5- recurrent neural networks

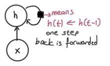
☆ typically for processing sequential data

☆ share parameters through time → otherwise cannot generalize to unknown sequence length

☆ have cycles / feedback connections

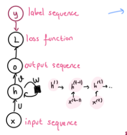
$$h^{(t)} = f(h^{(t-1)}; x^{(t)}; \theta)$$

$h^{(t)}$: hidden state (fixed length); f: model; $x^{(t)}$: input (arbitrary length), t: time; θ: shared parameters



ex: I love apples ($x^{(1)}$ $x^{(2)}$ $x^{(3)}$) → $h^{(3)} = f(f(f(h^{(0)}; x^{(1)}; \theta); x^{(2)}; \theta); x^{(3)}; \theta) = g^{(t)}(x^{(1)}, x^{(2)}, \dots x^{(t)}; \theta)$

* if we used this, diff param for each, and could work only with fixed len



→ turing machine

$$a^{(t)} = b + W h^{(t-1)} + U x^{(t)}$$
$$h^{(t)} = \tanh(a^{(t)})$$
$$o^{(t)} = c + V \cdot h^{(t)}$$
$$\hat{y}^{(t)} = \mathrm{softmax}(o^{(t)})$$

* U, W, V, c, b are learnable parameters

$$L(\{x_1, x_2 \dots\}, \{y_1, y_2 \dots\}) = \sum_t L^{(t)}$$

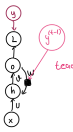
* there is an output for each input in the sequence

☆ the gradient of L wrt $h^{(t)}$ depends on all the h's up to it

BPTT = back propagation through time

☆ cannot paralelize

☆ time and memory comlexity = $O(t)$ ↳ len



simpler → less powerful, not a turing machine, more efficient training

teacher forcing = instead of using the output of previous input, it uses its grand truth, parallelized linear time training

bidirectional RNNs = output at t depends on both the past and the future

encoder-decoder sequence-to-sequence architecture =

↳ encoder RNN reads / processes the input sequence, then emits the learned content C

↳ decoder produces the output sequence based on C

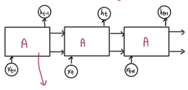
vanishing or exploding gradient = weight will be w^t at RNN with t layers / sequence

☆ either will explode ($w_0 > 1$) else vanish ($w_0 < 1$)

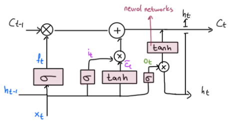
☆ gradient clipping = $\hat{g} \leftarrow \text{threshold} / \|\hat{g}\| * \hat{g}$ for exploding gradient

☆ regularizing the gradient = for vanishing gradient

long short-term memory (LSTM)



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$\tilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$



f_t: forget gate (the percentage to remember)

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

i_t: input gate (update the long term memory)

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

O_t: output gate

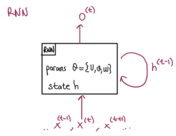
$$O_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

gated recurrent unit (GRU) = simpler LSTM

↳ the forget and input gate combined into a single update gate

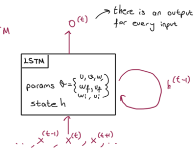
↳ merges the cell and the hidden state (no C_t)

RNN



$$h^{(t)} = \sigma(W \cdot h^{(t-1)} + U \cdot x^{(t)})$$
$$o^{(t)} = \sigma(V h^{(t)})$$

LSTM



$$\tilde{h}^{(t)} = \sigma(W \cdot h^{(t-1)} + U \cdot x^{(t)})$$
$$h^{(t)} = \underbrace{f}_{\text{forget gate}} \cdot h^{(t-1)} + \underbrace{i}_{\text{input gate}} \cdot \tilde{h}^{(t)}$$

$$f = \sigma(W_f \cdot h^{(t-1)} + U_f \cdot x^{(t)}) \qquad i = \sigma(W_i \cdot h^{(t-1)} + U_i\, x^{(t)})$$

RNN applications =

☆ character level language modeling = predicting next char in a given sequence

↳ one-hot encoding of characters

☆ word level language modeling =

↳ word embeddings = to represent each word with fixed-dimensions vectors

word2vec = widely used word-embedding model (2013)

↳ tends to map semontically similar words to nearby points

↳ (<content>, <word>) pairs: x y z t ... (sentence) ⟶ ([x, z], y), ([y, t], z) .. when N=1 (step)

• (biggest − big) + small = smallest (detects semantic relationships)

☆ image captioning = generating textual description of an image (image is an input too)

☆ machine translation = two RNNs (encoder & decoder), different sized input and output

↳ used for language translation and speech recognition

↳ encoder ⟶ summary (s) ⟶ decoder (performance decreases with longer sentences)

↳ attention mechanism: google's neural machine translation (transformers)

☆ pixelRNN = predicts the next pixels in an image, can also generate images from strach

☆ neural turing machines (NTM) = increased memory size (RNN limited due to vanishing gradient)

↳ selective read or write on the memory

↳ every component is differentiable

↳ better than LSTMs in: copy, repeated copy, associated recall, sorting