Recurrent Deep Learning Models and its applications

Ngai Ho Wang

January 21, 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, Simard, & Frasconi, 1994). Eliminating many practical applications of RNNs. After that, Hochreiter and 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, Hinton, & Williams, 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.

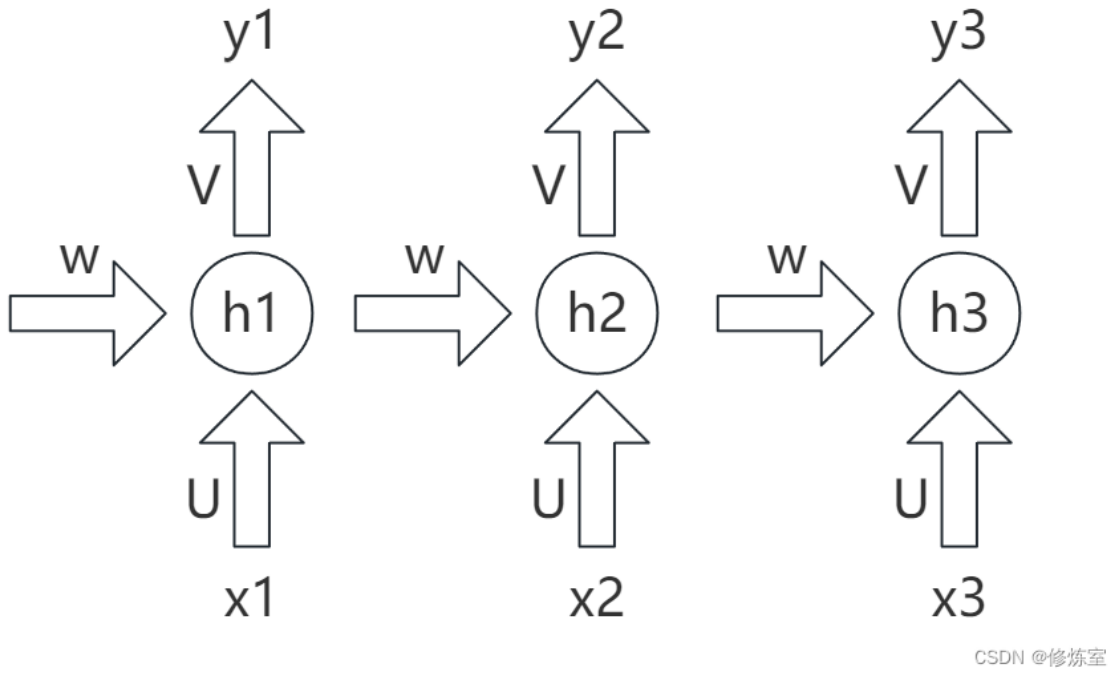


Figure 1: Unfolded RNN

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

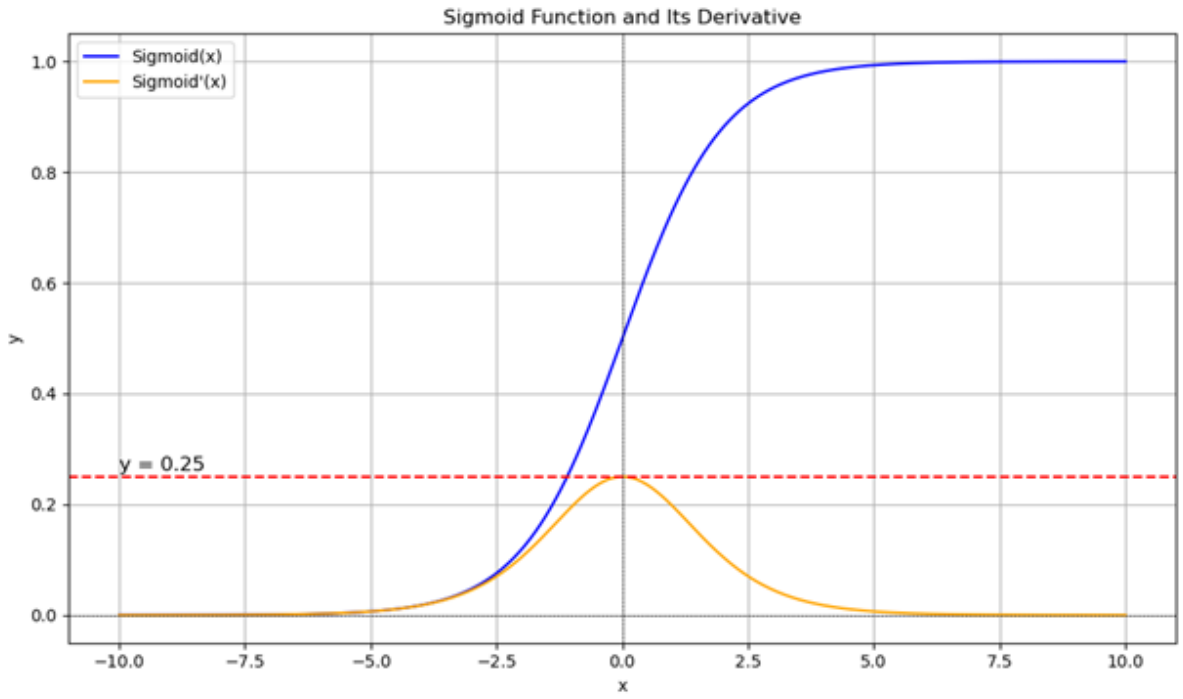
Activation function

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.



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

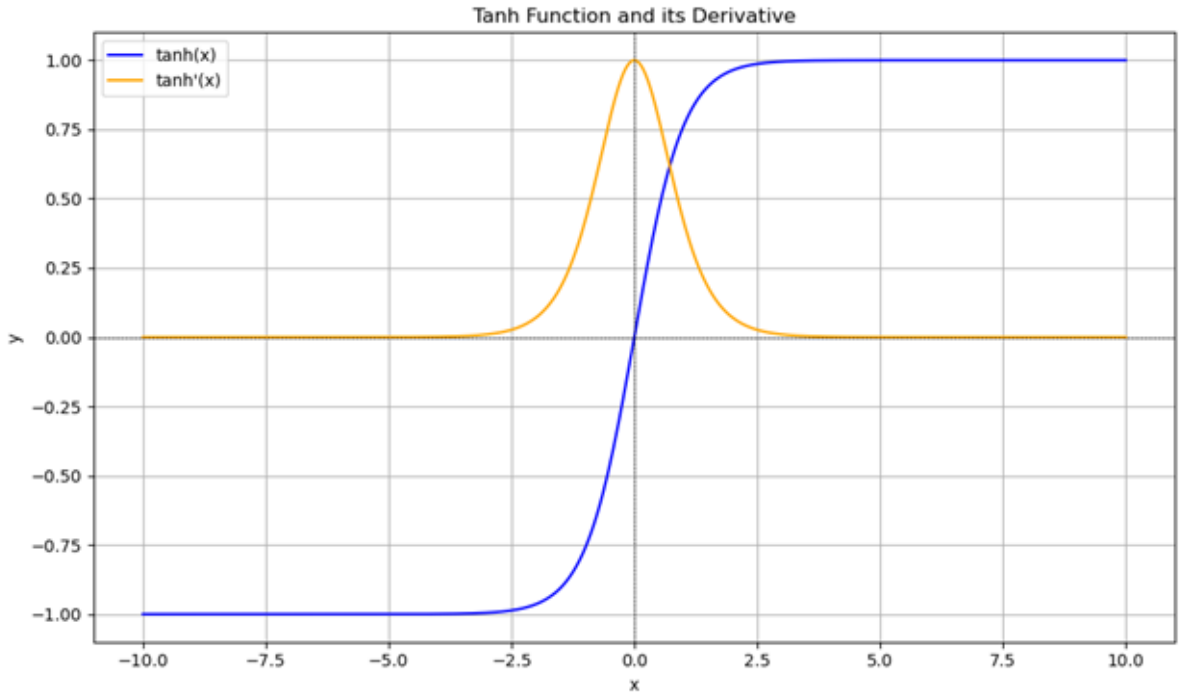
Hyperbolic tangent activation function

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).

8 Inserting Images

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

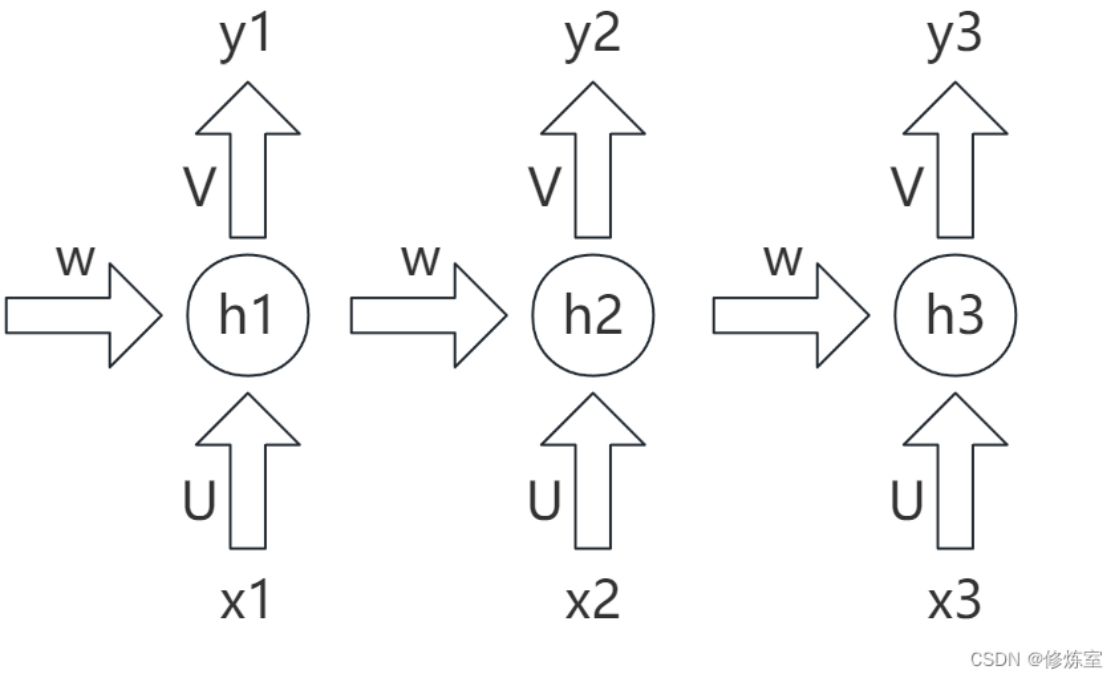


Figure 2: An example image.

9 Tables

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

Table 2: 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.