UCS505 - Computer Graphics

Project Report

RNN VISUALIZER

BE Third Year- CSE

Sub Group: 3CS7 / 3Q22

Submitted by: Group No. 1

102217182 JYOTANSH MOHINDRU

102217207 RIJUL BANSAL

102217218 PARTH TAGGAR

Under the Mentorship of

Dr. Amrita Kaur
Assistant Professor

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
Patiala – 147001

May 2025

TABLE OF CONTENTS

1. INTRODUCTION

1.2 Project Overview

This project is an OpenGL and GLUT implementation of an interactive visualization tool that illustrates the forward pass of an untrained Recurrent Neural Network (RNN). The core objective is to provide an intuitive, step-by-step graphical representation of how RNNs process sequential data, helping users, particularly students and beginners, understand the internal operations of RNNs.

RNNs are widely used for sequence modelling tasks such as natural language processing and time series prediction. Due to their recurrent structure and hidden states, their behaviour can be difficult to grasp intuitively. This project addresses that challenge by breaking down and visually representing each computational step within the RNN.

Key features include:

- ***Interactive Step Control:***
  Users advance the visualization one timestep at a time by pressing the spacebar, enabling detailed inspection of the network's computations.

- ***Detailed Data Flow Visualization:*** Inputs, previous hidden states, outputs, and weight matrices are displayed with arrows showing the data dependencies clearly.

- ***Side-by-Side Timestep Layout with Horizontal Scrolling:*** Each timestep's RNN workspace is arranged horizontally. When the number of timesteps exceeds the visible window width, horizontal scrolling allows users to navigate and view all timesteps seamlessly.

- ***Matrix and Vector Display:*** Weight matrices and vectors are displayed in dedicated boxes, updating consistently as the sequence progresses.

This tool serves as an educational aid to demystify the "black box" nature of RNNs by visually exposing their internal computations.

1.2 Scope of the Project

The project focuses on a clear, graphical, and interactive visualization of the forward pass of an untrained RNN using OpenGL and GLUT in Python. It emphasizes instructional clarity over performance or advanced RNN features. Specific scopes include:

- ***Untrained RNN Visualization:***
  The model uses fixed, untrained weights for clarity; no training or backpropagation is implemented.
- ***Binary Input Sequences:***

  Simple predefined binary inputs are used to keep computations straightforward.

- ***Stepwise Progression:***

  Users manually progress through the sequence one step at a time via keyboard input, allowing focused study of each timestep.

- ***Component-Level Detailing:***

  Inputs, previous hidden states, current hidden states, outputs, and weight matrices are shown explicitly for each timestep.

- ***Horizontal Scrolling:***

  Added scrolling support lets the user move the viewport horizontally to explore sequences longer than the window width.

- ***No Learning Simulation:***

  The visualization purely reflects inference forward pass, not training or adaptive changes.

This project targets educational environments, workshops, or self-study for learners wishing to build foundational understanding of RNN internals and data flow.

---

2. USER DEFINED FUNCTIONS

S No.	Function Name	Function Description
1	tanh	Implements the hyperbolic tangent activation function using NumPy.
2	softmax	Computes softmax of a vector for output probability distribution.
3	rnn_step	Performs one forward step of the RNN, computing new hidden state and output from input and previous hidden state.
4	run_rnn_sequence	Iterates over input sequence, runs rnn_step for each input, and collects the states, outputs, and weights for visualization.
5	draw_text	Renders bitmap text at specified screen coordinates (OpenGL).
6	draw_box	Draws labeled rectangular boxes representing vectors/matrices (OpenGL).
7	draw_arrow	Draws arrows between boxes to indicate data flow (OpenGL).
8	grid_cell_center	Calculates cell center position in grid layout (OpenGL).
9	format_vector	Converts NumPy vector to a multiline formatted string for display.
10	format_matrix	Converts NumPy matrix to a multiline formatted string for display.
11	link	Draws arrows connecting grid cells to show data dependencies (OpenGL).
12	draw_workspace	Draws the entire RNN visualization for one timestep, including input, hidden states, outputs, and weights.
13	display	GLUT display callback rendering all active timesteps with horizontal scrolling.
14	keyboard	GLUT keyboard callback to handle stepping through the sequence (spacebar).
15	special_keys	GLUT special keys callback to handle horizontal scrolling via arrow keys.

3. CODE SNIPPETS

rnn_visualizer.py — ***OpenGL Visualization***

```
import numpy as np
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import math
from rnn_math import run_rnn_sequence  # Imports RNN math logic

# ==== Window & Grid Setup ====
W, H = 1400, 700              # Window size
GRID_R, GRID_C = 5, 5         # Grid layout (rows, columns)
CELL_W, CELL_H = 120, 120     # Size of each cell in the grid
H_GAP = 60                    # Horizontal gap between timesteps

# ==== Box Sizes for Drawing ====
BOX_W_SMALL, BOX_H_SMALL = 80, 60   # For vectors
BOX_W_LARGE, BOX_H_LARGE = 120, 80  # For weight matrices
LINE_HEIGHT = 14                    # Line spacing for text

BOX_W = BOX_W_LARGE  # Default width for arrow calculation
BOX_H = BOX_H_LARGE  # Default height for arrow calculation

# ==== Input & Computation ====
inputs = [np.array([1,0]), np.array([0,1]), np.array([1,0])]
history = run_rnn_sequence(inputs)  # Compute all timesteps
time_step = 0                       # Current visible timestep

# ==== Text Rendering ====
def draw_text(x, y, s):
    glRasterPos2f(x, y)
    for ch in s:
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, ord(ch))

# ==== Draw a Labeled Box with Optional Value ====
def draw_box(cx, cy, w, h, label, val=""):
    x0, y0 = cx - w/2, cy - h/2
    glColor3f(0,0,0)
    glBegin(GL_LINE_LOOP)
    for dx, dy in [(0,0),(w,0),(w,h),(0,h)]:
        glVertex2f(x0+dx, y0+dy)
    glEnd()
    draw_text(x0+5, y0 + h - LINE_HEIGHT, label)
    if val:
        lines = val.split("\n")
        for i, line in enumerate(lines):
            draw_text(x0+5, y0 + h - LINE_HEIGHT*(2+i), line)
```

```
# ==== Draw a Directional Arrow Between Two Points ====
def draw_arrow(x1, y1, x2, y2, lab=""):
    glColor3f(0,0,0)
    glBegin(GL_LINES)
    glVertex2f(x1,y1); glVertex2f(x2,y2)
    glEnd()
    ang = math.atan2(y2-y1, x2-x1); sz=6
    glBegin(GL_TRIANGLES)
    glVertex2f(x2,y2)
    glVertex2f(x2 - sz*math.cos(ang-0.3), y2 - sz*math.sin(ang-0.3))
    glVertex2f(x2 - sz*math.cos(ang+0.3), y2 - sz*math.sin(ang+0.3))
    glEnd()
    if lab:
        draw_text((x1+x2)/2, (y1+y2)/2 + 8, lab)

# ==== Utility: Convert Grid Position to Center Coordinates ====
def grid_cell_center(r, c, ox, oy):
    x = ox + c*CELL_W + CELL_W/2
    y = oy - r*CELL_H - CELL_H/2
    return x, y

# ==== Format Vectors and Matrices as Strings ====
def format_vector(v):
    return "\n".join(f"{val:.2f}" for val in v)

def format_matrix(m):
    return "\n".join(" ".join(f"{val:.2f}" for val in row) for row in m)

# ==== Draw Arrow From Box to Box Using Grid Coordinates ====
def link(fr, fc, tr, tc, ox, oy, box_w, box_h, lab=""):
    x1, y1 = grid_cell_center(fr, fc, ox, oy)
    x2, y2 = grid_cell_center(tr, tc, ox, oy)
    dx, dy = x2 - x1, y2 - y1
    length = math.hypot(dx, dy)
    if length == 0:
        return
    unit_dx, unit_dy = dx / length, dy / length
    x1b = x1 + unit_dx * box_w / 2
    y1b = y1 + unit_dy * box_h / 2
    x2b = x2 - unit_dx * box_w / 2
    y2b = y2 - unit_dy * box_h / 2
    draw_arrow(x1b, y1b, x2b, y2b, lab)
```

```
# ==== Draw One RNN Time Step ====
def draw_workspace(ox, oy, data):
    t = data['t']
    pos = {
      "y":      (0,2),
      "Wya":    (1,2),
      "a_prev": (2,0),
      "Waa":    (2,1),
      "RNN":    (2,2),
      "a":      (2,3),
      "Wax":    (3,2),
      "x":      (4,2),
    }
    vals = {
      "x":      format_vector(data['x']),
      "a_prev": format_vector(data['a_prev']),
      "a":      format_vector(data['a']),
      "y":      format_vector(data['y']),
      "Wax":    format_matrix(data['Wax']),
      "Waa":    format_matrix(data['Waa']),
      "Wya":    format_matrix(data['Wya']),
    }
    for label, (r,c) in pos.items():
        cx, cy = grid_cell_center(r,c,ox,oy)
        if label=="a_prev": txt = f"a<{t-1}>"
        elif label=="a":    txt = f"a<{t}>"
        else:               txt = f"{label}<{t}>"
        if label=="RNN":
            w,h = BOX_W_SMALL, BOX_H_SMALL
            v = ""
        elif label in ("Wax","Waa","Wya"):
            w,h = BOX_W_LARGE, BOX_H_LARGE
            v = vals[label]
        else:
            w,h = BOX_W_SMALL, BOX_H_SMALL
            v = vals[label]
        draw_box(cx, cy, w, h, txt, v)

    # Draw arrows showing data flow
    link(4, 2, 3, 2, ox, oy, BOX_W, BOX_H, "Wax")
    link(3, 2, 2, 2, ox, oy, BOX_W, BOX_H)
    link(2, 0, 2, 1, ox, oy, BOX_W, BOX_H)
    link(2, 1, 2, 2, ox, oy, BOX_W, BOX_H, "Waa")
    link(2, 2, 2, 3, ox, oy, BOX_W, BOX_H)
    link(2, 2, 1, 2, ox, oy, BOX_W, BOX_H)
    link(1, 2, 0, 2, ox, oy, BOX_W, BOX_H, "Wya")
```

```
# ==== Display Callback ====
def display():
    glClear(GL_COLOR_BUFFER_BIT)
    glLoadIdentity()
    if time_step == 0:
        draw_text(20,20, "Press SPACE to step through RNN")
    for i in range(time_step):
        ox = 50 + i*(GRID_C*CELL_W + H_GAP)
        oy = H - 50
        draw_workspace(ox, oy, history[i])
    glutSwapBuffers()

# ==== Keyboard Input Callback (Spacebar to Step Forward) ====
def keyboard(k, x, y):
    global time_step
    if k==b' ' and time_step < len(history):
        time_step += 1
        glutPostRedisplay()

# ==== Initialize OpenGL Projection ====
def init():
    glClearColor(1,1,1,1)
    glMatrixMode(GL_PROJECTION); glLoadIdentity()
    gluOrtho2D(0, W, 0, H)
    glMatrixMode(GL_MODELVIEW)

# ==== Main Loop ====
def main():
    glutInit()
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA)
    glutInitWindowSize(W, H)
    glutCreateWindow(b"RNN Grid Visualizer")
    glutDisplayFunc(display)
    glutKeyboardFunc(keyboard)
    init()
    glutMainLoop()

if __name__ == "__main__":
    main()
```

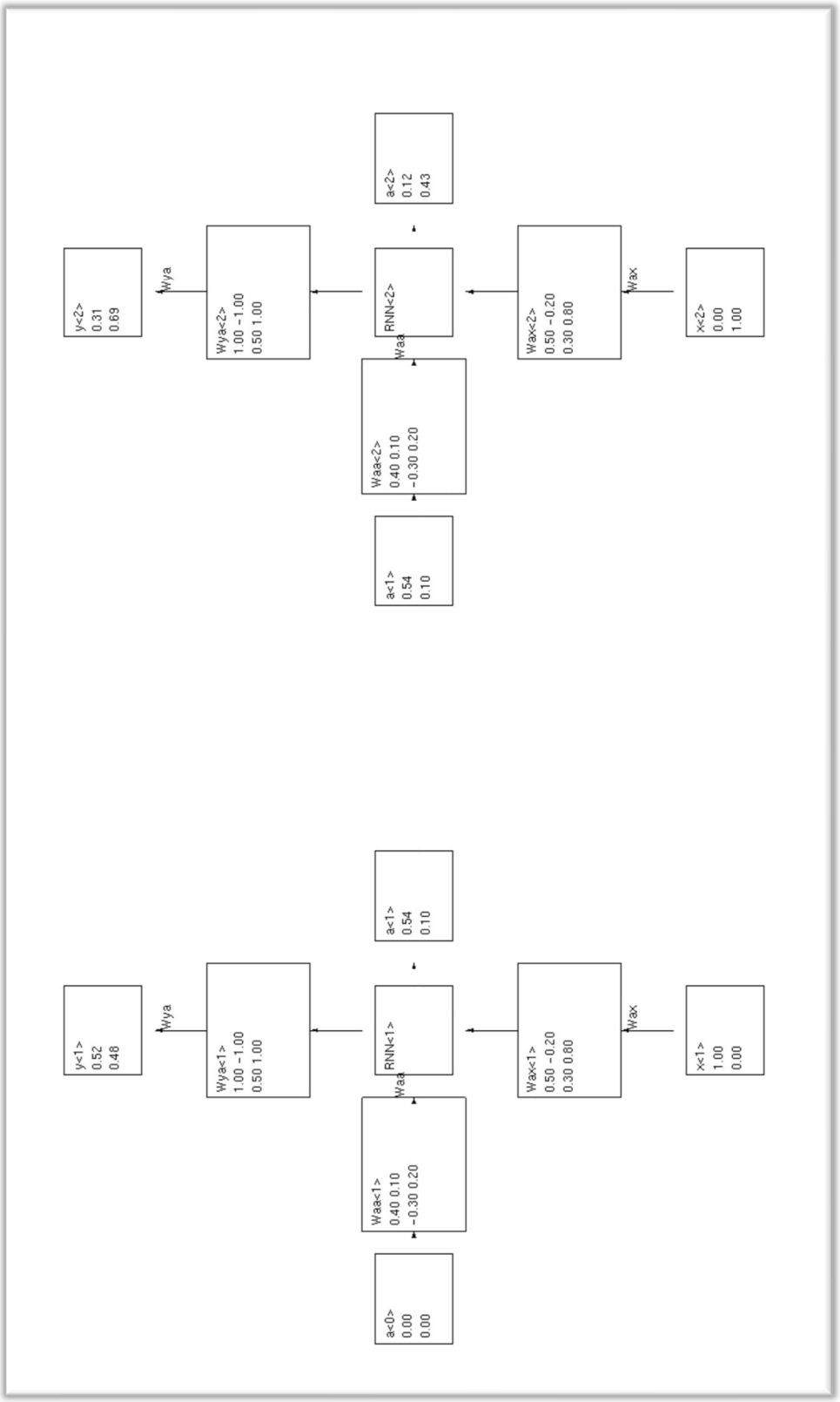
rnn_math.py – RNN Computation Logic

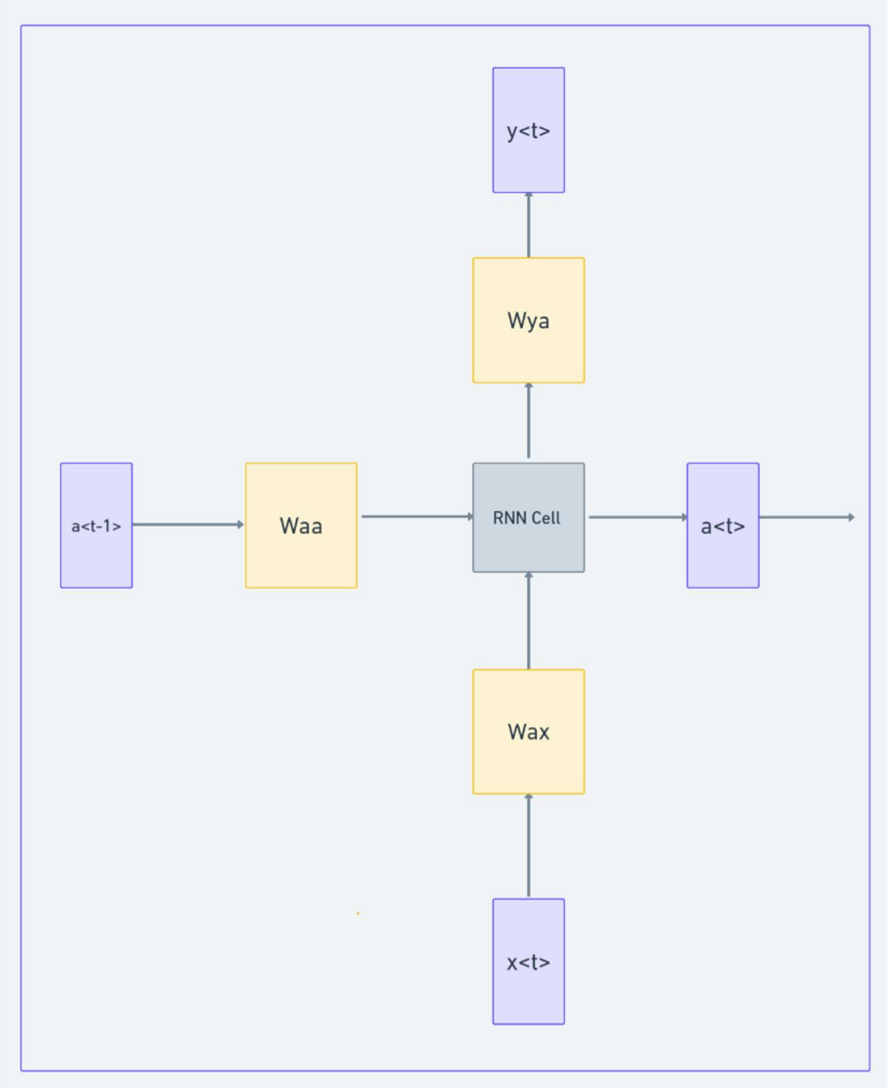
```
import numpy as np

# ==== RNN Parameters (Fixed, Untrained Weights and Biases) ====
Wax = np.array([[0.5, -0.2],[0.3, 0.8]])  # Weight from input to hidden
Waa = np.array([[0.4,  0.1],[-0.3, 0.2]]) # Weight from hidden to hidden
Wya = np.array([[1.0, -1.0],[0.5,  1.0]]) # Weight from hidden to output
ba  = np.array([0.1, -0.2])               # Hidden bias
by  = np.array([0.0,  0.0])               # Output bias

# ==== Activation Functions ====
def tanh(x):
    return np.tanh(x)

def softmax(z):
    e = np.exp(z - np.max(z))
    return e / e.sum()

# ==== Single RNN Step (Forward Pass for One Time Step) ====
def rnn_step(x, a):
    a_new = tanh(Waa.dot(a) + Wax.dot(x) + ba)
    y_new = softmax(Wya.dot(a_new) + by)
    return a_new, y_new

# ==== Runs Forward Pass Over a Sequence of Inputs ====
def run_rnn_sequence(inputs):
    """
    inputs: list of input vectors (np.array)

    Returns a list of dictionaries per time step:
      Each dictionary contains:
        'x': input vector,
        'a_prev': previous hidden state,
        'a': current hidden state,
        'y': output,
        'Wax', 'Waa', 'Wya', 'ba', 'by' (all constant matrices/vectors)
    """
    a_prev = np.zeros(Waa.shape[0])
    history = []

    for t, x in enumerate(inputs, start=1):
        a_new, y_new = rnn_step(x, a_prev)
        history.append({
            't': t,
            'x': x,
            'a_prev': a_prev,
            'a': a_new,
            'y': y_new,
```

```
        'Wax': Wax,
        'Waa': Waa,
        'Wya': Wya,
        'ba': ba,
        'by': by,
    })
    a_prev = a_new
return history
```

---

4. SCREENSHOT





5. REFERENCES

1. J. Varty, “Visualizing RNNs,” joshvarty.github.io, https://joshvarty.github.io/VisualizingRNNs/
2. GLUT API documentation – FreeGLUT project. http://freeglut.sourceforge.net/docs/api.php

END OF REPORT