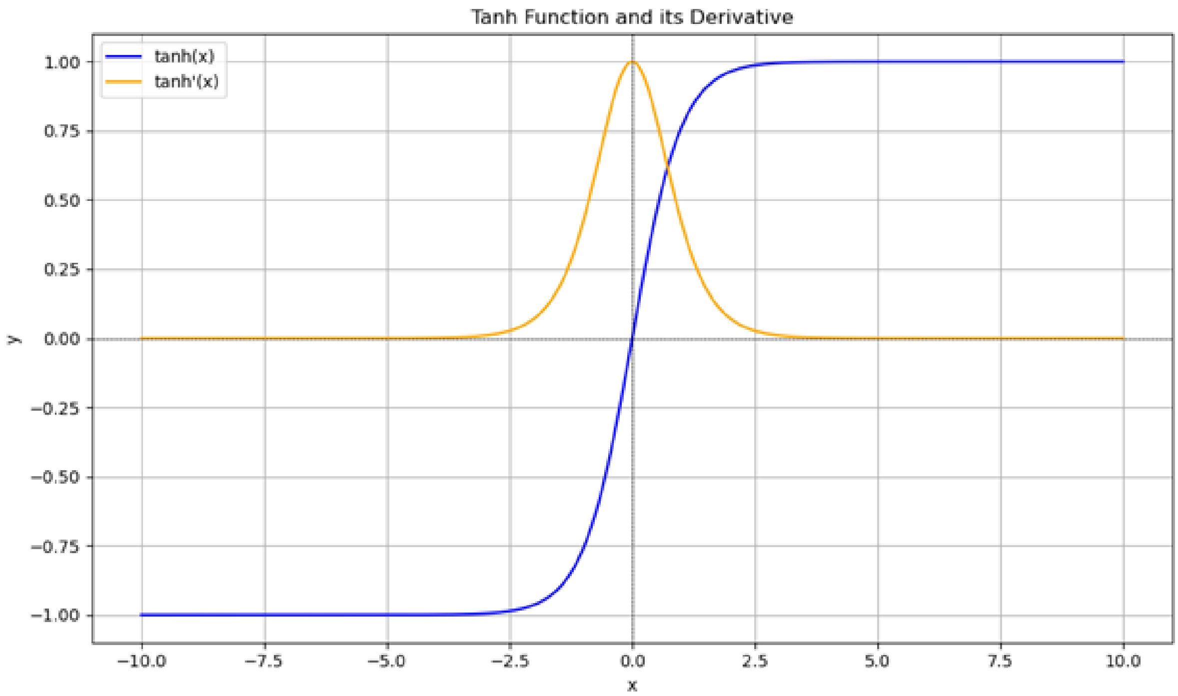
Hyperbolic tangent activation function

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.



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

2.2.3 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).

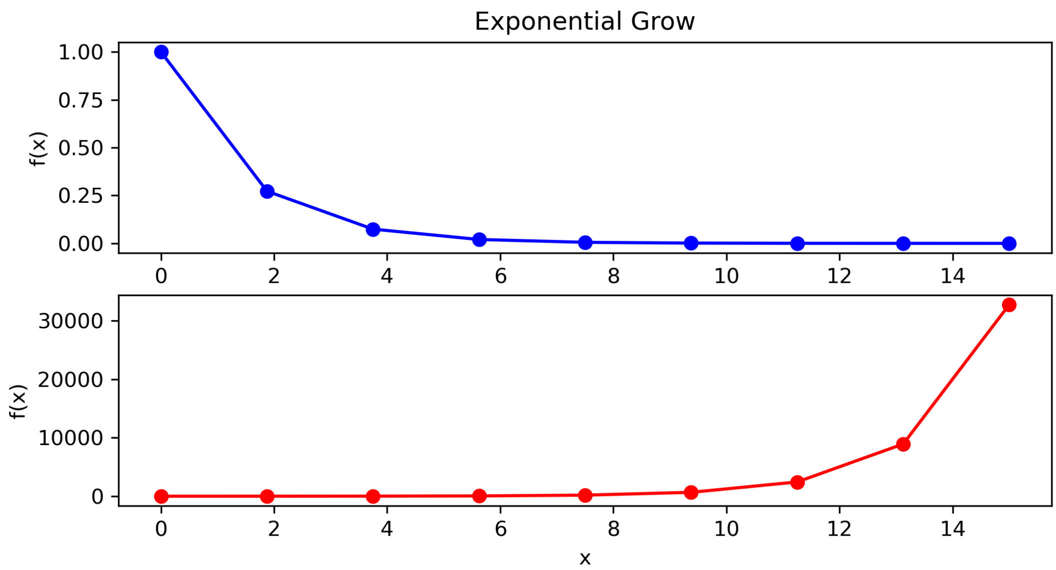
Vanishing Gradients

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.

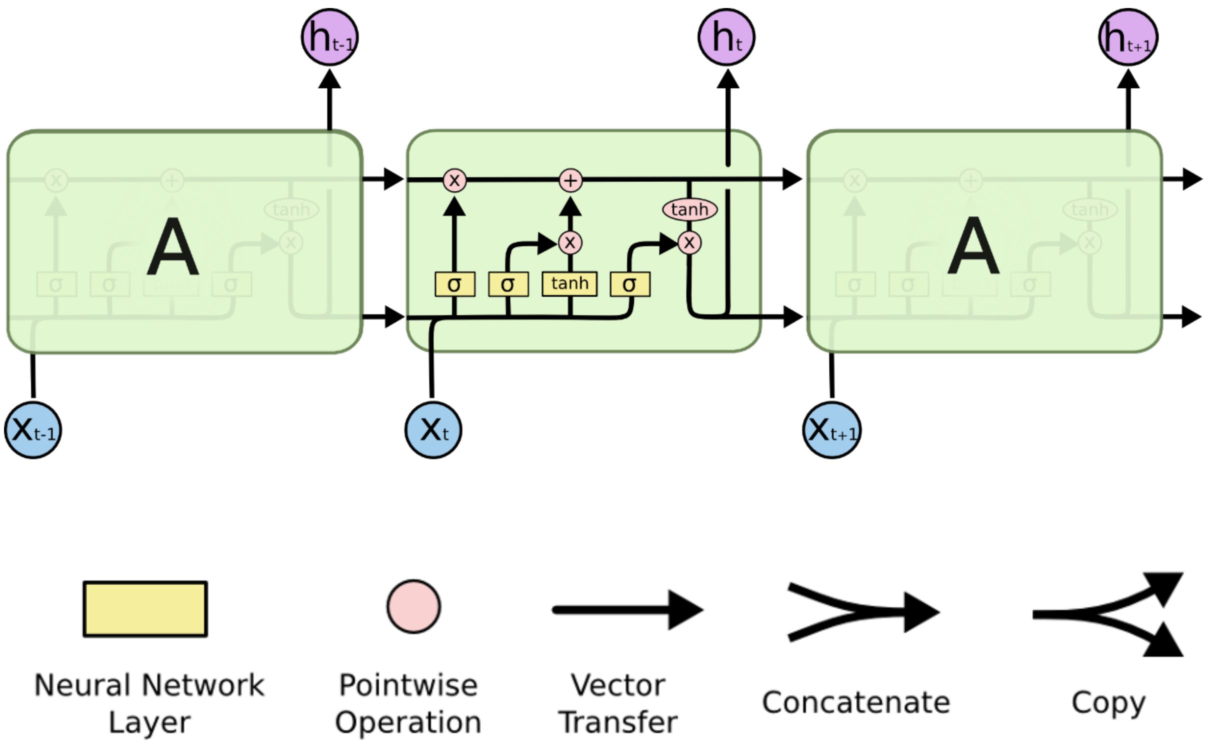


2.2.4 Long short-term memory (LSTM)

Long short-term memory proposed by (Hochreiter & Schmidhuber, 1997). LSTM is designed for handling long time step problems. The architecture of LSTM can prevent vanishing gradient and exploring gradient. The main difference between LSTM and RNN is the number of gates. LSTM introduced input, forget and output gates. This allows LSTM to manage the flow of information more effectively, retaining important information over longer sequences.

Architecture

LSTMs introduce a memory cell that can maintain information over long time steps the cell is controlled by three gates, input gate, output gate, and forget gate. Each cell of LSTMs inside has 3 sigmoid and 1 tanh layer. Below graph unfolds the LSTM hence we can analyze different gates.



Forget gate:

The forget gate is a component of the LSTM, designed to manage the flow of information within the cell state. The function of forget gate is to determine which information should be retained in memory cell (Hochreiter & Schmidhuber, 1997).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{28}$$

Input gate:

The input gate controls how much new information from the current time step is allowed to enter the cell (Hochreiter & Schmidhuber, 1997). For the $\widetilde{C}_t$, the purpose is to suggest updates for the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_c) \tag{29}$$

$$\widetilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{30}$$

Cell State Update:

The forget gate will drop the meaningless information and add some potential information.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{31}$$

Output gate:

The output gate is able to control how much or what information from the cell state should be passed to the next layer or used in predictions.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{32}$$

Hidden State Update:

The hidden state is influenced by output value and current cell state.

$$h_t = o_t * tanh(C_t) \tag{33}$$

n = number of features in the input vector x_t.
m = number of units in LSTM.

Notation	**Meaning**	**Dimension**
x_t	Input vector at time t	$n \times 1$
h_t	Hidden state output at time t	$m \times 1$
C_t	Cell state at time t	$m \times 1$
f_t	Forget gate output at time t	$m \times 1$
i_t	Input gate output at time t	$m \times 1$
o_t	Output gate output at time t	$m \times 1$
$\widetilde{C}_t$	Candidate memory cell at time t	$m \times 1$
W_f	Weight matrix for the forget gate	$m \times (m+n)$
W_i	Weight matrix for the input gate	$m \times (m+n)$
W_C	Weight matrix for the candidate memory cell	$m \times (m+n)$
W_o	Weight matrix for the output gate	$m \times (m+n)$
b_f	Bias vector for the forget gate	$m \times 1$
b_i	Bias vector for the input gate	$m \times 1$
b_C	Bias vector for the candidate memory cell	$m \times 1$
b_o	Bias vector for the output gate	$m \times 1$

Table 2: Unfolded RNN

Number of parameters:

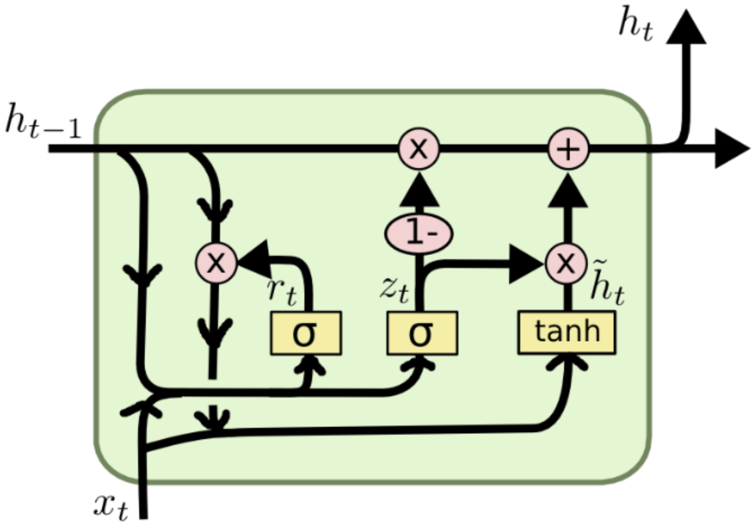
1. Weights matrix for the input
   - Forget gate: $n \times m$
   - Input gate: $n \times m$
   - Cell gate: $n \times m$
   - Output gate: $n \times m$

2. Weight matrix for the hidden state
    - Hidden state for forget gate: $m \times m$
    - Hidden state for input gate: $m \times m$
    - Hidden state for cell gate: $m \times m$
    - Hidden state for output gate: $m \times m$
3. Bias term
    - Bias for forget gate: $1 \times m$
    - Bias for input gate: $1 \times m$
    - Bias for cell gate: $1 \times m$
    - Bias for output gate: $1 \times m$

Total parameters: $4 \times (n + m + 1) \times m$

2.2.5 Gated Recurrent Unit (GRU)

The gated recurrent unit (GRU) was proposed by (Cho et al., 2014) to make each recurrent unit to adaptively capture dependencies of different time scales. The GRU has 2 gates, update gate and reset gate. Update gate:



The update gate determines how much of the past information should be retained in the current hidden state.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{34}$$

Reset gate:

The reset gate is similar with the update gate, but the candidate hidden state is influenced by the reset gate.

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{35}$$

Candidate hidden state:

The candidate hidden state combined with previous hidden state and current input to form the potential new information that can be added to the current hidden state.

$$\widetilde{h_t} = tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h) \tag{36}$$

Final hidden state

The final hidden state of the GRU at time t is a linear interpolation between the previous final hidden state and the candidate hidden state.

$$h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h_t} \tag{37}$$

Notation	Meaning	Dimension
x_t	Input vector at time t	$n \times 1$
h_t	Hidden state output at time t	$m \times 1$
r_t	Reset gate output at time t	$m \times 1$
z_t	Update gate output at time t	$m \times 1$

W_z	Weight matrix for the update gate	$m \times (m+n)$
W_r	Weight matrix for the candidate memory cell	$m \times (m+n)$
W_h	Weight matrix for the output gate	$m \times (m+n)$
b_z	Bias vector for the input gate	$m \times 1$
b_r	Bias vector for the candidate memory cell	$m \times 1$
b_h	Bias vector for the output gate	$m \times 1$

Number of parameters:

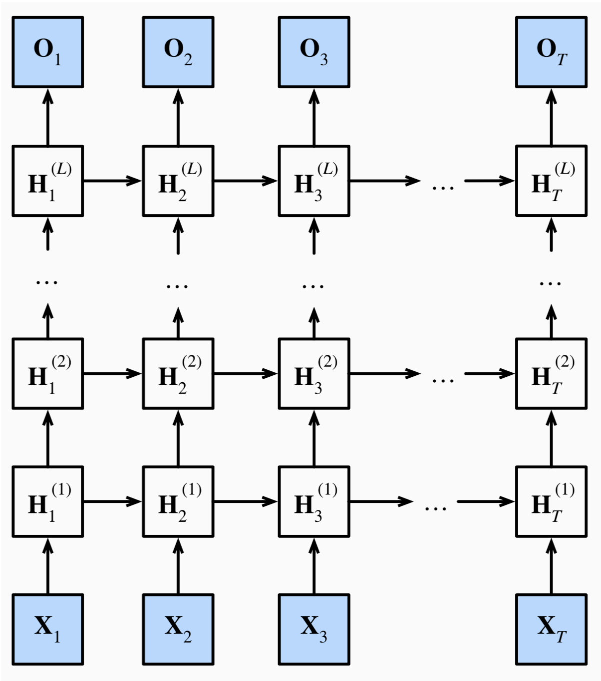
1. Weights matrix for the input
   - Update gate: $n \times m$
   - Reset gate: $n \times m$
   - Candidate hidden state: $n \times m$
2. Weight matrix for the hidden state
   - Hidden state for update gate: $m \times m$
   - Hidden state for reset gate: $m \times m$
   - Hidden state candidate hidden state: $m \times m$
3. Bias term
   - Bias for update: $1 \times m$
   - Bias for reset: $1 \times m$
   - Bias for candidate hidden state: $1 \times m$

Total parameters: $3 \times (n + m + 1) \times m$

2.2.6 Deep recurrent neural networks (DRNNs)

Deep architecture networks with multiple layers that can hierarchically learn complex representations (Bengio, 2009). By extending of this concept, stacking recurrent layers to form Deep Recurrent Neural Networks aligns with this principle. A number of research papers have been prove that the performance of DRNNs is out-performance then conventional RNNs. (Delalleau & Bengio, 2011; Le Roux & Bengio, 2010; Pascanu et al., 2013)

The architecture of DRNNs are similar with conventional RNNs. We simply stack the recurrent layers vertically. For the first layer, this layer receives the input and combines it with its previous hidden state h_{t-1} (Equation 38). The second layer receive the hidden state of the first layer and treat as the input of the layer 2 (Equation 39). We can extend this concept to L layers (Equation 40).



$$h_t^{(1)} = f(W_{xh}^{(1)} x_t + W_{hh}^{(1)} h_{t-1}^{(1)} + b_h^{(1)}) \tag{38}$$

$$h_t^{(2)} = f(W_{xh}^{(2)} h_t^{(1)} + W_{hh}^{(2)} h_{t-1}^{(2)} + b_h^{(2)}) \tag{39}$$

$$h_t^{(L)} = f(W_{xh}^{(L)} h_t^{(L-1)} + W_{hh}^{(L)} h_{t-1}^{(L)} + b_h^{(L)}) \tag{40}$$

$$o_t = W_{hy} h_t^{(L)} + b_y \tag{41}$$

$$y_t = g(o_t) \tag{42}$$

Where x_t is the input at time t. $h_t^{(l)}$ is the hidden state for the l layer at time t. $W_{xh}^{(l)}, W_{hh}^{(l)}$ are the weight matrices for the input to hidden and hidden to hidden connections in layer l, respectively. W_{hy} is weight matrix for the output layer. $b_h^{(l)}$ is the bias vector for the l layer (except output layer), b_y is the bias vector for output layer. $g(\cdot)$ and $f(\cdot)$ are an activation function.

2.2.7 Hidden Markov Model

Before reviewing Hidden Markov Model (HMM), it is essential to understand what Markov models is, or Markov chains. Markov chains are fundamental models in probability theory and statistic, and it is a stochastic process describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. This property known as the Markov property (Roberts & Rosenthal, 2004). In rigorous terms, let the state space be defined as: Let the state space be defined as:

$$S = \{s_1, s_2, \ldots, s_N\}. \tag{43}$$

$$P(s_{t+1} \mid s_t, s_{t-1}, s_{t-2}, \cdots, s_1) = P(s_{t+1} \mid s_t) \tag{44}$$

The state transition probability is denoted by:

$$P_{ij} = P(s_{t+1} = s_j \mid s_t = s_i). \tag{45}$$

Given an initial state s_1 with probability $P(s_1)$, the joint probability of a state sequence $\{s_1, s_2, \ldots, s_T\}$ can be written as:

$$P(s_1, s_2, \ldots, s_T) = P(s_1) \cdot P(s_2 \mid s_1) \cdot P(s_3 \mid s_2) \cdots P(s_T \mid s_{T-1}) \tag{46}$$

We can simplfy the joint probability of a state sequence as equestion 47.

$$P(s_1, s_2, \ldots, s_T) = P(s_1) \prod_{t=1}^{T-1} P(s_{t+1} \mid s_t) \tag{47}$$

Then it is essential to discuss marginal probability in a Markov model because it provides critical information about the likelihood of being in particular state s_j at a specific time t. The marginal probability of being in state s_j at time t defined as:

$$\pi_t(s_j) = P(s_t = s_j) \tag{48}$$

It is the probability of the system being in state s_j at time t, irrespective of how the system transitioned to s_j. We can obtainde by summing over all possible transition probability P_{ij} where s_i belong in state space.

$$\pi_{t+1}(s_j) = \sum_{s_i \in S} \pi_t(s_i) \cdot P_{ij} \tag{49}$$

If Markov chain is irreducible and aperiodic, the stationary distribution defined as below:

$$\pi_j = \sum_{i=1}^{n} \pi_i P_{ij} \tag{50}$$

Hidden markov model

Hidden Markov Model (HMMs) is a statistical model that extends the Markov chains by introducting a layer of latent (unobservable) variables. And connecting them to observable outputs (emission probability). In contrast to standard Markov chains, hidden states are never be observed directly. Instead, we observe a sequence of data (observations) that are probabilitically generated by these hidden state. This structure has proven useful for applications in speech recognition, biological sequence analysis, and signal processing (Rabiner, 1989). Lets defined the

Literature Review on Hidden Markov Models
Hidden Markov Models (HMMs) extend classical Markov chains by incorporating an additional layer of latent (unobservable) variables that govern the generation of the observed data. In contrast to standard Markov chains—where the state of the system is directly observed—an HMM assumes that the underlying process is hidden and that each observed datum is stochastically generated from the corresponding hidden state. This two-layer structure has proven to be both theoretically interesting and practically useful for applications in speech recognition, biological sequence analysis, and signal processing (Rabiner, 1989).
1. From Markov Chains to Hidden Markov Models
Let $\{X_n\}_{n\geq 1}$ be a Markov chain on a measurable space $(\mathcal{X}, \mathcal{F})$ satisfying

$$P\big(X_{n+1} \in A \mid X_n, X_{n-1}, \dots, X_1\big) = P\big(X_{n+1} \in A \mid X_n\big) \quad \text{for all } A \in \mathcal{F}.$$

This *Markov property* ensures that the future evolution of the process depends only on its present state, not on the full past history.
In extending this framework to HMMs, we introduce an observable process $\{Y_n\}_{n\geq 1}$ defined on another measurable space $(\mathcal{Y}, \mathcal{G})$. The connection between the hidden states and the observations is governed by the following assumption:
- **Conditional Independence of Observations:** Given the current hidden state,

$$P\big(Y_n \in B \mid X_n, X_{n-1}, \dots, X_1, Y_{n-1}, \dots, Y_1\big) = P\big(Y_n \in B \mid X_n\big)$$

for every $B \in \mathcal{G}$. This states that the observation Y_n is independent of past states and observations, conditioned on X_n.
Thus, an HMM is fully specified by the following three components: 1. The **initial distribution** π on $\mathcal{X}$:

$$\pi(x) = P(X_1 = x).$$

2. The **transition kernel** $P(x, dy)$ of the hidden Markov chain:

$$p_{ij} = P(X_{n+1} = j \mid X_n = i).$$

3. The **emission (observation) distribution** $B(x, dy)$ or $b_x(y)$:

$$b_x(y) = P(Y_n = y \mid X_n = x).$$

2. Joint and Marginal Distributions
Using the above structure, the joint probability of a sequence of hidden states and observations $\{(X_1, Y_1), \dots, (X_N, Y_N)\}$ is given by

$$\begin{aligned} P\big(X_1, \dots, X_N,\, Y_1, \dots, Y_N\big) &= \pi(X_1)\, b_{X_1}(Y_1) \prod_{n=2}^{N} P\big(X_n \mid X_{n-1}\big)\, b_{X_n}(Y_n) \\ &= \pi(X_1)\, b_{X_1}(Y_1)\, p_{X_1 X_2}\, b_{X_2}(Y_2) \cdots p_{X_{N-1} X_N}\, b_{X_N}(Y_N). \end{aligned}$$

Marginalization over the hidden states yields the likelihood of the observations:

$$P(Y_1, \dots, Y_N) = \sum_{X_1 \in \mathcal{X}} \cdots \sum_{X_N \in \mathcal{X}} \pi(X_1) \prod_{n=2}^{N} p_{X_{n-1} X_n} \prod_{n=1}^{N} b_{X_n}(Y_n),$$

which in many applications is used for parameter estimation and inference.
3. Inference and Algorithmic Developments
Due to the latent structure in HMMs, direct computation of the likelihood or the posterior distribution of the hidden states is usually intractable when the state space is large or continuous. This has spurred the development of several efficient algorithms:
- **The Forward-Backward Algorithm:** Utilizes dynamic programming to recursively compute the probabilities of partial observation sequences. For instance, denoting the forward variable by

$$\alpha_n(x) = P(Y_1, Y_2, \ldots, Y_n, X_n = x),$$

one can compute

$$\alpha_{n+1}(y) = \left[\sum_{x\in\mathcal{X}} \alpha_n(x)\, p_{xy}\right] b_y(Y_{n+1}).$$

- **The Viterbi Algorithm:** Determines the most probable sequence of hidden states (i.e., finds $\hat{X}_1, \ldots, \hat{X}_N$ such that the joint probability is maximized).
- **The Baum-Welch Algorithm (Expectation-Maximization):** Iteratively estimates the unknown parameters (transition probabilities and emission probabilities) of the HMM when they are not known a priori.
These methods rely on the underlying assumption that the hidden state process is a Markov chain, and many of the rigorous convergence and mixing properties discussed in the literature on general state space Markov chains (e.g., Roberts and Rosenthal, 2004; Meyn and Tweedie, 1993) carry over—often with additional considerations due to the unobserved structure.
4. Theoretical Underpinnings and Convergence
As with observable Markov chains, the study of HMMs involves rigorous mathematical analysis centered on convergence properties and ergodicity. Under appropriate conditions (e.g., the hidden chain is φ-irreducible, aperiodic, and satisfies a suitable drift condition), one can prove that the filtering distributions

$$P(X_n \in \cdot \mid Y_1, \ldots, Y_n)$$

converge to a unique invariant measure. Techniques such as **coupling** and **regeneration**—which have been successfully employed in establishing quantitative convergence rates for Markov chains (Roberts & Rosenthal, 2004)—are also adapted to the HMM context to study the stability and consistency of inference procedures.
For example, under the standard assumptions of an HMM, if the hidden state chain is geometrically ergodic then for a function $h : \mathcal{X} \to \mathbb{R}$ with finite second moment under the stationary measure π, the central limit theorem holds for the additive functional

$$\frac{1}{\sqrt{n}} \sum_{i=1}^{n} \left[h(X_i) - \pi(h)\right],$$

which is essential for quantifying the uncertainty in estimators derived via the Expectation-Maximization or sequential Monte Carlo methods (see also Billingsley, 1995).
5. Conclusion
In summary, by extending the framework of Markov chains—characterized by the Markov property, established convergence criteria, small set theory, and drift conditions—to a hidden state setting, Hidden Markov Models provide a robust foundation for dealing with complex sequence data where the underlying process is not directly observable. This

extension leads to rich algorithmic and inferential techniques that have been rigorously analyzed and widely adopted in several applied fields.