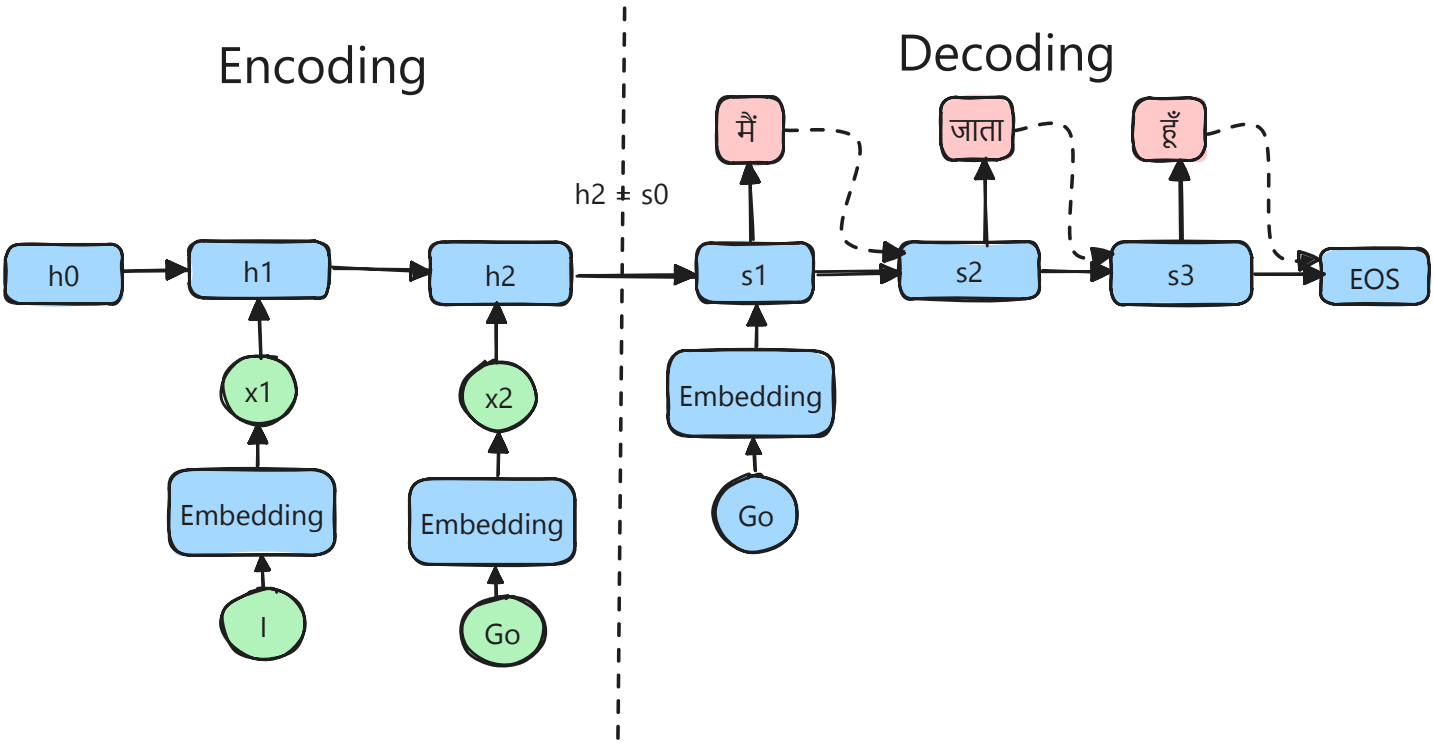
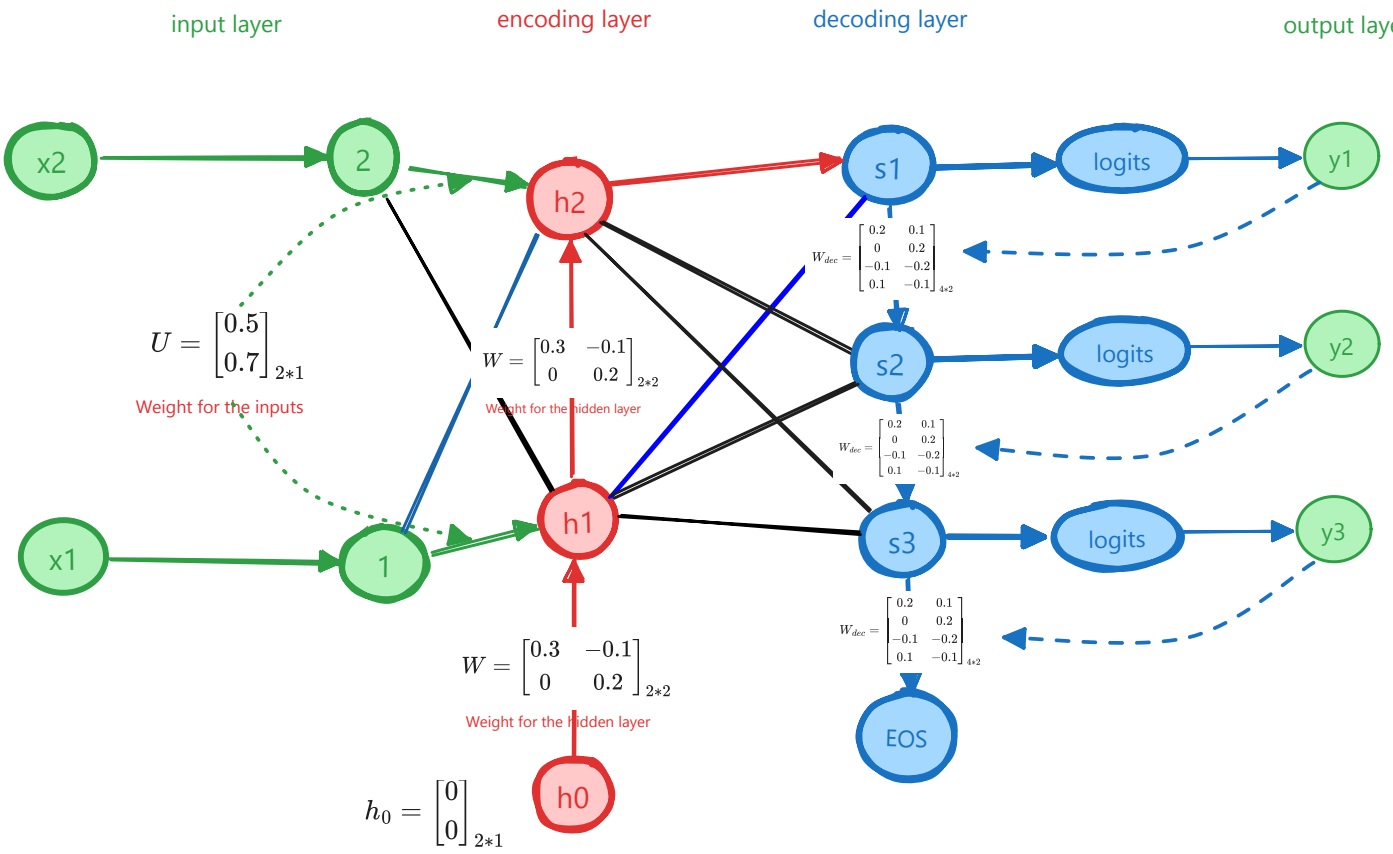
RNN - Recurrent neural network

English to Hindi translation using RNN

Block diagram for RNN





Mathematical relationships for RNN

1. Equation for the hidden state at time t -
   $h_t = \tanh(Wh_{t-1} + Ux_t + b)$: function of the hidden state at time t-1, the input at time t and the bias term.
2. Equation for the output at time t -
   $y_t = softmax(Vh_t + c)$

Steps in RNN

Step#	Description for Encoder	Description for Decoder
1	Embed inputs	Create embeddings for vocabulary
2	Decide number of hidden layers and states	Decide number of hidden layers and states
3	Initialize the 1st hidden state h0	Initialize 1st hidden state
4	Activation function $h_t = \tanh(Wh_{t-1} + Ux_t + b)$	Activation function $s_t = \tanh(W_{dec}s_{t-1} + Vy_t + c)$
5	initialize the weights and bias for hidden states	Initialize the weights and bias for hidden states
6	Calculate all the hidden states of encoder using step 4	Calculate the 1st output state

Step#	Description for Encoder	Description for Decoder
7	Encoder work is done. pass on to decoder last state	Calculate logits matrix for 1st output state
8	N/A	Calculate probability distribution of output using softmax()
9	N/A	Determine the word corresponding to the highest probability
10	N/A	Compute all the output states repeating steps 6 to 9 for all states

Recurrent Neural Network RNN

Objective :

Translate English sentence - "I go" to Hindi sentence - "मैं जाता हूँ" using Recurrent neural network

Encoding :

Step 1 : Embedding inputs

Convert the input tokens into embeddings.
Let x("I") = x1 = 1
Let x("go") = x2 = 2

Step 2 : Decide number of hidden layers & states in the hidden layer.

Decide number of hidden layers in the neural network and number of states in each layer.
Let hidden state size = s = 2. Number of layers of the neural network = n = 1

Step 3 : Initialize the 1st hidden state of the encoder

Initialize the 1st hidden state h_0 based on the hidden size s. It will be a matrix of dimensions s x n = 2x1. Number of rows = Number of hidden states in the layer. Number of columns = Number of layers in the neural network. **The matrix is like a neural network standing erect.**

$$h_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{2*1}$$

Step 4 : Mathematical relation between the hidden state

Mathematical relation between the hidden state -

$$h_t = \tanh(Wh_{t-1} + Ux_t + b)$$

where $\tanh()$ is the activation function, W is the weight matrix for the hidden state, U is the weight matrix for the input, b is the bias

Step 5 : Initialize the weights and biases of the neural network randomly

$$W = \begin{bmatrix} 0.3 & -0.1 \\ 0 & 0.2 \end{bmatrix}_{2*2}$$

$$U = \begin{bmatrix} 0.5 \\ 0.7 \end{bmatrix}_{2*1}$$

$$b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{2*1}$$

Code for Step 5

```
import numpy as np
# Initialization of the weight and biases for the neural network for encoder
W = np.array([[0.30, -0.10], [0, 0.20]])
h0 = np.array([[0.0], [0.0]])
U = np.array([[0.50], [0.70]])
b = np.array([[0.0], [0.0]])
x1 = 1
x2 = 2
```

Step 6 : Calculate the hidden states h_1 and $h2$ using the formula in step 4

$h_1 = \tanh(Wh_0 + Ux1 + b)$

$$h_1 = \begin{bmatrix} 0.46 \\ 0.60 \end{bmatrix}$$

$h_2 = \tanh(Wh_1 + Ux2 + b)$

$$h_2 = \begin{bmatrix} 0.79 \\ 1.91 \end{bmatrix}$$

Code for Step 6

```
# Matrix multiplication
h1 = np.tanh(np.matmul(W, h0) + U * x1 + b)
h2 = np.tanh(np.matmul(W, h1) + U * x2 + b)

print("h1:\n", h1)
print("h2:\n", h2)
```

```
h1:
 [[0.46211716]
 [0.60436778]]
h2:
 [[0.79253003]
 [0.90884977]]
```

Decoding

Step 1 : Create embeddings of the Hindi words.

- Let y(Go) = y1 = 0.5,
- y("मैं") = y2 = 1,
- y("जाता") = y3 = 1.1,
- y("हूँ") = y4 = 0.9,
- y(EOS) = y5 = 0.0

Step 2 : Decide number of hidden layers & state

It remains same as that of the encoder.

Step 3 : Initialize the 1st hidden state

The first output layer will be a copy of the last hidden state of the encoder.

$$s_0 = h_2 = \begin{bmatrix} 1.08 \\ 1.54 \end{bmatrix}$$

Step 4 : Mathematical relation between the hidden state

Mathematical relation between the hidden state -

$$s_t = \tanh(W_{dec} s_{t-1} + V y_t + c)$$

where $\tanh()$ is the activation function, W' is the weight matrix for the hidden state, V is the weight matrix for the output

y, c is the bias

Step 5 : Initialize the weights and biases of the neural network randomly

$$W_{dec} = \begin{bmatrix} 0.2 & 0.1 \\ 0.3 & 0.4 \end{bmatrix}_{2*2}$$

$$V = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}_{2*1}$$

$$d = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{2*1}$$

Code for Step 5

```
# Initialization of the weight and biases for the neural network for decoder
W_dec = np.array([[0.20, 0.10], [0.3, 0.40]])
s0 = h2
V = np.array([[0.10], [0.20]])
c = np.array([[0.0], [0.0]])
y1 = 0.5
y2 = 1
y3 = 1.1
y4 = 0.9
y5 = 0.0
```

Step 6 : Calculate the output state s_1

$s_1 = \tanh(W_{dec}h_2 + Vy_1 + c)$

$s_1 = \begin{bmatrix} 0.5 \\ 0.7 \end{bmatrix}$

Step 7 : Compute logits matrix (output matrix) for s_1

initialize W_out and b_out

$$W_{out} = \begin{bmatrix} 0.2 & 0.1 \\ 0 & 0.2 \\ -0.1 & -0.2 \\ 0.1 & -0.1 \end{bmatrix}_{4*2}$$

$$b_{out} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{4*1}$$

Determine logits1 matrix corresponding to s1

$$logits_1 = W_{out} s_t + b_{out}$$

$$logits_1 = \begin{bmatrix} 0.119 \\ 0.121 \\ -0.150 \\ -0.031 \end{bmatrix}$$

Step 8 : Convert logits1 to probability values via softmax

each row gives the probability of मै जाता हूँ and EOS respectively. The highest probability is that of जाता. Hence the next word is जाता.

$$softmax(logits_1) = \begin{bmatrix} 0.2756 \\ 0.2763 \\ 0.2107 \\ 0.2372 \end{bmatrix}$$

Step 9 : Determine the word based on the highest probability

The word corresponding to the highest probability is जाता. Hence the next word is जाता.

Step 10 : Repeat steps 6 to 9 to determine the next word

Code :

```
# Matrix multiplication
from scipy.special import softmax
s1 = np.tanh(np.matmul(W_dec, h2) + V * y1 + c)
print("s1:\n", s1)
W_out = [[0.2,0.1],[0,0.2],[-0.1, -0.2],[0.1,-0.1]]
b_out = [[0.0],[0.0],[0.0],[0.0]]
logits1 = np.matmul(W_out, s1) + b_out
print("logits1:\n", logits1)
probability = softmax(logits1, axis=0)
print("probability:\n", probability)
```

```
s1:
 [[0.29075518]
 [0.6051916 ]]
logits1:
 [[ 0.1186702 ]
 [ 0.12103832]
 [-0.15011384]
 [-0.03144364]]
probability:
 [[0.27568797]
 [0.27634161]
 [0.2107106 ]
 [0.23725982]]
```

Few observations

1. The hidden state at time t is a function of the hidden state at time t-1, the input at time t and the bias term.
2. W and U are the weight matrices or parameters. That are trained using the training set. In the beginning, they are randomly initialized. And then they are trained, using the corpus so that the loss is minimized.
3. b is the bias term. It is also randomly initialized and then trained.
4. x_t is the input at time t. It is the embedding of the token at time t.
5. h_t is the hidden state at time t. It is the memory of the network at time t.