RNN

September 24, 2025 10:31 AM

RNN
Recurrent Neural Networks

• Used for 'sequential' data (where order/time matters)

Given a sequence,
predict next item in sequence

• Like CNNs look at neighbors (spatial models),
RNNs look at past & future (temporal models)
(sometimes)

e.g. $x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
$\quad\quad\quad\; t_1 \;\; t_2 \;\; t_3 \; \ldots \quad\quad\quad\quad\quad t_{10}$

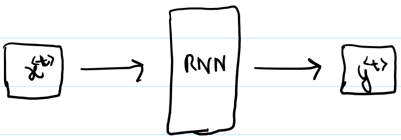
given $x^{<t>}$, predict $x^{<t+1>}$

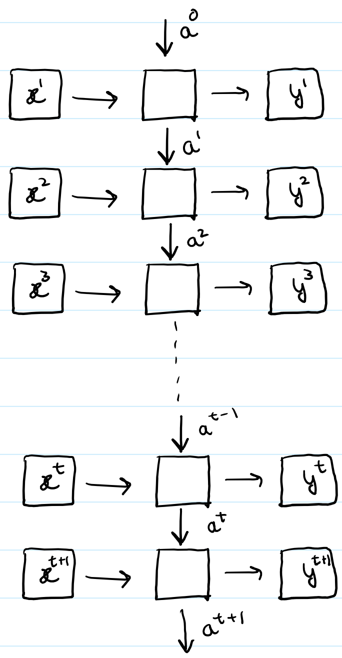
let $x^{<t+1>}$ i.e. our target be y & since we're using x^t at time t, lets call it y^t

i.e. $y^{<t>} = x^{<t+1>}$

$\Rightarrow$ given $x^{<t>}$, predict $y^{<t>}$

e.g. at $t = 1$, $x^{<1>} = 1$, $y^{<1>} = x^{<2>} = 2$
$t = 2$, $x^{<2>} = 2$, $y^{<2>} = x^{<3>} = 3$





Basically a Feed Forward NN 'unrolled' across time

We want our RNN to sort of keep a memory of every x^t that passes through
& we do it through the hidden state that model learns
this hidden state stores memory upto a point in sequence (t)

In a standard neural network a hidden layer is usually defined as:

$h = \text{activation}(Wx + b)$

In RNN:

standard act. functions of Neurons

$h^{<t>} = \text{activation function}(\text{function}(h^{<t-1>}, x^{<t>}))$

hidden state at time 't' (memory upto point t)

function defining relationship of previous inputs to current

previous hidden state (memory)

current input (adding current to memory)

Now for function describing (h_{t-1}, x_t), we use linear function

$$f(h_{t-1}, x_t) = \underbrace{W_{hh}}_{} h^{<t-1>} + \underbrace{W_{xh}}_{\text{Learnt coeff (weights)}} x^{<t>} + \underbrace{b_h}_{\text{bias}}$$

∴ The hidden layer becomes:

$$h^{\langle t \rangle} = \underbrace{a}_{\text{tanh}} \left(W_{hh} h^{\langle t-1 \rangle} + W_{xh} x^{\langle t \rangle} + b_h \right)$$

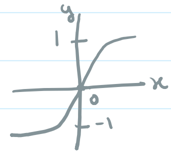
& ∴ the output becomes:

$$y^{\langle t \rangle} = \underbrace{a}_{\text{None}} \left(W_{hy} h^{\langle t \rangle} + b_y \right)$$

★ $\tanh(x) \rightarrow [-1, 1]$
centered around 0

as $x \rightarrow \infty$
$(\tanh(x))' \rightarrow 0$
ie. gradient vanishes

Default for Pytorch's RNN



Lets walk through our example
with a simple RNN which we define to have an,

input size = 1
hidden size = 1
output size = 1
tanh activation

$$h^{\langle t \rangle} = \tanh\left(W_{hh} h^{\langle t-1 \rangle} + W_{xh} x^{\langle t \rangle} + b_h \right)$$
$$y^{\langle t \rangle} = W_{hy} h^{\langle t \rangle} + b_y$$

Step 1 Initialize Weights

W_{hh}	W_{xh}	b_h	W_{hy}	b_y	h^0
0.5	0.8	0	1.2	0	0

Step 2 Forward pass

Compute $h^{\langle t \rangle}$ & $y^{\langle t \rangle}$ for every t where $t = 1 \ldots t-1$ ie. (1,10) in our case

for $t = 1$, $h^{\langle 1 \rangle} = \tanh\left(0.5(0) + 0.8(1) + 0\right) = 0.664$
$y^{\langle 1 \rangle} = 1.2(0.664 + 0) = 0.797$

$t = 2$, $h^{\langle 2 \rangle} = \tanh\left(0.5(0.664) + 0.8(2) + 0\right) = 0.958$
$y^{\langle 2 \rangle} = 1.2(0.958) = 1.150$

.... and so on.

Step 3 Calculate Loss function

Forgot to mention before, but let's take MSE

At time 't'

$$\mathcal{L}^{\langle t\rangle} = \frac{1}{2}(\underbrace{\hat{y}^{\langle t\rangle}}_{\text{Predicted}} - \underbrace{y^{\langle t\rangle}}_{\text{Original}})^2$$

For full sequence,

$$\mathcal{L}_{total} = \sum_t \mathcal{L}^{\langle t\rangle} = \sum_t \frac{1}{2}(\hat{y}^{\langle t\rangle} - y^{\langle t\rangle})^2$$

objective function to minimize

For our example, this'll be,

$$\mathcal{L}^{\langle 1\rangle} = \frac{1}{2}(0.797-2)^2 = 0.719$$

$$\mathcal{L}^{\langle 2\rangle} = \frac{1}{2}(1.150-3)^2 = 1.711$$

$$\ldots \quad \mathcal{L}^{\langle 9\rangle}$$

Step 4 Backpropagation Through Time
BPTT

compute gradients

$$\frac{\partial \mathcal{L}}{\partial W_{hh}}, \frac{\partial \mathcal{L}}{\partial W_{xh}}, \frac{\partial \mathcal{L}}{\partial W_{hy}}, \frac{\partial \mathcal{L}}{\partial b_h}, \frac{\partial \mathcal{L}}{\partial b_y}$$

$$\Rightarrow \sum_t \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial W_{hh}}, \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial W_{xh}}, \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial W_{hy}}, \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial b_h}, \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial b_y}$$

this is why we call it 'unrolling'

$$\boxed{\frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial \hat{y}} = \hat{y}^{\langle t\rangle} - y^{\langle t\rangle}}$$

where,

$$\frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial W_{hh}} = \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}^{\langle t\rangle}}{\partial h^{\langle t\rangle}} \cdot \frac{\partial h^{\langle t\rangle}}{\partial W_{hh}} = (\hat{y}^{\langle t\rangle} - y^{\langle t\rangle})\, W_{hy}\,(1-h^{\langle t\rangle})^2\, h^{t-1}$$

$$\frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial W_{xh}} = \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}^{\langle t\rangle}}{\partial h^{\langle t\rangle}} \cdot \frac{\partial h^{\langle t\rangle}}{\partial W_{xh}} = (\hat{y}^{\langle t\rangle} - y^{\langle t\rangle})\, W_{hy}\,(1-h^{\langle t\rangle})^2\, x^{\langle t\rangle}$$

$$\frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial W_{hy}} = \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}^{\langle t\rangle}}{\partial W_{hy}} = (\hat{y}^{\langle t\rangle} - y^{\langle t\rangle})\,(h^{\langle t\rangle})$$

$$\frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial b_h} = \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial \hat{y}}\, \frac{\partial \hat{y}^{\langle t\rangle}}{\partial h^{\langle t\rangle}}\, \frac{\partial h^{\langle t\rangle}}{\partial b_h} = (\hat{y}^{\langle t\rangle} - y^{\langle t\rangle})\, W_{hy}\,(1-h^{\langle t\rangle})^2$$

$$\frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial b_y} = \frac{\partial \mathcal{L}^{\langle t\rangle}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b_y} = (\hat{y}^{\langle t\rangle} - y^{\langle t\rangle})\,(1)$$

Apply gradient descent

$$\theta = \{W_{hy}, W_{xh}, W_{hh}, b_h, b_y\}$$

$$\underbrace{\overset{\text{new}}{\theta}}_{\text{New param}} := \underbrace{\theta}_{\text{Existing param}} - \underbrace{\alpha}_{\text{Learning Rate}} \underbrace{\frac{\partial \mathcal{L}}{\partial \theta}}_{\text{Gradient computed w.r.t param above}}$$

Types of RNN

- Long Short Term Memory
  - For long range dependencies
  - Complex cell (input, forget & output gate)
  - Decides what to remember or forget
- Gated Recurrent Unit
  - LSTM Lite
  - Merges some gates for computational advantage
- Bidirectional RNN
  - Past + Future
- Deep RNN
  - Multiple RNNs / Skip / Residual connections