24) RNN

Sequence Processing

- Data ordered sequentially & neighbors are correlated
- RNN leverages it to ① predict the short-term future ② categorize the past

I love you English
te quiero Spanish

① Time series data: easy to interpret as a numerical table

② Language: encode each letter/word numerically

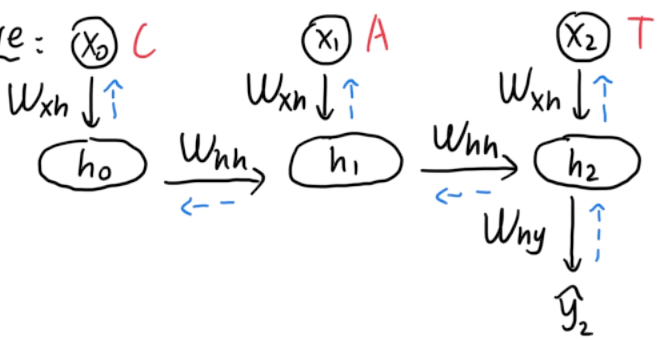
Ex) ord(Hello) ⇒ 8-5-12-12-15, Vocab_index(Entire Words) ⇒ 234-3048

- Non-RNN Limitation: ① Vanilla CNN: fixed input length, ignores context & order
  ② FFN: ignores structures, independent learning params

Vanilla RNN

- Recurrent network designed to sequence processing

- architecture:



RNN layer ← make: input_size = 3, hidden_size = 3, num_layer = 1

Transformed result (>1 translation, =1 categorization)

- Backprop Risks: Due to $\alpha \frac{\partial J}{\partial w}$, a deep RNN is oversensitive to recent inputs, vanishing to distant inputs

- metaparams:

① Input size: # features / data channels (1 for words)
② Sequence length: length of overlapping data segment
③ Batch size: # data segments processed per epoch

translation data: 0 1 2 3 4 5 6

↓train

$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 3 & 4 & 5 \end{bmatrix}$ 4, 6

x_0 y

① = 1
② = 4
③ = 2

More on RNNs

① Hidden State h: activations of a hidden layer $\in \mathbb{R}^{hidden\_size}$ ← memory from previous sequence

- Changes at each timestep t from x_t: $h_t = \tanh(W_{xh} x_t + W_{hh} h_{(t-1)})$
- Initializes with 0s so it's not biased with random memory at $t=0$

② Embedding: represents each letter as N-dim vector with linear weights

· Interpretation: Over training, embedding layer learns structural patterns b/t data

N is arbitrary # and is abstract to interpret

	1	2	3 ← dim index
h	2	0	1
i	4	1	2

· Issue in One-Hot Encoding: makes each encoded letter is orthogonal (did not capture natural correlation in a word)

	a	...	h	i	...	z
h	0	0	1	0	0	0
i	0	0	0	1	0	0

GRU & LSTM

- Gate: a sigmoid output that controls the flow of information when multiply
- Gated Recurrent Unit (GRU): useful for longer patterns but need more data

· Process:

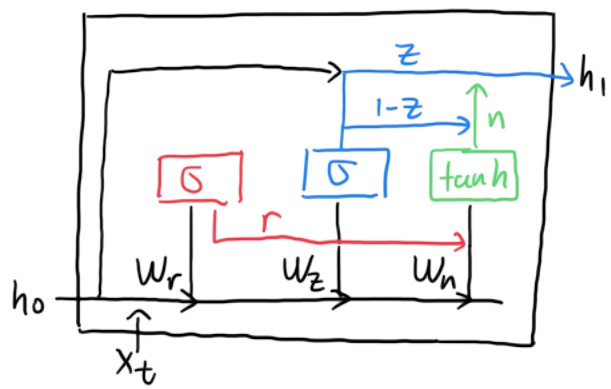
① Reset Gate: $r_t = \sigma(W_r[x_t; h_{t-1}])$
↳ model learns how much of past to update

② Update Gate: $z_t = \sigma(W_z[x_t; h_{t-1}])$
↳ model controls what information to update

③ New information: $n_t = \tanh(W_n[x_t; r_t h_{t-1}])$
↳ model learns a new hidden state candidate

④ Hidden state (output): $h_t = (1-z_t) n_t + z_t h_{t-1}$

architecture (1 hidden unit)



- Long Short-Term Memory (LSTM): useful for memory-heavy & complex tasks

· Process:

① Forget Gate: $f_t = \sigma(W_f[x_t; h_{t-1}])$ knocks out unimportant items

② Input Gate: $i_t = \sigma(W_i[x_t; h_{t-1}])$ filters how much new data to add

③ Cell Gate: $g_t = \tanh(W_i[x_t; h_{t-1}])$ computes actual new data

④ Output Gate: $O_t = \sigma(W_o[x_t; h_{t-1}])$ controls how much info to predict

⑤ Cell State: $C_t = f_t C_{t-1} + i_t g_t$ is what LSTM remembers

⑥ Hidden State: $h_t = O_t \tanh(C_t)$ is what LSTM predicts