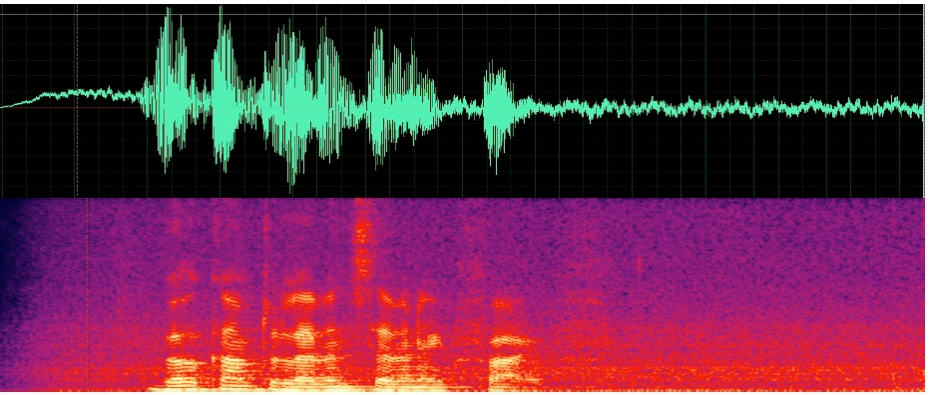
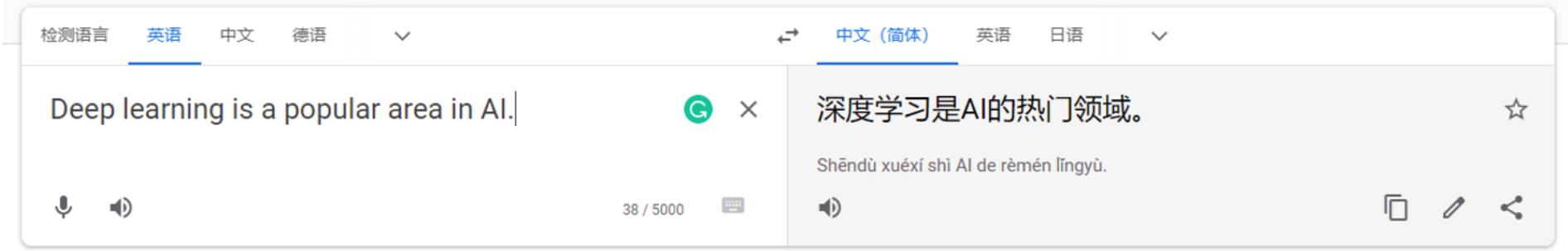
Recurrent Neural Networks

Sequence Data



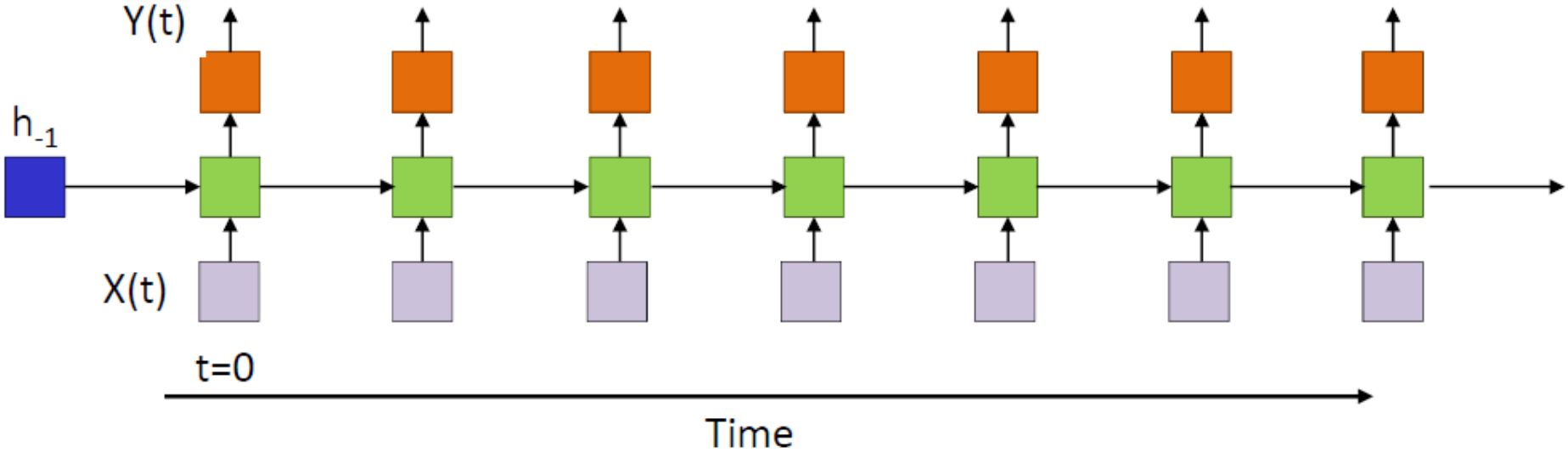


State-Space Model

Hidden Markov Model

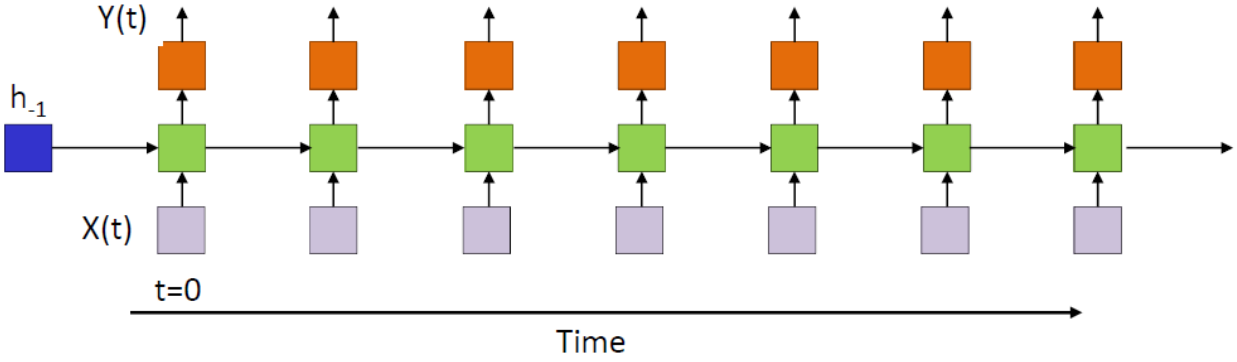
- h_t: hidden state
- X_t: input
- Y_t: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- h_{-1}: initial state

Also stuff reinforcement learners think about!



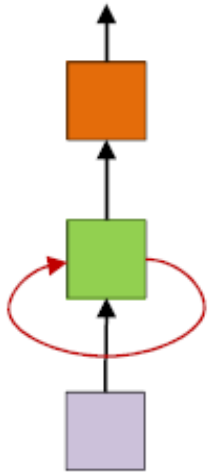
Recurrent Neural Network

- h_t: hidden state
- X_t: input
- Y_t: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- h_{-1}: initial state



Fully-connect NN vs. RNN

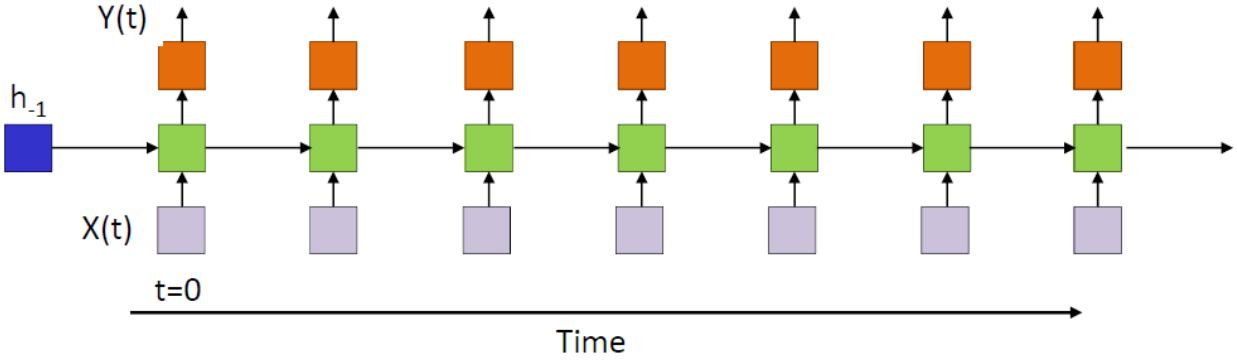
- h_t: a vector summarizes all past inputs (a.k.a. "memory")
- h_{-1} affects the entire dynamics (typically set to zero)
- X_t affects all the outputs and states after t
- Y_t depends on $X_0, \ldots, X_t$



$$h_t \approx \sigma_1\left(W^{(11)}\, h_{t-1}\right)$$
$$\approx \sigma_1\left(W^{(11)}\, \sigma_1\left(W^{(11)}\, h_{t-2}\right)\right) \cdots$$
$$\approx \text{Repeat } \sigma_1\left(W^{(11)}\right) \; t-1 \text{ times } \cdots h_1$$

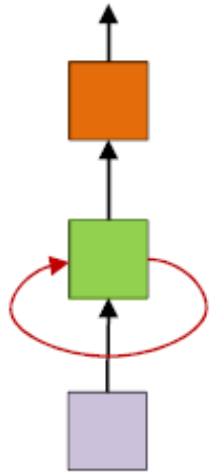
Recurrent Neural Network

- h_t: hidden state
- X_t: input
- Y_t: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- h_{-1}: initial state

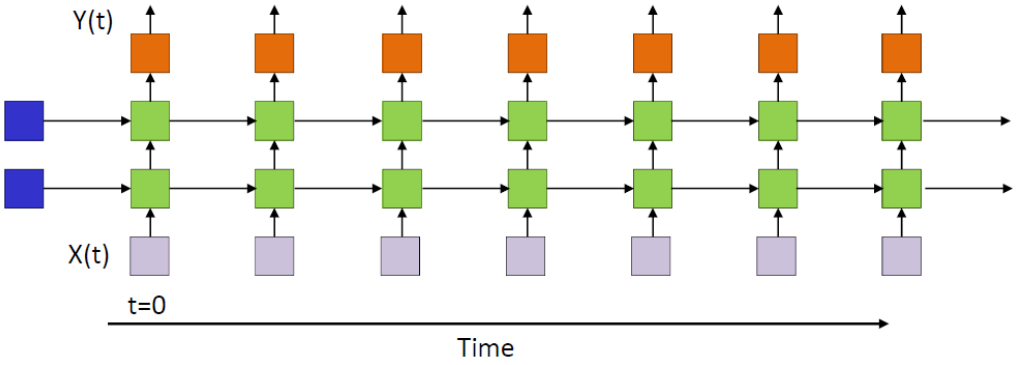


Fully-connect NN vs. RNN

- RNN can be viewed as repeated applying fully-connected NNs
- $h_t = \sigma_1(W^{(1)}X_t + W^{(11)}h_{t-1} + b^{(1)})$
- $Y_t = \sigma_2(W^{(2)}h_t + b^{(2)})$
- σ_1, σ_2 are activation functions (sigmoid, ReLU, tanh, etc)

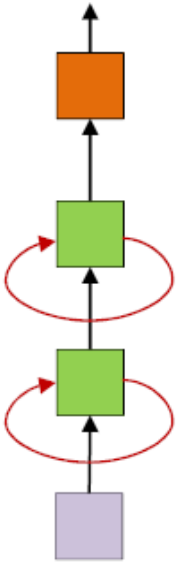


Recurrent Neural Network



Stack *K* layers of fully-connected NN

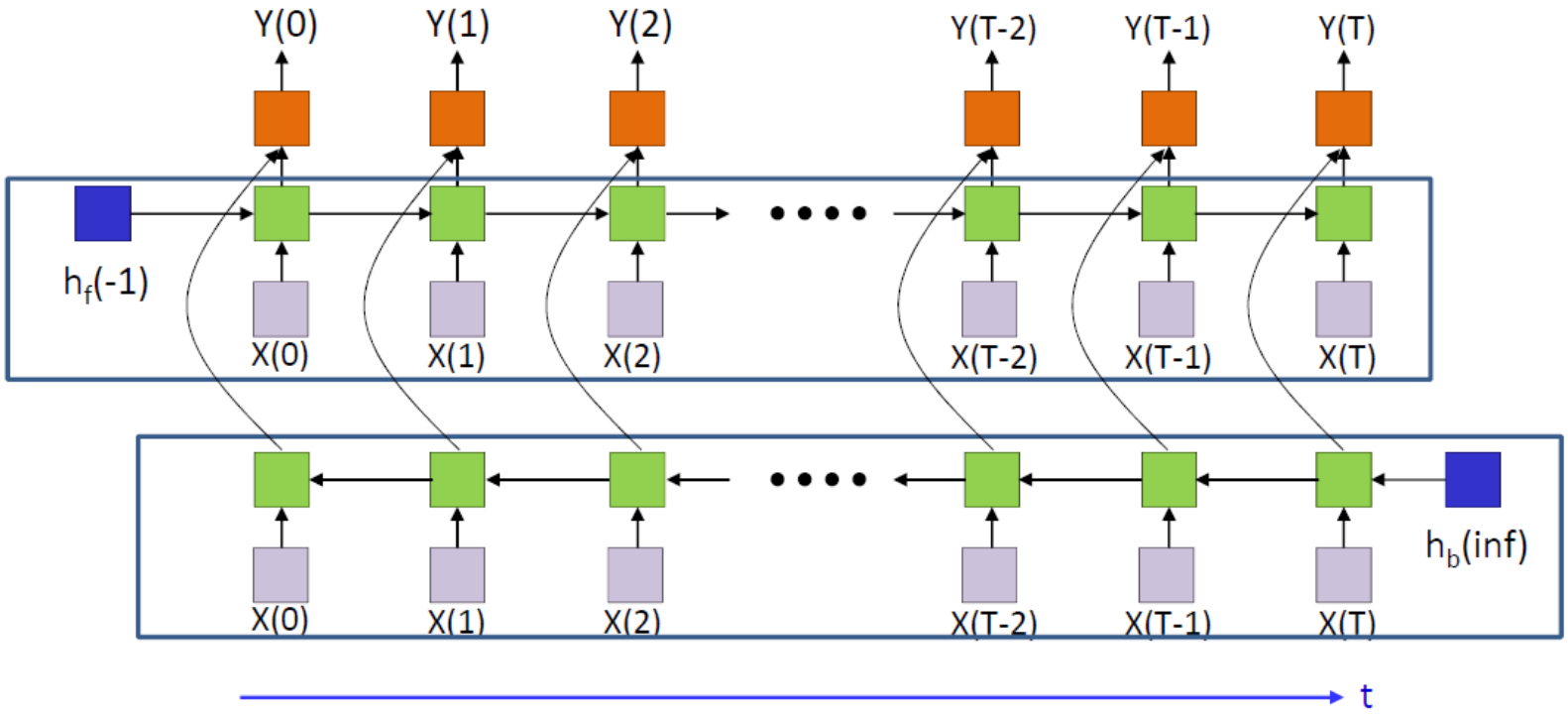
- $h_t^{(k)}$: hidden state
- X_t: input
- Y_t: output
- $h_t^{(1)} = f_1^{(1)}(h_{t-1}^{(1)}, X_t; \theta)$
- $h_t^{(k)} = f_1^{(k)}(h_{t-1}^{(k)}, h_t^{(k-1)}; \theta)$
- $Y_t = f_2(h_t^{(K)}; \theta)$
- $h_{-1}^{(k)}$: initial states



Extensions

What if Y_t depends on the entire inputs?

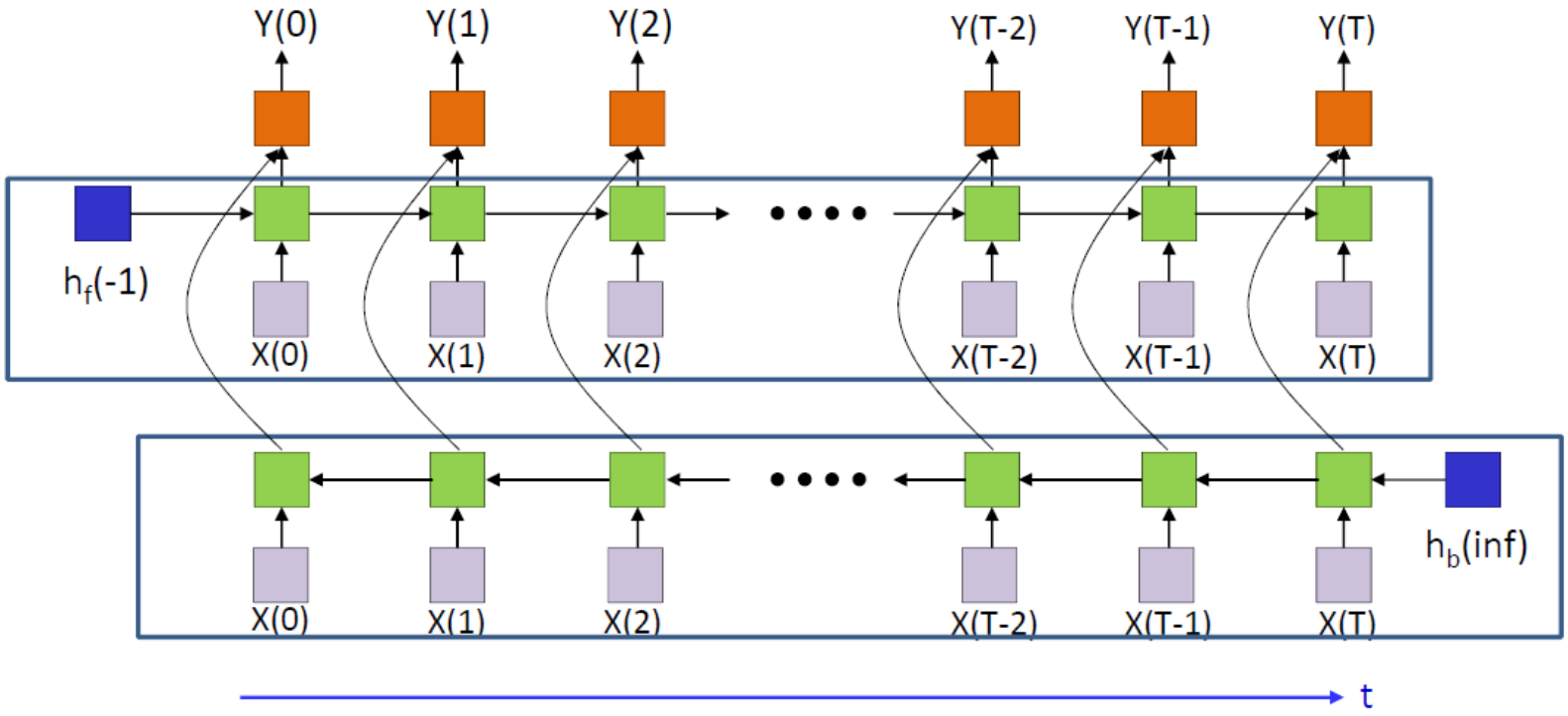
- Biredictional RNN:
  - AN RNN for forward dependencies: t= 0,…,T
  - An RNN for backward dependencies: t= T,…0
  - $Y_t = f_2(h_t^f, h_t^b; \theta)$



Extensions

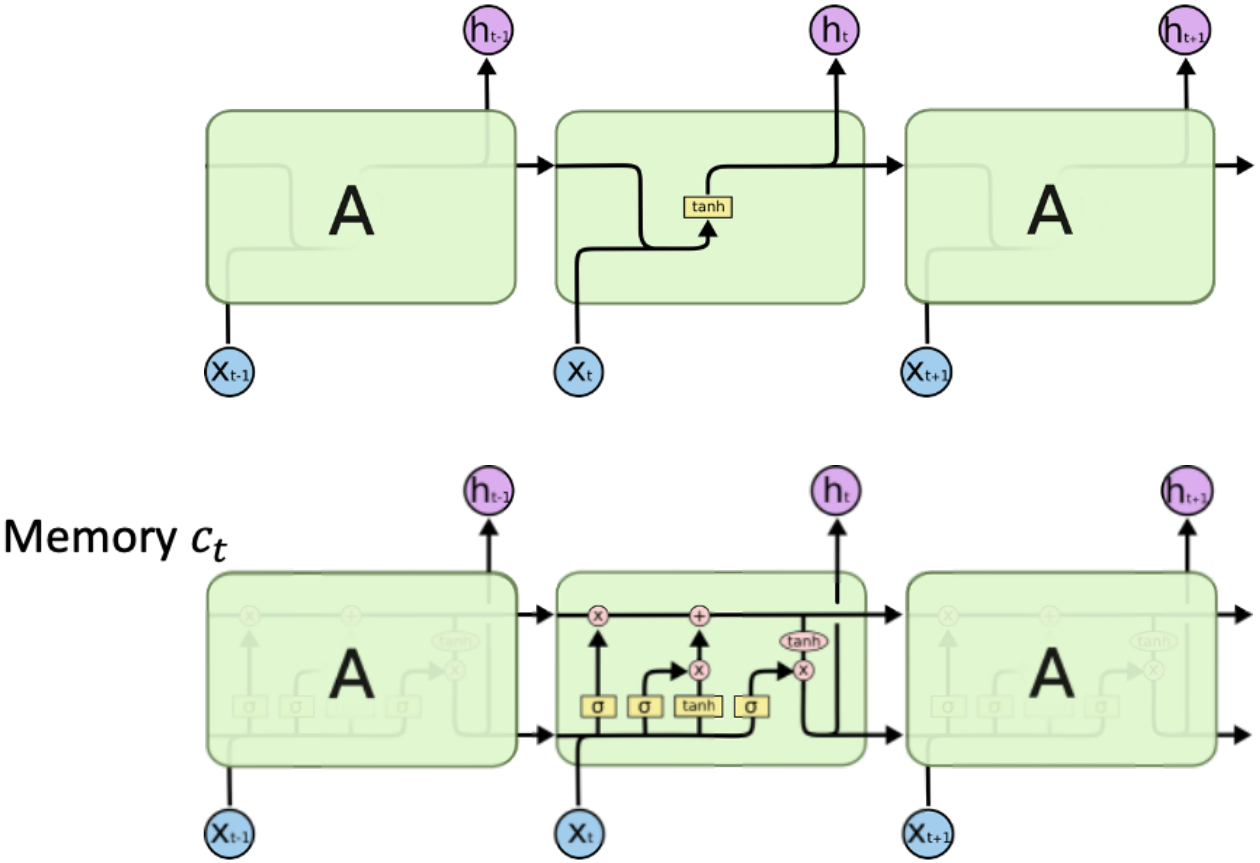
RNN for sequence classification (sentiment analysis)

- $Y = \max_t Y_t$
- Cross-entropy loss



Preserve Long-Term Memory

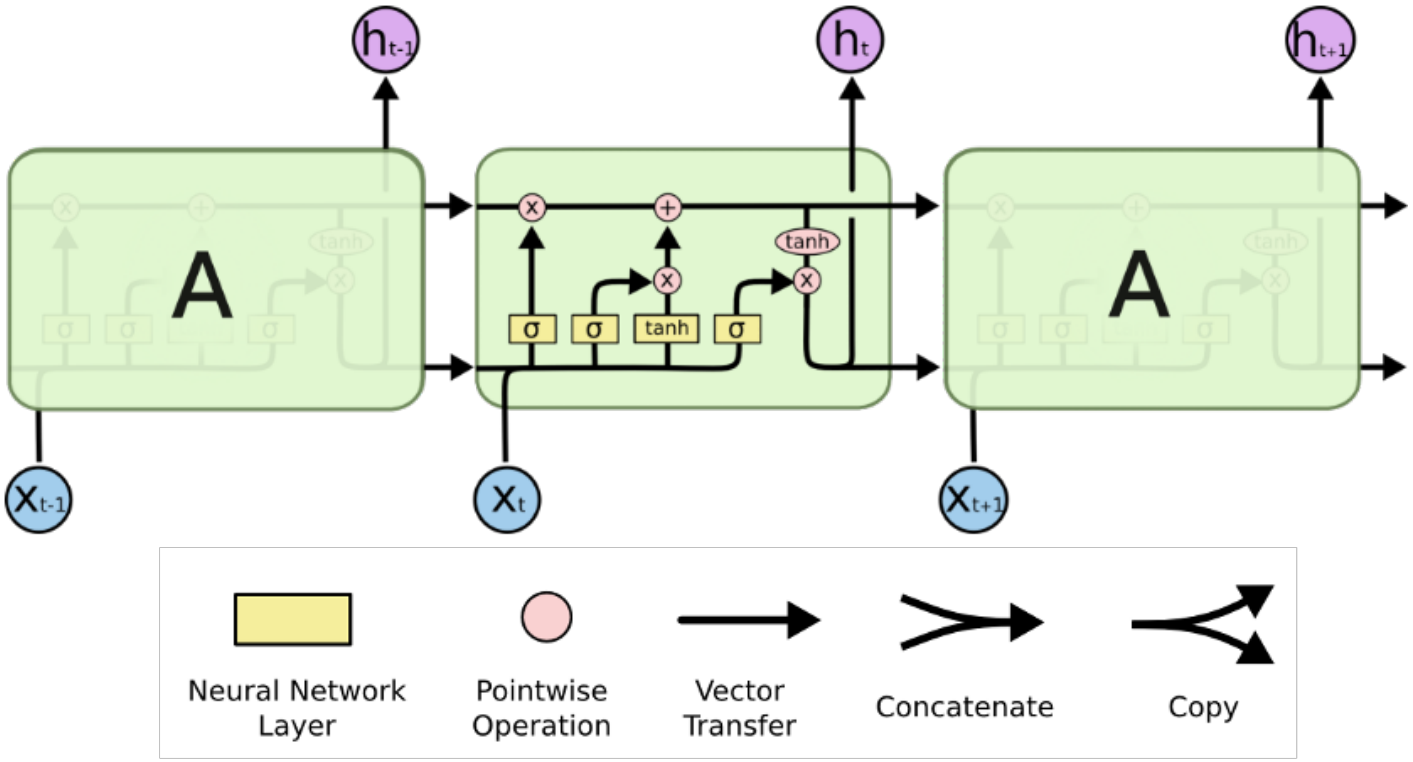
- Difficult for RNN to preserve long-term memory
  - The hidden state h_t is constantly being written (short-term memory)
  - Use a separate cell to maintain long-term memory



Long Short-Term Memory Network

LSTM (Hochreitcher & Schmidhuber, ’97)

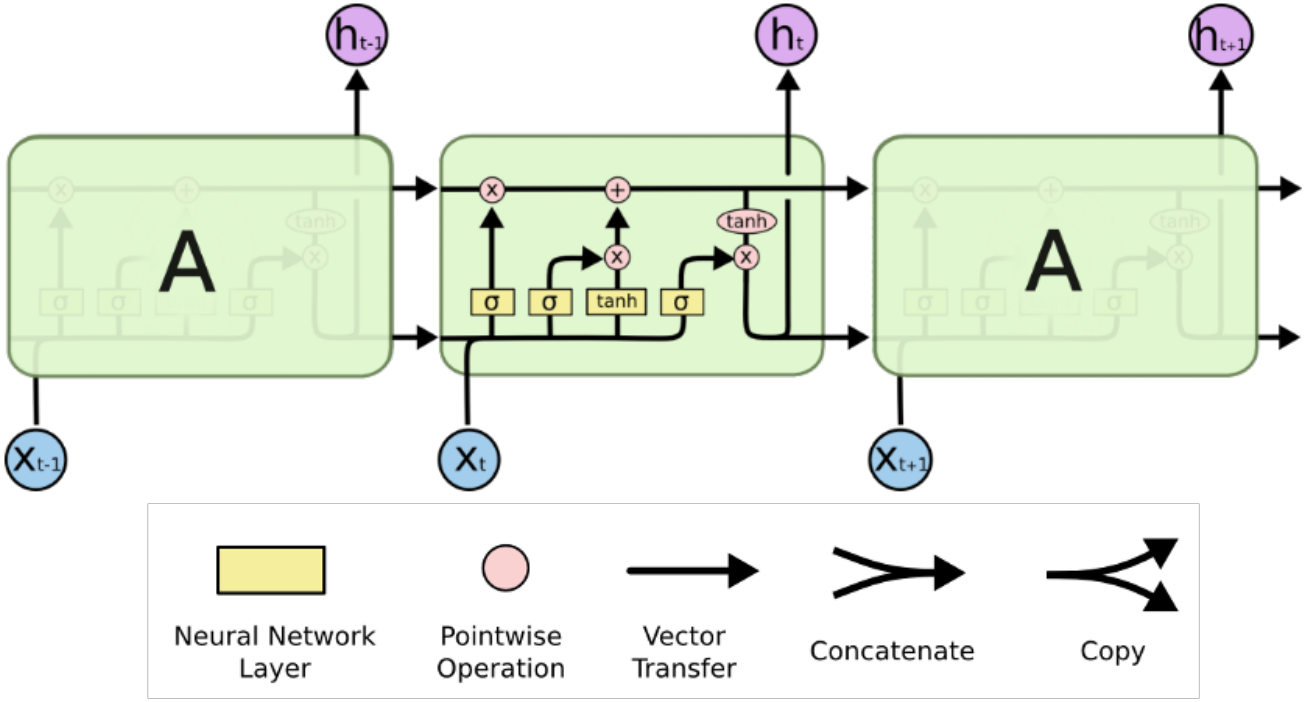
- RNN architecture for learning long-term dependencies
- σ: layer with sigmoid activation



Long Short-Term Memory Network

LSTM (Hochreitcher & Schmidhuber, '97)

- Core idea: maintain separate state h_t and cell c_t (memory)
- h_t: full update every step
- c_t: only *partially* update through gates
  - σ layer outputs importance ($[0,1]$) for each entry and only modify those entries of c_t



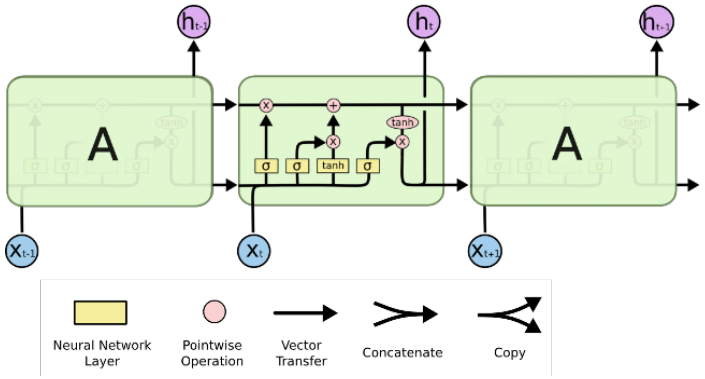
Long Short-Term Memory Network

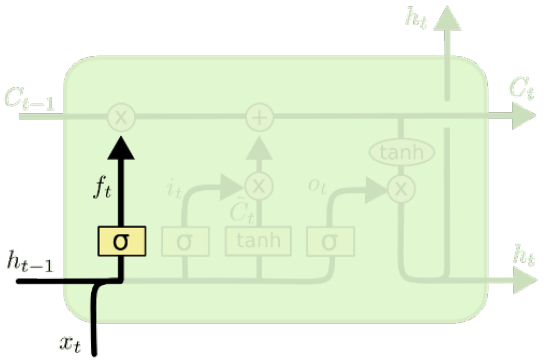
Forget gate $f_t \in \mathbb{R}^d$ # d is the hidden cell dimension

- f_t outputs whether we want to "forget" things in $c_t \in \mathbb{R}^d$
  - Compute $c_{t-1} \odot f_t$ (element-wise)
  - $f_t(i) \to 0$: want to forget $c_t(i)$ $i = 1, \ldots, d$
  - $f_t(i) \to 1$: we want to keep the information in $c_t(i)$

$$c_{t-1} \otimes f_t = \begin{pmatrix} c_{t-1}[1] \cdot f_t[1] \\ \vdots \\ c_{t-1}[d] \cdot f_t[d] \end{pmatrix} \in \mathbb{R}^d$$

pointwise product: $\odot$ or $\otimes$





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

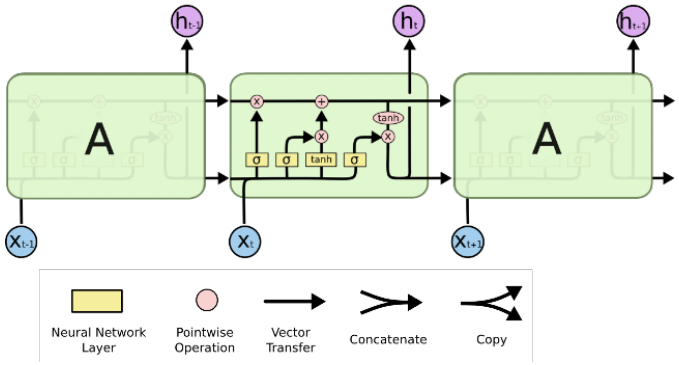
$$\in [0, 1]$$

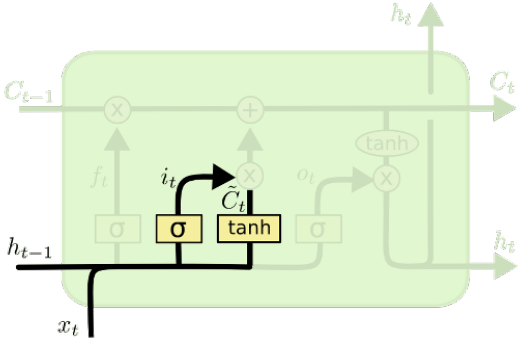
Long Short-Term Memory Network

Input gate i_t

- i_t extracts useful information from X_t to update memory
  - $\tilde{c}_t$: information from X_t to update memory
  - i_t: which dimension in the memory should be updated by X_t
    - $i_t(j) \rightarrow 1$: we want to use the information in $\tilde{c}_t(j)$ to update memory
    - $i_t(t) \rightarrow 0$: $\tilde{c}_t(j)$ should not contribute to memory

$$i_t \otimes \tilde{c}_t$$





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

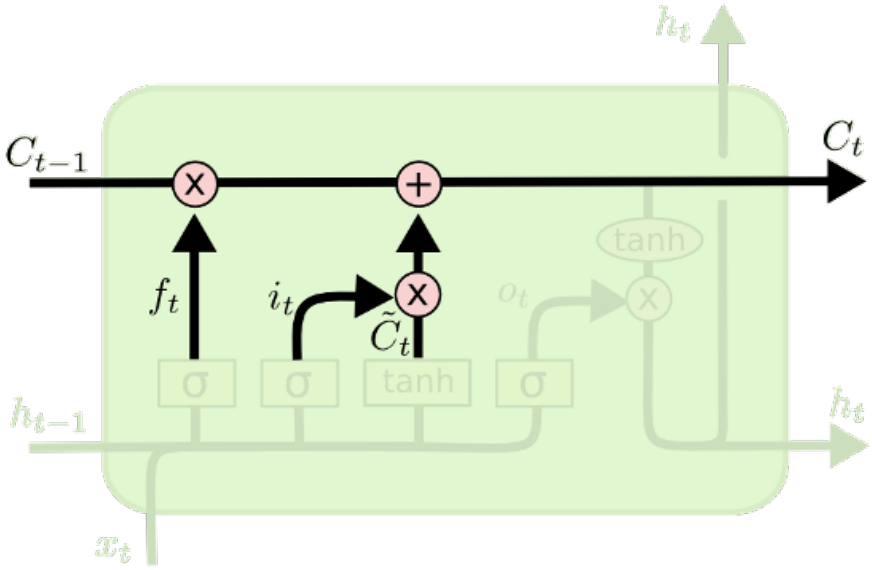
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Sigmoid

Long Short-Term Memory Network

Memory update

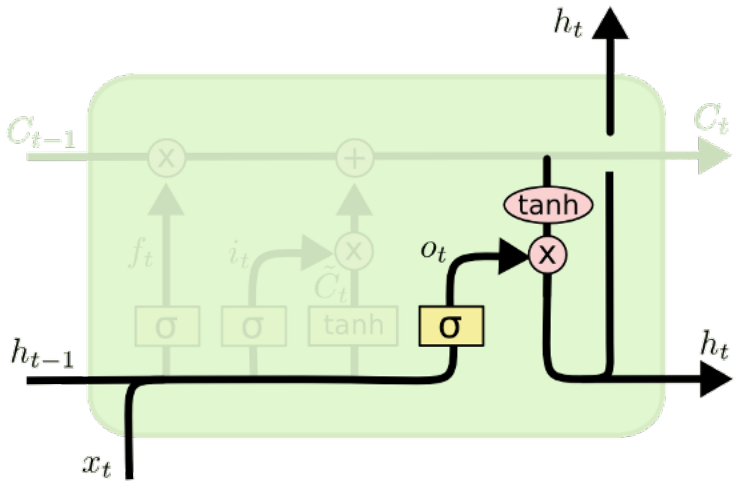
- $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
- f_t forget gate; i_t input date
- $f_t \odot c_{t-1}$: drop useless information in old memory
- $i_t \odot \tilde{c}_t$: add selected new information from current input



Long Short-Term Memory Network

Output gate o_t

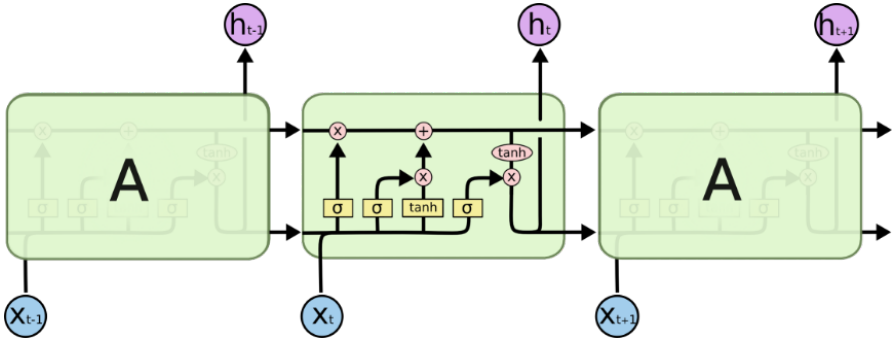
- Next hidden state $h_t = o_t \odot \tanh(c_t)$
  - $\tanh(c_t)$: non-linear transformation over all past information
  - o_t: choose important dimensions for the next state
    - $o_t(j) \rightarrow 1$: $\tanh(c_t(j))$ is important for the next state
    - $o_t(j) \rightarrow 0$: $\tanh(c_t(j))$ is not important



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

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM review



- $\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$ # compute potential memory update based on input, previous hidden state
- $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ # update memory using input and forget gates
- $h_t = o_t \odot \tanh(c_t)$ # squash memory and apply output gate to get next hidden state
- $Y_t = g(h_t)$ # produce final output based on hidden state

Remarks:

1. No more matrix multiplications for c_t
2. LSTM does not have guarantees for gradient explosion/vanishing
3. LSTM is the dominant architecture for sequence modeling from ’13 - ’16. (Often still used in robotics)
4. Why tanh?

    # squishes output to [-1,1]. So why?

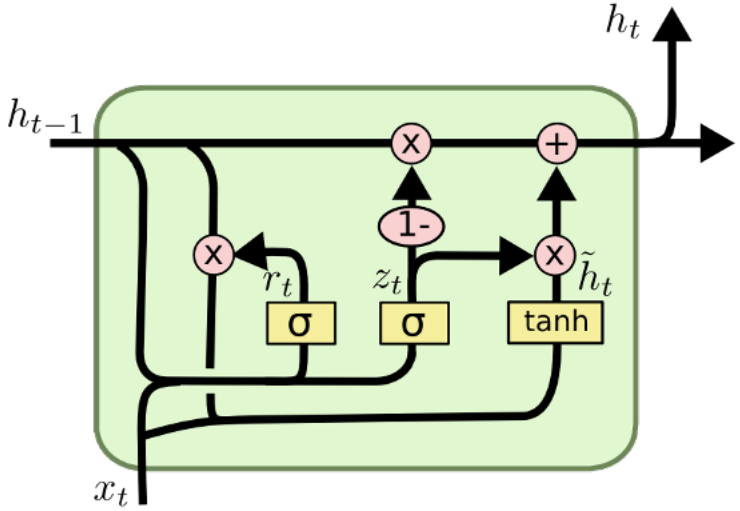
    # Allow memory to be incremented decremented via [-1,1] updates

    # Since memory c_t could continue to grow, squash it to compute hidden state

LSTM Variant

Gated Recurrent Unit (GRU, Cho et al. '14)

- Merge h_t and c_t: much fewer parameters
- Go from input, forget, output gates to just **update and reset**
- In my experience / reading of the literature, it works just as well



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

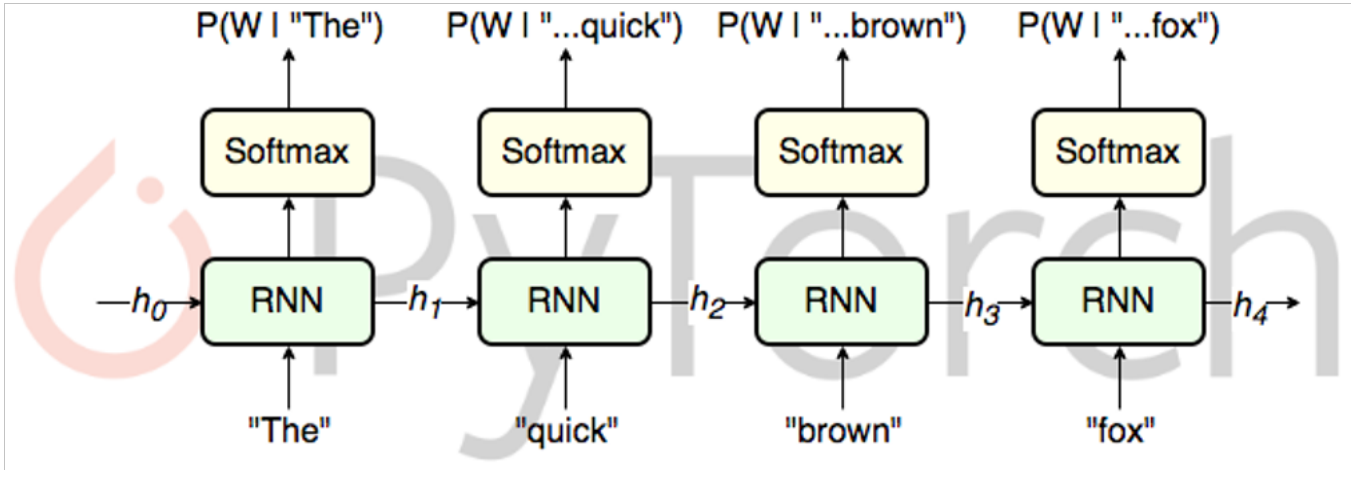
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM application: language model

- Autoregressive language model: $P(X;\theta) = \Pi_{t=1}^{L} P(X_t \mid X_{i<t};\theta)$
  - X: a sentence
  - Sequential generation
- LSTM language model
  - X_t: word at position t.
  - Y_t: softmax over all words
- Data: a collection of texts:
  - Wiki



We got very fancy hierarchical LSTMs before Transformers

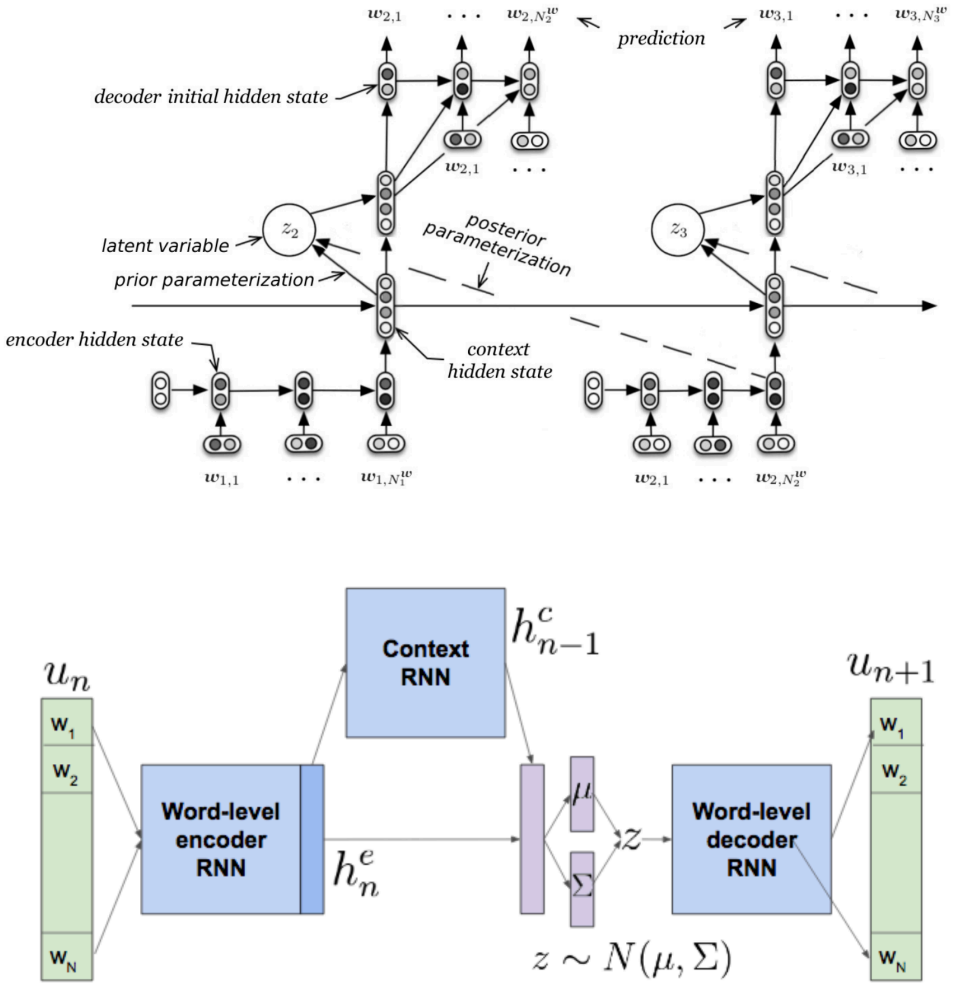


Figure 1: Simplified diagram of the variational hierarchical dialog model.

[1]Serban, I., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A., & Bengio, Y. (2017, February). A hierarchical latent variable encoder-decoder model for generating dialogues. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 31, No. 1).

[2]Jaques, N., Ghandeharioun, A., Shen, J.H., Ferguson, C., Lapedriza, A., Jones, N., Gu, S. and Picard, R., 2019. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*.