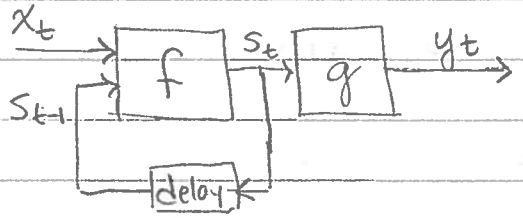
L 10-1

Recurrent neural networks

Learning to be a state machine!

Recall:



Assume x_t is $l \times 1$ — f_1 and f_2 are activation fcns

s_t is $m \times 1$

Change in notation to match latex: "V"

y_t is $V \times 1$

Parameters W^{sx} is $m \times l$

W^{ss} is $m \times m$ — $W_\emptyset$ is $m \times 1$

W^o is $V \times m$ — $W^o_\emptyset$ is $V \times 1$

always good to check dimensions!

$$s_t = f_1\left(W^{sx} x_t + W^{ss} s_{t-1} + W_\emptyset\right)$$

$$y_t = f_2\left(W^o s_t + W^o_\emptyset\right)$$

Training seq to seq (c.f. Dancing cheek to cheek)

Training set: $[(x^{(1)}, y^{(1)}), \ldots, (x^{(q)}, y^{(q)})]$

- $x^{(i)}$ and $y^{(i)}$ are length $n^{(q)}$ sequences
- the sequences in the same pair are the same length
- sequences in different pairs may have different lengths

Loss function: let $\Theta = (W^{sx}, W^{ss}, W^o, W_\emptyset, W^o_\emptyset)$

$$L(\Theta) = \sum_{i=1}^{q} Loss_{seq}\left(y^{(i)}, RNN(x^{(i)}; \Theta)\right)$$

(RNN(...): result of running transduce on x)

(need loss function on sequences.)

Usually:

$$Loss_{seq}\left(y^{(i)}, p^{(i)}\right) = \sum_{j=1}^{n^{(q)}} Loss\left(y^{(i)}_j, p^{(i)}_j\right)$$

sorry i"

Change in notation to match latex: "V"

s_t is $m \times 1$ activation fcns

y_t is $V \times 1$

Parameters W^{sx} is $m \times l$

W^{ss} is $m \times m$ $W_\emptyset$ is $m \times 1$

W^o is $V \times m$ $W^o_\emptyset$ is $V \times 1$

always good to check dimensions!

$$s_t = f_1(W^{sx} x_t + W^{ss} s_{t-1} + W_\emptyset)$$

$$y_t = f_2(W^o s_t + W^o_\emptyset)$$

Training seq to seq (c.f. Dancing cheek to cheek)

Training set: $[(x^{(1)}, y^{(1)}), \ldots, (x^{(q)}, y^{(q)})]$

- $x^{(i)}$ and $y^{(i)}$ are length $n^{(q)}$ sequences
- the sequences in the same pair are the same length
- sequences in different pairs may have different lengths

Loss function: let $\Theta = (W^{sx}, W^{ss}, W^o, W_\emptyset, W^o_\emptyset)$

$$L(\Theta) = \sum_{i=1}^{q} \text{Loss}_{seq}(y^{(i)}, \text{RNN}(x^{(i)}; \Theta))$$

(RNN(...): result of running transduce on x)

need loss function on sequences.

Usually:

$$\text{Loss}_{seq}(y^{(i)}, p^{(i)}) = \sum_{j=1}^{n^{(q)}} \text{Loss}(y^{(i)}_j, p^{(i)}_j)$$

Sorry i

sum of losses along sequence

note: ~~this is consistent~~ with Latex'ed notes but inconsistent with some of my prev notes which puts arguments in other order

L 10-2

Our choice of per-element loss and output activation function f_2 go together, and depend on the problem. Might be

y_j	f_2	Loss
real	linear	squared
one-hot	softmax	NLL
	etc.	

f_1 is typically tanh

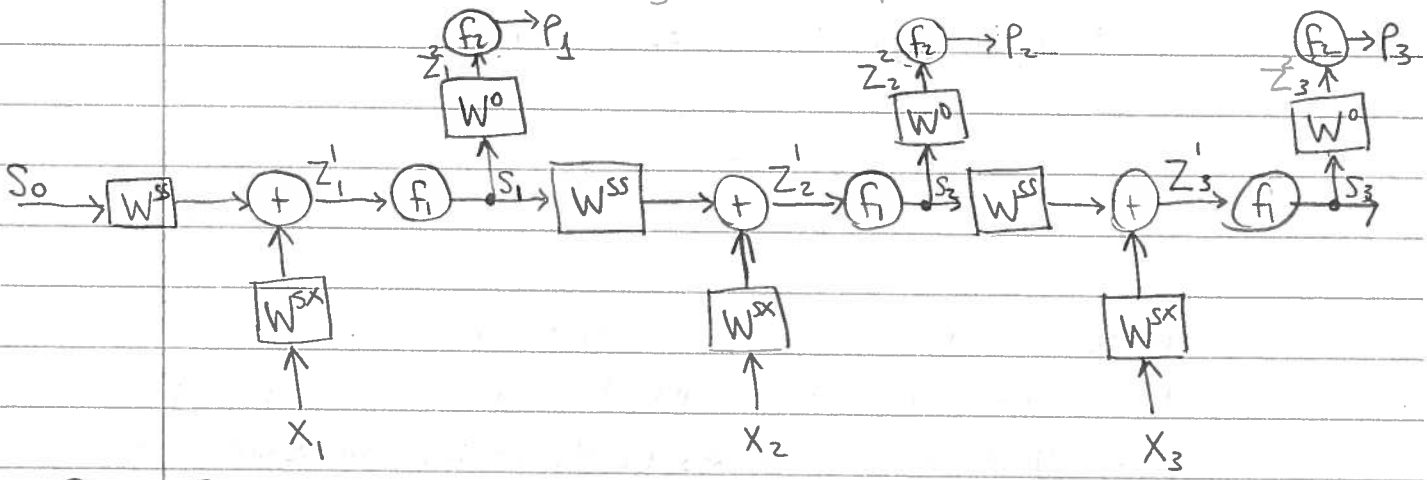
How to find θ to minimize loss on our training data?
gradient descent! (BPTT)

Easy way to think about it: backpropagation through time

Do basic SGD:

① Sample training pair (x, y); let its length be n

② "Unroll" the RNN to length T (picture for $T=3$ below)



③ Do forward pass computing:

$$Z_t' = W^{sx} x_t + W^{ss} S_{t-1} + W_\emptyset$$

$$S_t = f_1(Z_t')$$

$$Z_t^2 = W^0 S_t + W_\emptyset^0$$

"predicted" output = $P_1 \dots P_T$

$$P_t = f_2(Z_t^2)$$

④ Do backward pass to compute the gradients.

We need for all our W^{ss}, W^{sx}

$$\frac{\partial L}{\partial W} = \sum_{v=1}^{n} \frac{\partial L_v}{\partial W}$$

$$= \sum_{v=1}^{n} \sum_{t=1}^{n} \frac{\partial L_v}{\partial s_t} \cdot \frac{\partial s_t}{\partial W}$$

Total derivative: Sum over all paths through which W affects L_t

$$= \sum_{t=1}^{n} \frac{\partial s_t}{\partial W} \cdot \sum_{v=1}^{n} \frac{\partial L_v}{\partial s_t}$$

re-organizing

$$= \sum_{t=1}^{n} \frac{\partial s_t}{\partial W} \cdot \sum_{v=t}^{n} \frac{\partial L_v}{\partial s_t}$$

s_t only affects $L_v, L_{v+1}, \ldots L_n$

$$= \sum_{t=1}^{n} \frac{\partial s_t}{\partial W} \left[\frac{\partial L_t}{\partial s_t} + \underbrace{\sum_{v=t+1}^{n} \frac{\partial L_v}{\partial s_t}}_{\text{define } \delta^{s_t}} \right]$$

δ^{s_t} is the dependence of the loss on steps after t on the state at time t.

We can compute this backwards, with t going from n down to 1.

L 10-4

$$F_t = \sum_{v=t+1}^{n} \text{Loss}(y_v, p_v)$$ future loss

Trickiest part is figuring out how early states contribute to late losses.

At the last stage, $F_n = 0$ so $\delta^{S_n} = 0$

Working backwards:

$$\delta^{S_{t-1}} = \frac{\partial}{\partial S_{t-1}} \sum_{j=t}^{n} \text{Loss}(y_j, p_j)$$

$$= \frac{\partial S_t}{\partial S_{t-1}} \cdot \frac{\partial}{\partial S_t} \sum_{j=t}^{n} \text{Loss}(y_j, p_j)$$

$$= \frac{\partial S_t}{\partial S_{t-1}} \cdot \frac{\partial}{\partial S_t} \left[\text{Loss}(y_t, p_t) + \sum_{j=t+1}^{n} \text{Loss}(y_j, p_j) \right]$$

$$= \frac{\partial S_t}{\partial S_{t-1}} \cdot \left[\frac{\partial \text{Loss}(y_t, p_t)}{\partial S_t} + \delta^{S_t} \right]$$

Now, we need

$$\frac{\partial \text{Loss}(y_t, p_t)}{\partial S_t} = \frac{\partial Z_t^2}{\partial S_t} \cdot \frac{\partial \text{Loss}(y_t, p_t)}{\partial Z_t^2}$$

$(m \times 1)$ $(m \times v)$ $(v \times 1)$

W_0^T

$$\frac{\partial S_t}{\partial S_{t-1}} = \frac{\partial Z_t^1}{\partial S_{t-1}} \cdot \frac{\partial S_t}{\partial Z_t^1} = W_{ss}^T * f''(Z_t^1)$$

$(m \times m)$ $(m \times m)$ $(m \times m)$ ↑ not dot!

$$W_{ss}^T \begin{bmatrix} f'(Z_{t,1}) & & \\ & 0 & \\ & & f'(Z_{t,m}) \end{bmatrix}$$

So: $$\delta^{S_{t-1}} = W_{ss}^T * f''(Z_t^1) \cdot \left(W_0^T \frac{\partial L_t}{\partial Z_t^2} + \delta^{S_t} \right)$$

At the last stage, $F_n = 0$ so $\delta^{S_n} = 0$

Working backwards:

$$\delta^{S_{t-1}} = \frac{\partial}{\partial S_{t-1}} \sum_{j=t}^{n} \text{Loss}(y_j, p_j)$$

$$= \frac{\partial S_t}{\partial S_{t-1}} \cdot \frac{\partial}{\partial S_t} \sum_{j=t}^{n} \text{Loss}(y_j, p_j)$$

$$= \frac{\partial S_t}{\partial S_{t-1}} \cdot \frac{\partial}{\partial S_t} \left[\text{Loss}(y_t, p_t) + \sum_{j=t+1}^{n} \text{Loss}(y_j, p_j) \right]$$

$$= \frac{\partial S_t}{\partial S_{t-1}} \cdot \left[\frac{\partial \text{Loss}(y_t, p_t)}{\partial S_t} + \delta^{S_t} \right]$$

Now, we need

$$\frac{\partial \text{Loss}(y_t, p_t)}{\partial S_t} = \frac{\partial Z_t^2}{\partial S_t} \cdot \frac{\partial \text{Loss}(y_t, p_t)}{\partial Z_t^2}$$

$(m \times 1)$ $(m \times v)$ $(v \times 1)$

W_0^T

$$\frac{\partial S_t}{\partial S_{t-1}} = \frac{\partial Z_t^1}{\partial S_{t-1}} \cdot \frac{\partial S_t}{\partial Z_t^1} = W_{ss}^T * f''(Z_t^1)$$

$(m \times m)$ $(m \times m)$ $(m \times m)$

↑ not dot!

$$W_{ss}^T \begin{bmatrix} f'(Z_{t1}) & & \\ & \ddots & \\ 0 & & f'(Z_{tm}) \end{bmatrix}$$

So: $$\delta^{S_{t-1}} = \underbrace{W_{ss}^T * f''(Z_t^1)}_{\frac{\partial S_t}{\partial S_{t-1}}} \cdot \underbrace{\left(W_0^T \frac{\partial L_t}{\partial Z_t^2} + \delta^{S_t} \right)}_{\frac{\partial F_{t-1}}{\partial S_t}}$$

Weight updates

As we iterate backwards, sum up

$$\frac{\partial L}{\partial W^{ss}} \mathrel{+}= \frac{\partial s_t}{\partial W^{ss}} \cdot \frac{\partial F_{t-1}}{\partial s_t} = \frac{\partial z_t^1}{\partial W^{ss}} \cdot \frac{\partial s_t}{\partial z_t^1} \cdot \frac{\partial F_{t-1}}{\partial s_t}$$

s_{t-1} $f'(z_t^1)$ prev result

$$\frac{\partial L}{\partial W^{sx}} \mathrel{+}= \frac{\partial s_t}{\partial W^{sx}} \cdot \frac{\partial F_{t-1}}{\partial s_t} = \frac{\partial z_t^1}{\partial W^{sx}} \cdot \frac{\partial s_t}{\partial z_t^1} \cdot \frac{\partial F_{t-1}}{\partial s_t}$$

x_t $f'(z_t^1)$ prev result

Handle W^0 separately (it's easy):

$$\frac{\partial L}{\partial W^0} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial W^0} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial z_t^2} \cdot \frac{\partial z_t^2}{\partial W^0}$$

Assume we have $\partial L_t / \partial z_t^2 = (p_t - y_t)$
(true for quadratic, softmax-NLL, etc)

On each iteration

$$\frac{\partial L}{\partial W_0} \mathrel{+}= (p_t - y_t) \cdot s_t^T$$

$V \times m$ $(V \times 1)$ $(1 \times m)$

whew!

Can train a language model to predict next token: $c_{t} = \text{RNN}(c_1, \ldots, c_{t-1})$ c is character or word

by making training pairs

$x = (\langle \text{start} \rangle, c_1, c_2, c_3, \ldots, c_T)$

$y = (c_1, c_2, c_3, \ldots, c_T, \langle \text{end} \rangle)$

S_{t-1} $f'(z^1_t)$ prev result

$$\frac{\partial L}{\partial W^{sx}} \mathrel{+}= \frac{\partial s_t}{\partial W^{sx}} \cdot \frac{\partial F_{t-1}}{\partial s_t} = \frac{\partial z^1_t}{\partial W^{sx}} \cdot \frac{\partial s_t}{\partial z^1_t} \cdot \frac{\partial F_{t-1}}{\partial s_t}$$

x_t $f''(z^1_t)$ prev result

Handle W^0 separately (it's easy):

$$\frac{\partial L}{\partial W^0} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial W^0} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial z^2_t} \cdot \frac{\partial z^2_t}{\partial W^0}$$

Assume we have $\partial L_t / \partial z^2_t = (p_t - y_t)$
(true for quadratic, softmax-NLL, etc)

On each iteration

$$\frac{\partial L}{\partial W_0} \mathrel{+}= (p_t - y_t) \cdot s_t^T$$

$V \times m$ $(V \times 1)$ $(1 \times m)$

whew!

Can train a language model to predict next token: $c_{t} = \text{RNN}(c_1, \ldots, c_{t-1})$ c is character or word

by making training pairs

$x = (\langle\text{start}\rangle, c_1, c_2, c_3, \ldots, c_T)$

$y = (c_1, c_2, c_3, c_4, \ldots, \langle\text{end}\rangle)$