ARTIFICIAL NEURAL NETWORKS - WEEK 7

Recurrent Neural Networks (RNNs)

Dr. Aamir Akbar

Director of both AWKUM AI Lab and AWKUM Robotics, Final Year Projects (FYPs) coordinator, and lecturer at the department of Computer Science
Abdul Wali Khan University, Mardan (AWKUM)

CONTENTS

RECURRENT NEURAL NETWORKS (RNNS)

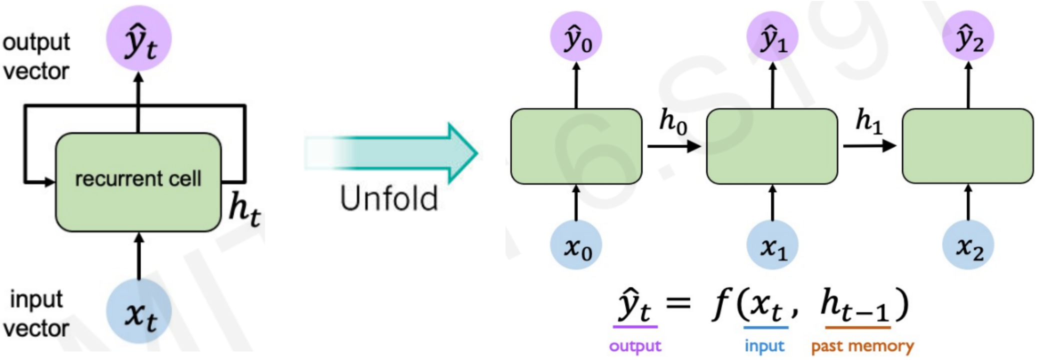
What are RNNss?

RNNs are a class of artificial neural networks designed to capture `sequential information` and dependencies within data by allowing connections between nodes to form directed cycles. Unlike traditional feedforward neural networks, RNNs have connections that enable them to exhibit `temporal dynamic behavior`, making them suitable for processing sequential data such as `time series, text, and speech`.

Example Applications of RNNs:

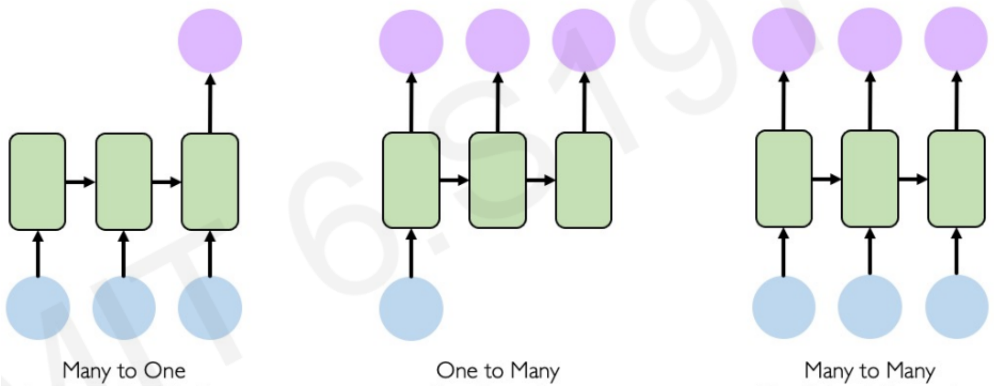
1. `NLP:` language translation, sentiment analysis, named entity recognition, text generation.
2. `Speech Recognition:` transcribe spoken language into text
3. `Time Series Prediction:` stock price prediction, weather forecasting, energy demand prediction.
4. `Gesture Recognition:` sign language recognition, human-computer interaction.
5. `Video Analysis:` action recognition, video captioning, and activity recognition.
6. `Music Generation:` music composition, melody generation.
7. `Others:` healthcare (e.g., medical signal analysis, patient monitoring), finance (e.g., fraud detection, algorithmic trading), and more.

RECURRENCE



In each forward pass, the hidden state h_t at time step t depends not only on the current input x_t but also on the previous hidden state h_{t-1}, which captures information from earlier time steps. This `recurrence` mechanism enables RNNs to process sequential data by maintaining a memory of past information and incorporating it into the current computation.

RNNS FOR SEQUENCE MODELING



RNNS FOR SEQUENCE MODELING

Many-to-One RNN:

- **Example:** Sentiment analysis of movie reviews, where a sequence of words (input) is analyzed to predict the sentiment (output) of the entire review.
- **Application:** Analyzing sentiment in text, speech, or other sequential data.

One-to-Many RNN:

- **Example:** Music generation, where a single input (e.g., a starting note or chord) is used to generate a sequence of outputs (e.g., a melody or a musical composition).
- **Application:** Generating music or text descriptions from a given prompt or initial seed.

Many-to-Many RNN:

- **Example:** Machine translation, where a sequence of words in one language is translated into another sequence of words in a different language.
- **Application:** Language translation, video captioning, speech recognition (phoneme-to-word mapping).

RNNS FOR SEQUENCE MODELING

Many-to-Many (Same Length) RNN:

Example: Video classification, where a sequence of video frames is classified into different categories, with input and output sequences of the same length.

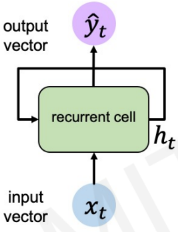
Application: Action recognition in videos, gesture recognition.

Many-to-Many (Different Lengths) RNN:

Example: Speech recognition, where variable-length audio sequences are transcribed into variable-length text sequences.

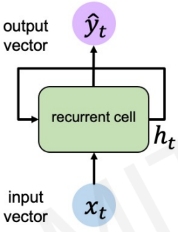
Application: Transcribing spoken language into text, speech-to-text systems.

STEPS TO HIDDEN STATE UPDATE AND OUTPUT

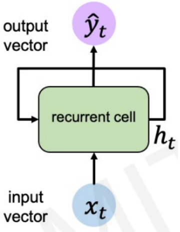


STEPS TO HIDDEN STATE UPDATE AND OUTPUT

1. **Input Vector:** x_t



STEPS TO HIDDEN STATE UPDATE AND OUTPUT



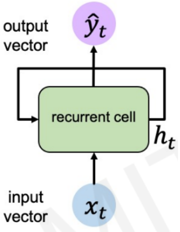
1. **Input Vector:** x_t

2. **Update the Hidden State:** h_t

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

where h_t is the hidden state at time t, x_t is the input at time t, W_{xh} (input to hidden) and W_{hh} (hidden to hidden) are the weight matrices, h_{t-1} is the hidden state at the previous time step, b_h is the bias term, and σ is the activation function.

STEPS TO HIDDEN STATE UPDATE AND OUTPUT



1. Input Vector: x_t

2. Update the Hidden State: h_t

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

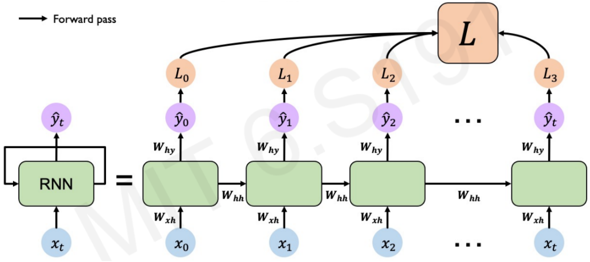
where h_t is the hidden state at time t, x_t is the input at time t, W_{xh} (input to hidden) and W_{hh} (hidden to hidden) are the weight matrices, h_{t-1} is the hidden state at the previous time step, b_h is the bias term, and σ is the activation function.

3. Output Vector: $\hat{y}_t$

$$\hat{y}_t = \sigma_{out}(W_{hy}h_t + b_y) \tag{2}$$

where $\hat{y}_t$is the output vector at time step t, h_t is the hidden state vector at time step t, W_{hy} is the weight matrix connecting the hidden state to the output, b_y is the bias vector for the output layer, and σ_{out} is the activation function.

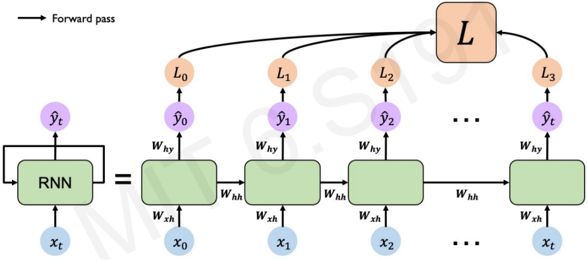
RNNS - BACKPROPAGATION THROUGH TIME (BPTT)



1. Loss Calculation at Each Time Step: At each time step t, the model generates an output $\hat{y}$ based on the input x_t and the current hidden state h_t. The loss $\mathcal{L}_t$ at time step t is calculated using a suitable loss function, such as cross-entropy loss for classification tasks or mean squared error for regression tasks. For example:

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

RNNS - BACKPROPAGATION THROUGH TIME (BPTT)

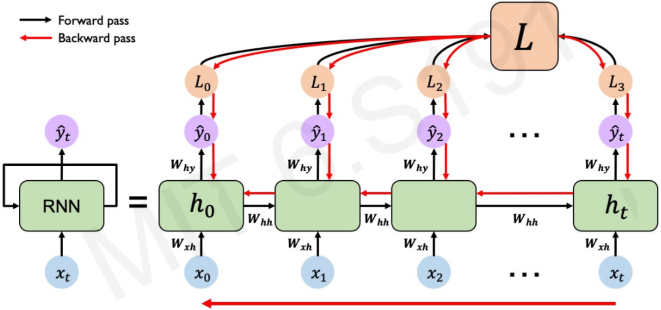


2. Aggregation/Average of Losses: The loss $\mathcal{L}_t$ at each time step are then aggregated or averaged to compute the final loss $\mathcal{L}$. The final loss can be expressed as:

$$\text{Aggregation:} \quad \mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t$$

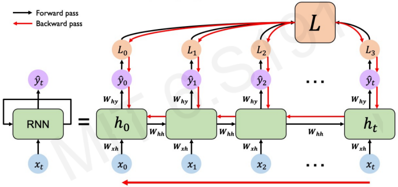
$$\text{Average:} \quad \mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}_t$$

RNNS - BACKPROPAGATION THROUGH TIME (BPTT)



3. Backpropagation and Optimization: After computing the final loss $\mathcal{L}$, the gradients of the loss with respect to the model parameters (weights and biases) are calculated using backpropagation through time (BPTT). These gradients are then used to update the model parameters using an optimization algorithm such as stochastic gradient descent (SGD) or its variants.

RNNS - BACKPROPAGATION THROUGH TIME (BPTT)



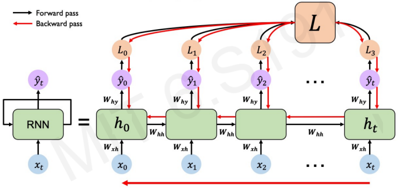
1. For each time step t, compute the gradient of the total loss $\mathcal{L}$ with respect to the output $\hat{y}_t$:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_t} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t}$$

2. Backpropagate the gradients through time to compute the gradients of the weights W_{hy}, W_{hh} and W_{xh}:

$$\frac{\partial \mathcal{L}}{\partial W_{hy}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial W_{hy}}$$

RNNS - BACKPROPAGATION THROUGH TIME (BPTT)

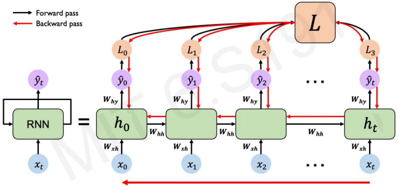


$$\frac{\partial \mathcal{L}}{\partial W_{xh}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_{xh}}$$

$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_{hh}}$$

3. Update the weights W_{hy}, W_{hh} and W_{xh} using the computed gradients and an optimization algorithm such as stochastic gradient descent (SGD).

PROBLEM WITH STANDARD RNN GRADIENT FLOW



Vanishing gradients problems: Repeated multiplication of weight matrices during backpropagation over long sequences. Lets consider $\frac{\partial \hat{y}_t}{\partial W_{hh}}$, which is the gradient of the hidden state h_t with respect to the hidden-to-hidden weights W_{hh}.

During BPTT, at each time step, the gradient $\frac{\partial \hat{y}_t}{\partial W_{hh}}$ is computed by multiplying the gradients of subsequent time steps (computed earlier in the backpropagation process) with the corresponding weights.

If the magnitude of the weights in W_{hh}, is less than 1, and this operation is repeated over many time steps, the gradient will `diminish exponentially`. This is because each multiplication by a weight less than 1 results in a smaller value, leading to `vanishing gradients` as the backpropagation progresses backward through time.

SOLUTIONS TO THE VANISHING GRADIENT PROBLEM

Solution 1: Use of Activation Function, e.g., `Relu`.

Solution 2: Weights Initialization: Initialize Weights to `Identity Matrix` and biases to zero. This helps prevent the weights from shrinking to zero.

Solution 2: Gated Cells: Use gates (e.g., `LSTM, GRU, etc`) to selectively add or remove information within each recurrent unit.