Recurrent Deep Learning Models and its applications

Ngai Ho Wang

February 2, 2025

Contents

1 Project goals

The goal of this paper is to analyze, implement, and compare the performance of RNN, LSTM, GRU and own model in selected NLP tasks. This paper aims to propose an enhanced RNN-based model for NLP tasks and make the recommendation for future development.

2 Objectives

1. Literature Review
    - Conduct a comprehensive review of existing paper on RNN, LSTM, and GRU, focusing on their architecture.
    - Review the NLP task.
2. Model implementation
    - Implement RNN, LSTM, GRU and own model by PyTorch for chosen NLP tasks.
3. Performance Comparison
    - Evaluate the performance by using appropriate metrics (e.g., accuracy, precision, recall, F1 score, BLEU, ROUGE).
4. Propose an advanced model
    - Develop an advanced model, aiming to enhance the performance on selected NLP tasks.

3 Expected outcomes

1. Model performance metrics
    - A detailed comparison of performance metrics across RNN, LSTM, GRU and own model.
2. Practical insights
    - Practical recommendations for future work in the area of RNN applications in NLP, based on the findings of this paper.

4 Research plan

1. Literature review
    - Review existing work on RNN, LSTM, GRU and their more sophisticated variants.
2. Model development

- Select NLP tasks. (e.g., text classification, summarization).
- Implement baseline models (RNN, LSTM, GRU) using PyTroch.

3. Performance Evaluation
    - Evaluate and compare the performance of the implemented model using appropriate metrics.
4. Model Enhancement
    - Design and implement an advanced model based on findings from the previous phases.
5. Final analysis and reporting
    - Analyze the results of the advanced model against baseline models.

5 Experimental design

1. Dataset selection
    - Select suitable NLP dataset.
2. Model configuration
    - Define architecture specifications for each model, including number of layers, number of hidden units, activation function, dropout rates.

6 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. Siri, Alexa and bixby have shown the possibility. Everyone can communicate with those machines and they with make the reasonable response to user.

Recurrent Neural Networks (RNNs) have been a foundational architecture in this domain, the architecture of RNNs is design for sequential data. It able to retain the information through hidden states. 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.

7 Literature Review

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.

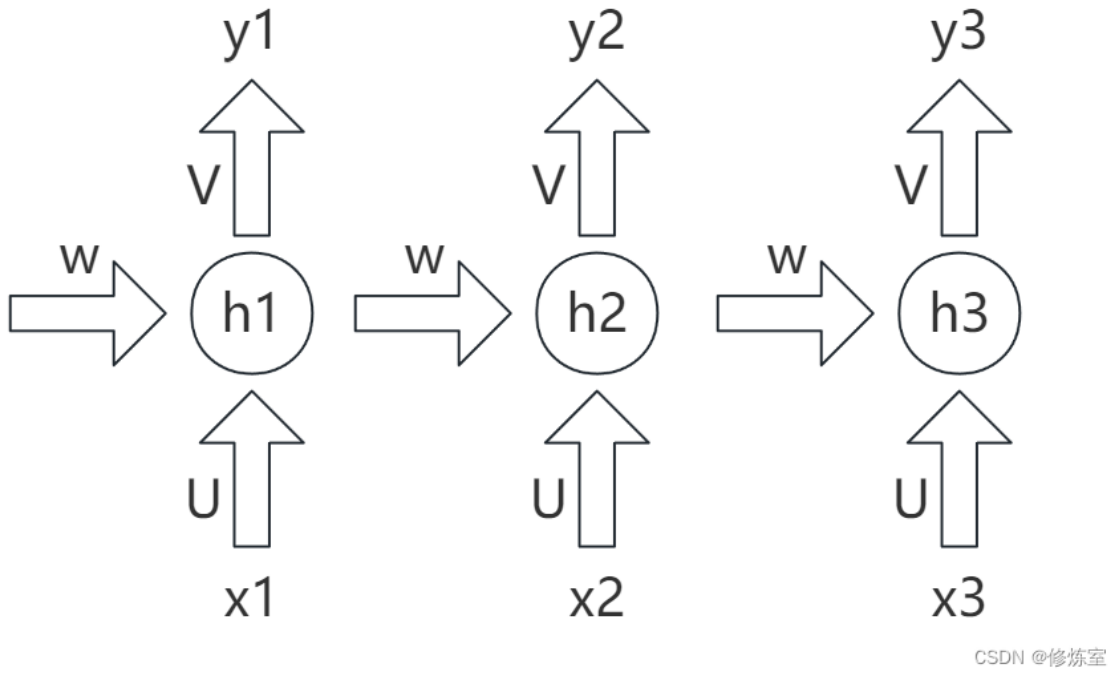


Figure 1: Unfolded RNN

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.

$$\frac{\partial \mathcal{L}_2}{\partial W} = \sum_{i=1}^{2} \frac{\partial L_i}{\partial W} \tag{4}$$

$$\frac{\partial L_i}{\partial W} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial h_i} \cdot \frac{\partial h_i}{\partial W} \tag{5}$$

$$\frac{\partial \mathcal{L}_2}{\partial W} = \frac{\partial \mathcal{L}_1}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial \hat{h}_1} \cdot \frac{\partial h_1}{\partial W} + \frac{\mathcal{L}_2}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial \hat{h}_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W} \tag{6}$$

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

$$\frac{\partial \mathcal{L}_2}{\partial U} = \sum_{i=1}^{2} \frac{\partial L_i}{\partial U} \tag{7}$$

$$\frac{\partial L_i}{\partial U} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial h_i} \cdot \frac{\partial h_i}{\partial U} \tag{8}$$

$$\frac{\partial \mathcal{L}_2}{\partial U} = \frac{\partial \mathcal{L}_1}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial \hat{h}_1} \cdot \frac{\partial h_1}{\partial U} + \frac{\mathcal{L}_2}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial \hat{h}_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial U} \tag{9}$$

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

$$\frac{\partial \mathcal{L}_2}{\partial V} = \sum_{i=1}^{2} \frac{\partial L_i}{\partial V} \tag{10}$$

$$\frac{\partial L_i}{\partial V} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial h_i} \cdot \frac{\partial h_i}{\partial V} \tag{11}$$

$$\frac{\partial \mathcal{L}_2}{\partial V} = \frac{\partial \mathcal{L}_1}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial \hat{h}_1} \cdot \frac{\partial h_1}{\partial V} + \frac{\mathcal{L}_2}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial \hat{h}_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial V} \tag{12}$$

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

$$\frac{\partial \mathcal{L}_2}{\partial b_h} = \sum_{i=1}^{2} \frac{\partial L_i}{\partial b_h} \tag{13}$$

$$\frac{\partial L_i}{\partial b_h} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial h_i} \cdot \frac{\partial h_i}{\partial b_h} \tag{14}$$

$$\frac{\partial \mathcal{L}_2}{\partial b_h} = \frac{\partial \mathcal{L}_1}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial \hat{h}_1} \cdot \frac{\partial h_1}{\partial b_h} + \frac{\mathcal{L}_2}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial \hat{h}_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial b_h} \tag{15}$$

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

$$\frac{\partial \mathcal{L}_2}{\partial b_y} = \sum_{i=1}^{2} \frac{\partial L_i}{\partial b_y} \tag{16}$$

$$\frac{\partial L_i}{\partial b_y} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial h_i} \cdot \frac{\partial h_i}{\partial b_y} \tag{17}$$

$$\frac{\partial \mathcal{L}_2}{\partial b_y} = \frac{\partial \mathcal{L}_1}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial \hat{h}_1} \cdot \frac{\partial h_1}{\partial b_y} + \frac{\mathcal{L}_2}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial \hat{h}_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial b_y} \tag{18}$$

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

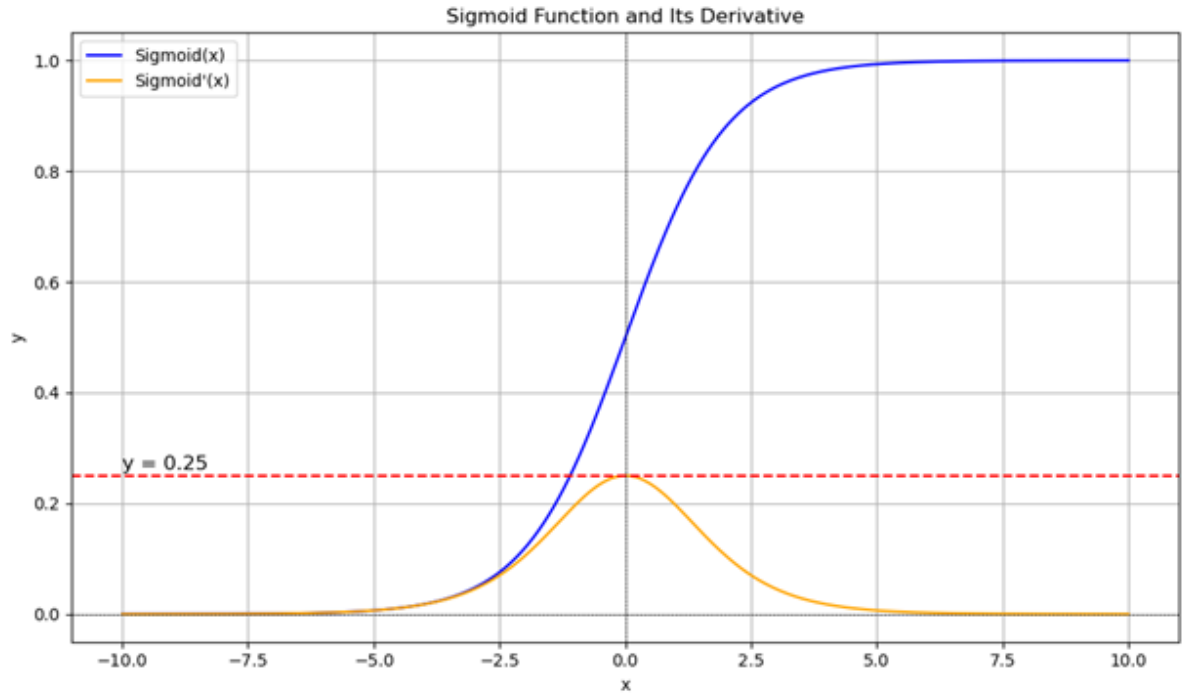
Activation function

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.

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

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

Below is the sigmoid function and its derivative.



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

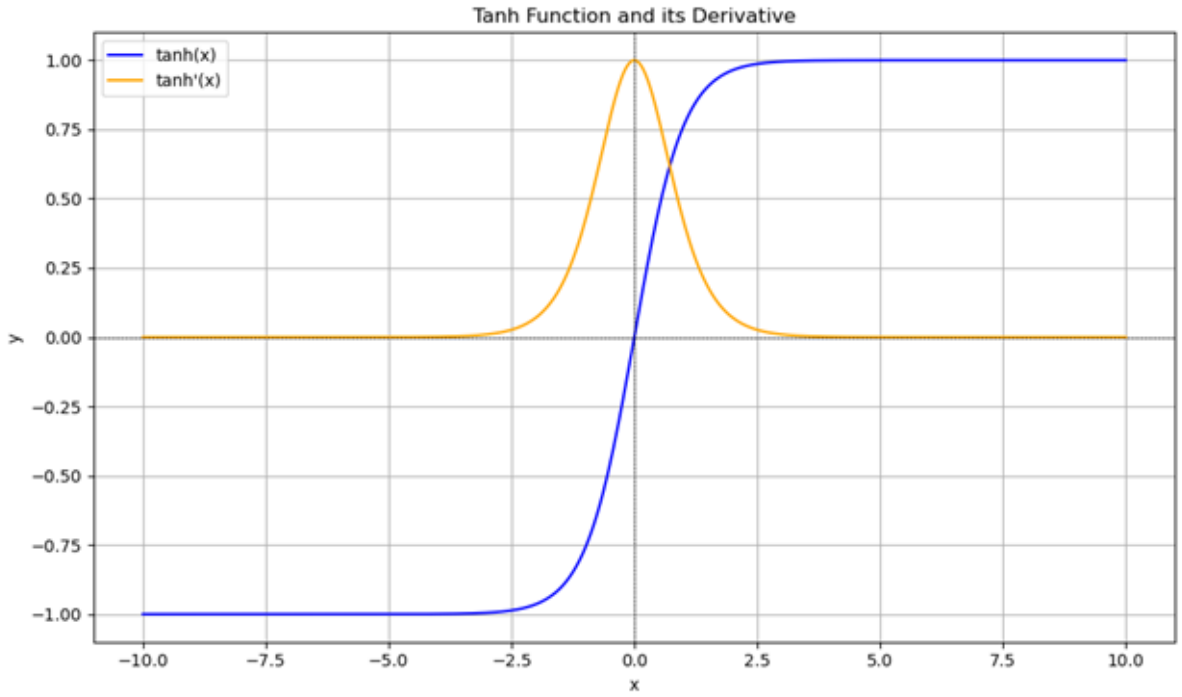
Hyperbolic tangent activation function

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

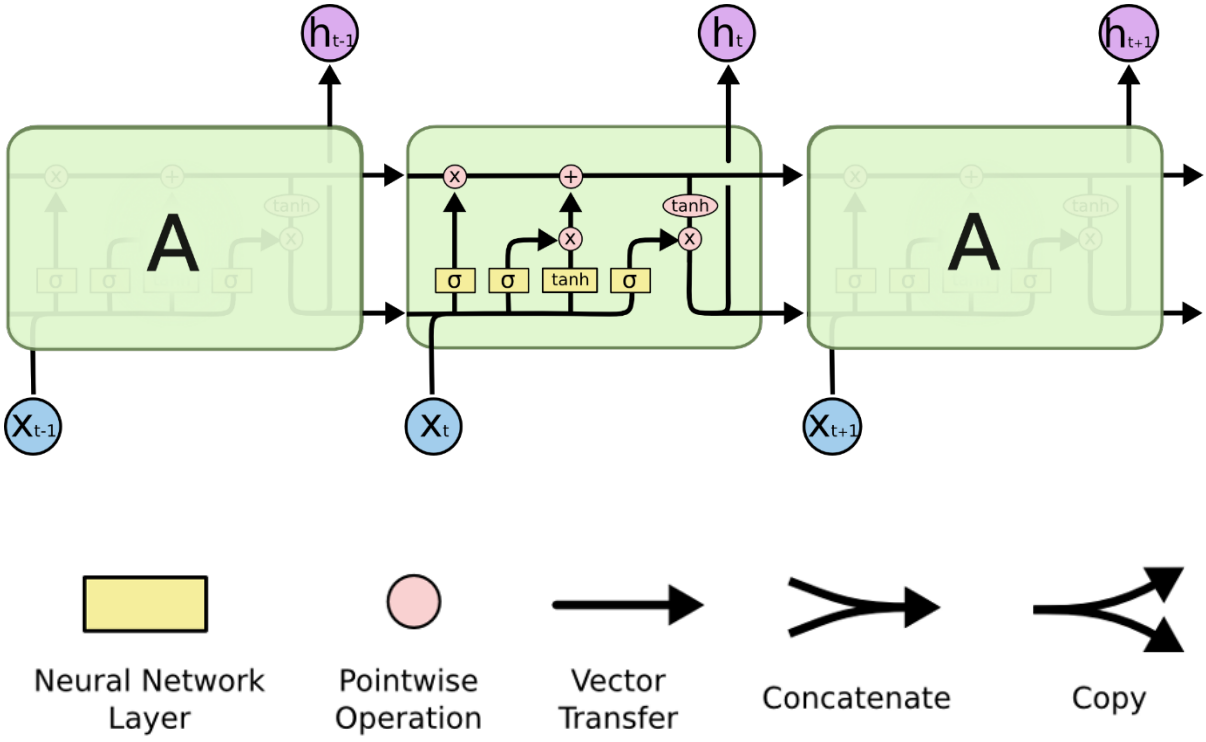
When we are doing the feedforward and get super large value computed by loss function. Then when updating the parameters. The updates to the weights will also be large. Resulting in higher loss and larger gradients in the next iterations. This will lead to exploding gradients.

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

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 gate, 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.

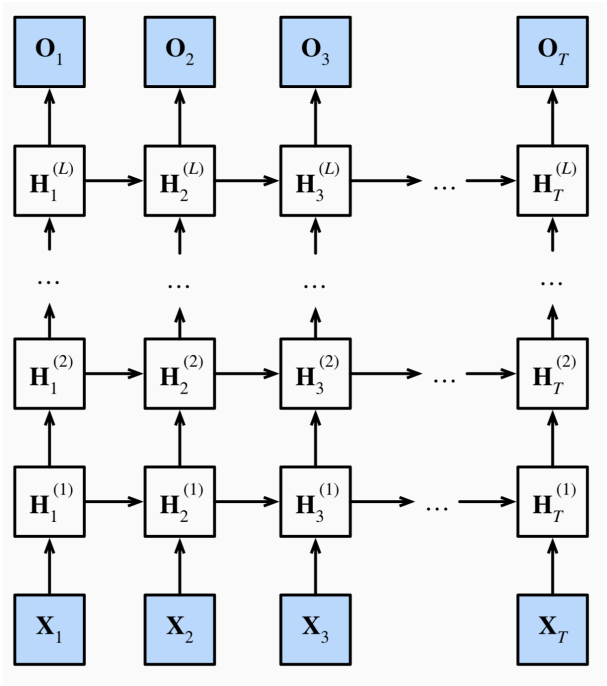
$$h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h_t} \tag{37}$$

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	$m \times 1$

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, b_y is the bias vector for output layer. $g(\cdot)$ and $f(\cdot)$ are an activation function.

For the recurrent layers,

Hidden Markov Model

For the NLP tasks, there may have some unobservable data, for example the implicit sentiment, user intent, emotional state. (Rabiner & Juang, 1993) proposes the hidden Markov model (HMM), the assumption is the existence of the latent process follows a Markov chain from which observations X are generated. In other word, there would exists an unobserved state sequence $Z = \{z_1, z_2 \cdots, z_T\}$ in observed sequence $X = \{x_1, x_2, \cdots, x_T\}$. Where the hidden states, z_t belonging to state-space $Q = \{q_1, q_2, \cdots, q_M\}$ follow a Markov chain goverened by:

- A state-transition probability matrix $A = [a_{ij}] \in \mathbb{R}^{M \times M}$ where $a_{ij} = p(z_{t+1} = q_j | z_t = q_i)$
- Initial state matrix $\pi = [\pi_i] \in \mathbb{R}_{1 \times M}$ with $\pi_i = p(z_1 = q_i)$

Furthermore, for the hidden state z_t, correspoinding to the observe data x_t is release by emission process $B = [b_j(x)]$ where $b_j(x) = p(x|z = q_i)$. We can assume $b_j(x)$ is follows the Gaussian mixture model (GMM).

$$p(x_t|z = q_j) = \sum_{l=1}^{k} c_{jl} \mathcal{N}(x_t|\mu_{jl}, \Sigma_{jl}) \tag{43}$$

where $\sum_{l=1}^{k} c_{jl} = 1$, $\forall j = \{1, \cdots, M\}$, k is the number of Gaussian mixture components and $\mathcal{N}(x_t|\mu_{jl}, \Sigma_{jl})$ denotes a Gaussian probability density with mean μ_{jl} and covariance Σ_{jl} for state j and mixture component l. The number of hidden states (M) and mixture component (k) are the two hyperparameters of the model which have to be provided apriori.

Therefore, the joint probability probability density function of the observation X can be expressed as:

$$p(X) = p(z_1) \prod_{T-1}^{t=1} p(z_{t+1}|z_t) \prod_{t=1}^{T} p(x_t|z_t) \tag{44}$$

The optimal parameters $[A, B, \pi]$, which maximize the likelihood of the observation sequence X (Equation 39), are determined using the Baum-Welch algorithm, an expectation-maximization method (Rabiner & Juang, 1993). Additionally, the probability of the system in a specific hidden state z_t corresponding to the observation x_t is calculated using the Viterbi algorithm.

Word representation

In RNN, the due to the architecture, the model can only handle vector, it is important that convert the word into machine readable format. By translating words into vectors, these representations capture semantic meanings, relationships and contexts. (Mikolov et al., 2013) proposed 2 model architectures for leaning distributed representations of words.

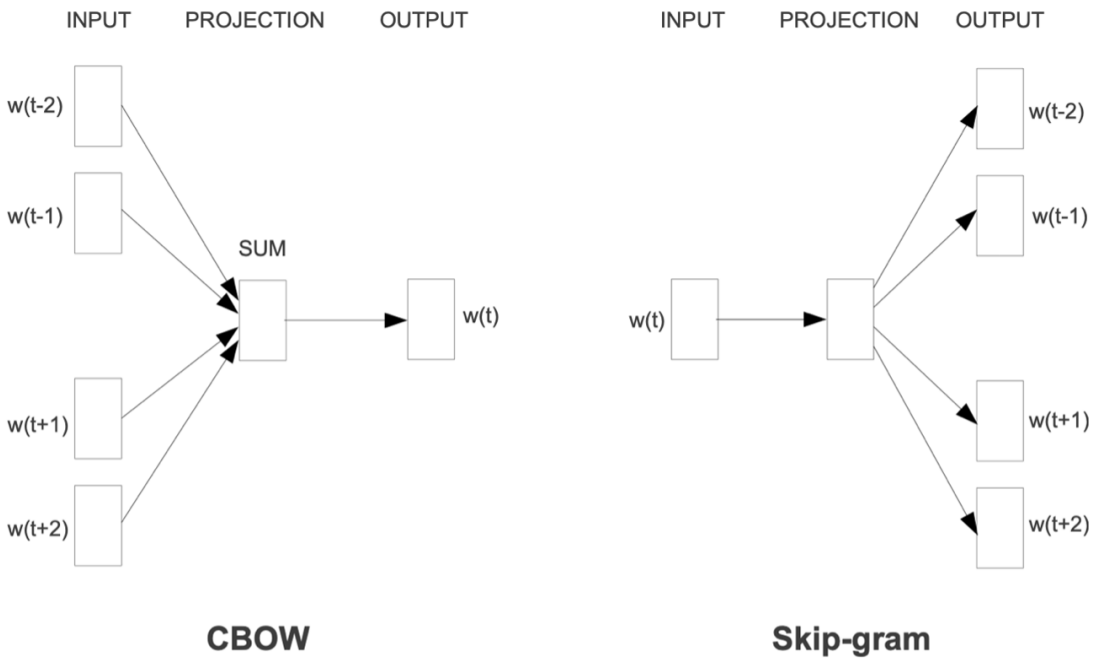
Continuous Bag of Words model CBOW operates by taking context words around a target word, it is flexible to adjust the window size which mean that we can control how many words around our target word. Hence aggregating their embeddings and passing to hidden layer to produce a probability distribution. Capture the semantic meanings and the relationship more effectively.

$$Q = N \times D + D \times log_2(V) \tag{45}$$

Continuous Skip gram model

The continuous skip gram model operates in opposite direction compare with CBOW. The model takes the surrounding word to predict the target word. In other word, treat the current word as an input to a log-linear classifier with continuous projection layer, and predict the probability distribution of surrounding words.

$$Q = C \times (D + D \times log_2(V)) \tag{46}$$



In this paper,

References

Bengio, Y. (2009). Learning deep architectures for ai. *Found. Trends Mach. Learn.*, *2*(1), 1–127. https://doi.org/10.1561/2200000006

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. https://doi.org/10.1109/72.279181

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint*. https://doi.org/10.48550/arxiv.1409.1259

Delalleau, O., & Bengio, Y. (2011). Shallow vs. deep sum-product networks. *In NIPS*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Le Roux, N., & Bengio, Y. (2010). Deep belief networks are compact universal approximators. *FouNeural Computation*, *22*(8), 2192–2207. https://doi.org/10.1162/neco.2010.08-09-1081

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint*. https://doi.org/10.48550/arxiv.1301.3781

Pascanu, R., Montufar, G., & Bengio, Y. (2013). On the number of response regions of deep feed forward networks with piece-wise linear activations. *In NIPS*. https://doi.org/10.48550/arxiv.1312.6098

Rabiner, L. R., & Juang, B. H. (1993). *Fundamentals of speech recognition*. PTR Prentice Hall.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature (London)*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560. https://doi.org/10.1109/5.58337

Wikipedia. (2023). Backpropagation through time [In Wikipedia, The Free Encyclopedia. Retrieved November 12, 2024, from https://en.wikipedia.org/wiki/Backpropagation_through_time].

8 Inserting Images

asdfpiojaseopritjf To insert an image, use the ‘graphicx‘ package. For example:

9 Tables

You can create tables using the ‘tabular‘ environment or ‘booktabs‘ for professional-quality tables. For example:

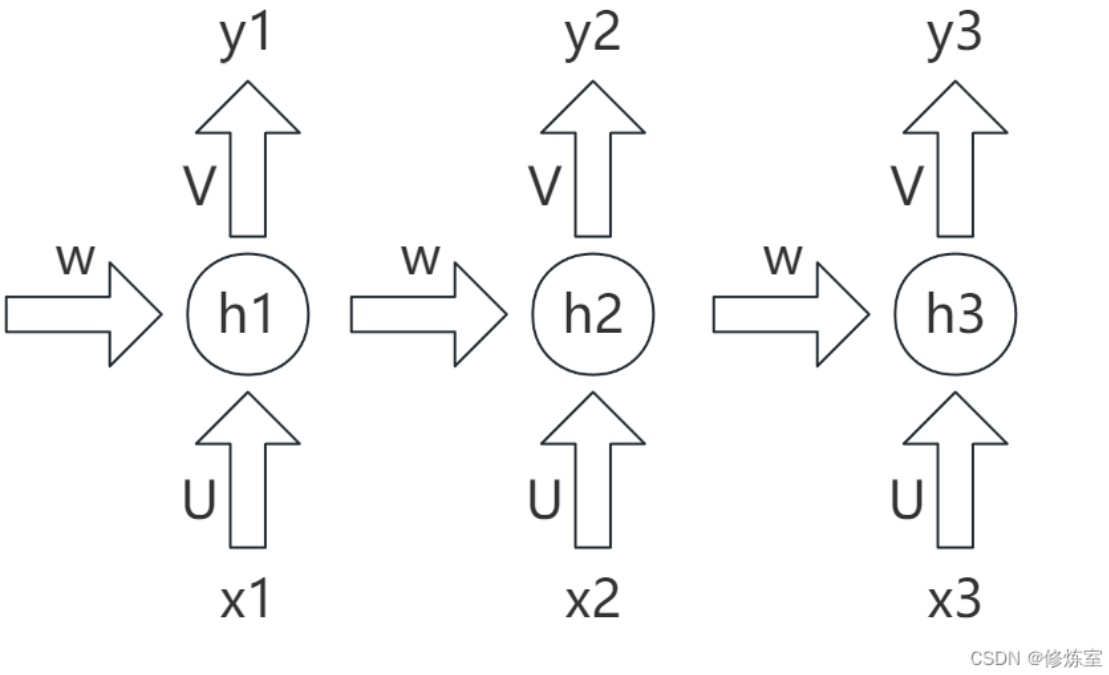


Figure 2: An example image.

Table 4: Example Table

Item	Description	Quantity
Apples	Fresh red apples	10
Oranges	Juicy oranges	5
Bananas	Ripe bananas	7

10 Hyperlinks

To add a hyperlink, use the ‘hyperref‘ package. For example: Visit the LaTeX project website.

11 Conclusions

This is the conclusion section. Summarize your findings or leave final remarks.

A Appendix

This is the appendix section, where you can include supplementary materials.