Sequence-to-sequence models

Sequence-to-sequence models

- Sequence goes in, sequence comes out
- Sequence-to-sequence models are a powerful learning framework that have found success in a wide range of applications
  - Automatic Speech Recognition (ASR): sound stream goes in, text comes out
  - Machine Translation (MT): source language sentence goes in, target language sentence comes out
  - Image captioning: Image goes in, caption comes out
  - text summarization: whole text goes in, summary comes out
  - Automatic email responses: Generating automatic responses to incoming emails
  - etc. etc.

Translation

Translation

The encoder decoder architecture

- The encoder network converts the source sentence into a vector or a matrix representation; the decoder network then converts the encoding into a sentence in the target language

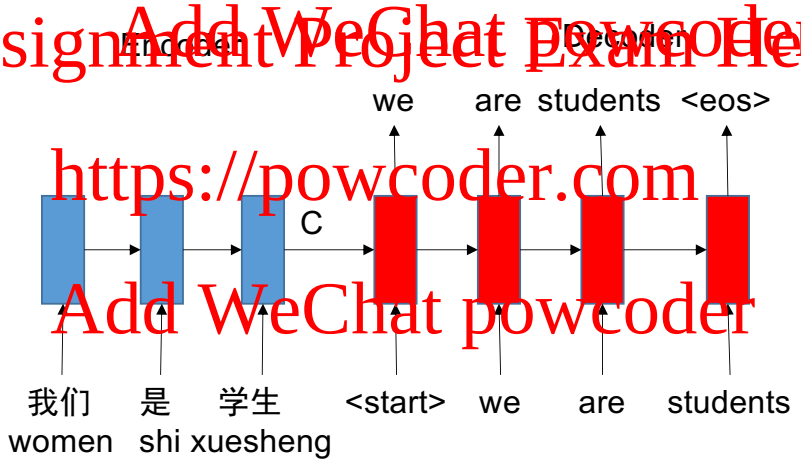
$$\boldsymbol{z} = \textsc{Encode}(\boldsymbol{w}^{(s)})$$

$$\boldsymbol{w}^{(t)} | \boldsymbol{w}^{(s)} \sim \textsc{Decode}(\boldsymbol{z})$$

where the second line means the decoder defines the conditional probability $P(\boldsymbol{w}^{(t)} | \boldsymbol{w}^{(s)})$.

- The decoder can be a recurrent neural network (e.g., LSTM) that generates one word at a time, while recurrently updating a hidden state, or it can be a transformer.
- The encoder decoder networks are trained end-to-end from parallel sentences.

Encoder decoder



Training objective

If the output layer of the decoder is a logistic function, then the entire network can be trained to maximize the conditional log-likelihood (or minimize the negative log-likelihood):

$$\log P(\boldsymbol{w}^{(t)} | \boldsymbol{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} P(w_m^{(t)} | \boldsymbol{w}_{1:m-1}^{(t)}, \boldsymbol{z})$$

$$w_m^{(t)} | \boldsymbol{w}_{1:m-1}^{(t)}, \boldsymbol{w}^{(s)} \sim \text{SoftMax}\left(\boldsymbol{\beta} \cdot \boldsymbol{h}_{m-1}^{(t)}\right)$$

where $\boldsymbol{h}_{m-1}^{(t)}$ is a recurrent function of the previously generated text $\boldsymbol{w}_{1:m-1}^{(t)}$ and the ecoding $\boldsymbol{z}$, and $\boldsymbol{\beta} \in \mathbb{R}^{(V^{(t)} \times K)}$ is the matrix of output word vectors for the $V^{(t)}$ words in the target language vocabulary

The LSTM variant

- In the LSTM variant of the encoder network, the encoder is set to the final hidden state of an LSTM on the source sentence:

$$\boldsymbol{h}_m^{(s)} = \text{LSTM}(\boldsymbol{x}_m^{(s)}, \boldsymbol{h}_{m-1}^{(s)})$$

$$\boldsymbol{z} \triangleq \boldsymbol{h}_{M^{(s)}}^{(s)}$$

where $\boldsymbol{x}^{(s)}$ is the embedding of the source language word $w_m^{(s)}$.

- The encoding then provides the initial hidden state for the decoder LSTM:

$$\boldsymbol{h}_0^{(t)} = \boldsymbol{z}$$

$$\boldsymbol{h}_m^{(t)} = \text{LSTM}(\boldsymbol{x}_m^{(t)}, \boldsymbol{h}_{m-1}^{(t)})$$

where $\boldsymbol{x}_m^{(t)}$ is the embedding of the target language word $w_m^{(t)}$

Tweaking the encoder decoder network

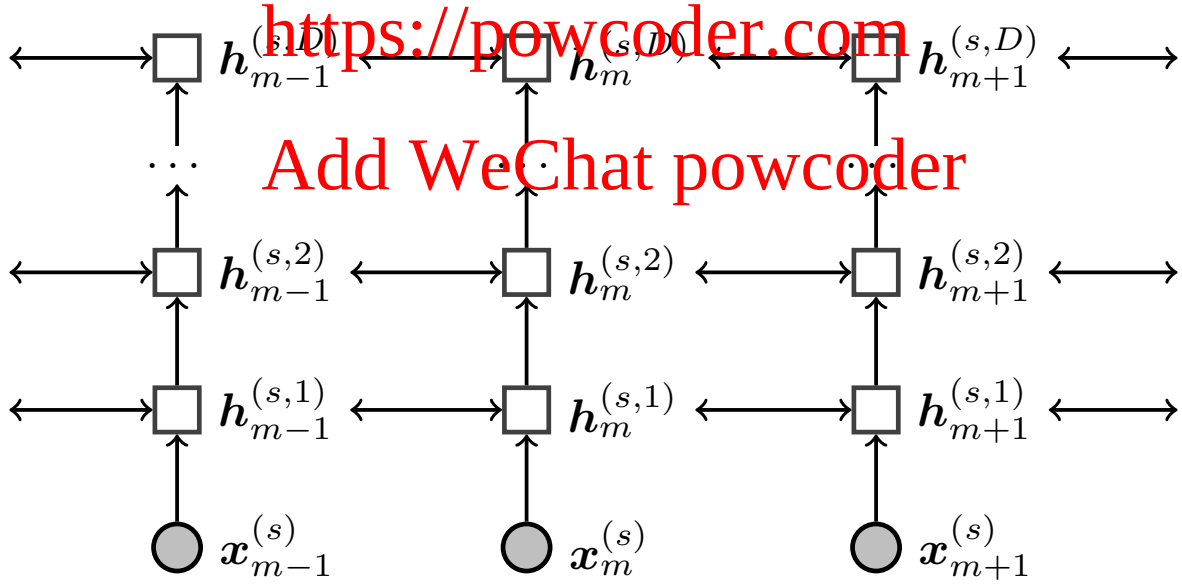
- Adding layers: The encoder and decoder networks can be implemented as **deep LSTMs** with multiple layers of hidden state
- Adding **attention** to give more weight to particular word or words in the source language when generating a word in the target language

Multi-layered LSTMs

Each hidden state $\boldsymbol{h}^{(s,i)}$ at layer i is treated as the input to an LSTM at layer $i+1$:

$$\boldsymbol{h}_m^{(s,1)} = \text{LSTM}(\boldsymbol{x}_m^{(s)}, \boldsymbol{h}_{m-1}^{(s)})$$

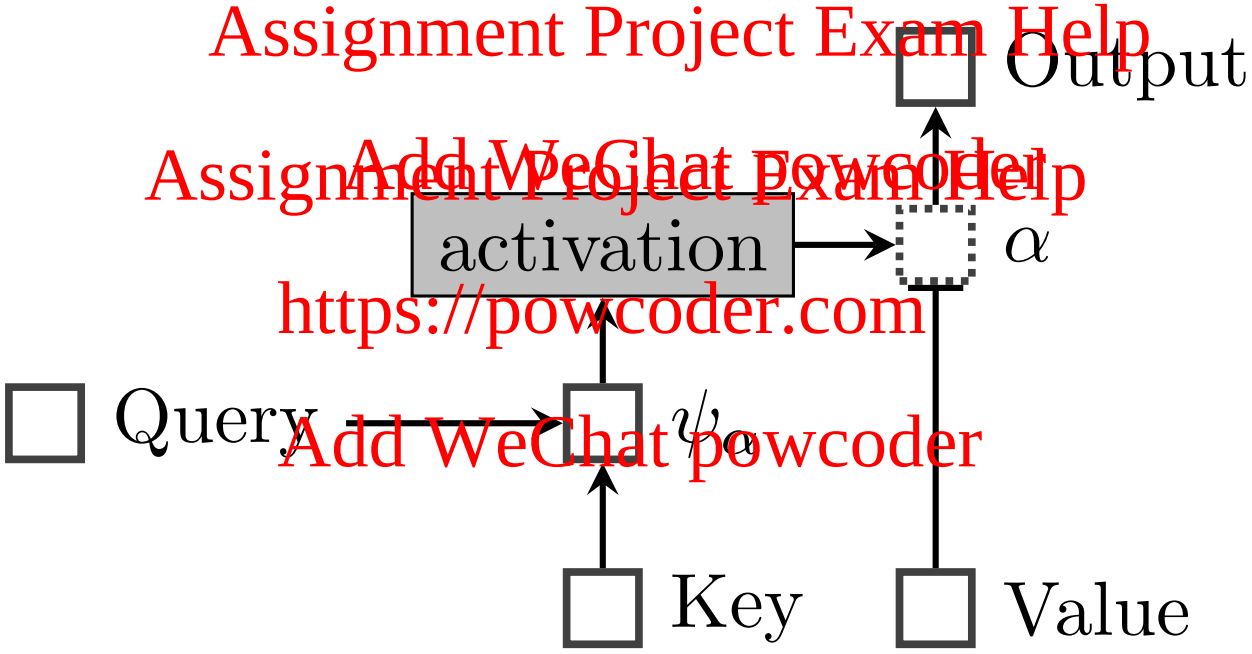
$$\boldsymbol{h}_m^{(s,i+1)} = \text{LSTM}(\boldsymbol{h}_m^{(s,i)}, \boldsymbol{h}_{m-1}^{(s,i+1)}), \forall i \geq 1$$



Neural attention

- Attention can be thought of as using a query to select from a memory of key-value pairs, with the keys, values and queries all being vectors
- For each key n in the memory, we compute a score with respect to the query m, which measures the "comptability" between the key and the query
- The scores are passed through an activation function, typically softmax, which results in a vector of non-negative numbers of length N, which equal to the size of the memory: $[\alpha_{m\to 1}, \alpha_{m\to 2}, \cdots, \alpha_{m\to N}]$
- Multiply each value in the memory v_n by the attention $\alpha_{m\to n}$, and sum them up, we get the output of the attention.
- The attention is typical concatenated with the decoding hidden state to output the target word

"Querying"



Step by step computation of attention

- Computing compatibility with a two layer feedforward network:

$$\psi_\alpha(m, n) = \boldsymbol{v}_\alpha \cdot \tanh(\boldsymbol{\Theta}_\alpha[\boldsymbol{h}_m^{(t)}; \boldsymbol{h}_n^{(s)}])$$

- Softmax attention

$$\alpha_{m\to n} = \frac{\exp \psi_\alpha(m, n)}{\sum_{n'=1}^{M^{(s)}} \exp \psi_\alpha(m, n')}$$

- Compute the weighted average over columns of $\boldsymbol{Z}$

$$\boldsymbol{c}_m = \sum_{n=1}^{M^{(s)}} \alpha_{m\to n} \boldsymbol{z}_n$$

- incorporate the context vector into the decoding model:

$$\tilde{\boldsymbol{h}}_m^{(t)} = \tanh\left(\Theta_c[\boldsymbol{h}_m^{(t)}, \boldsymbol{c}_m]\right)$$

$$P(w_{m+1}^{(t)} | \boldsymbol{w}_{1:m}^{(t)}, \boldsymbol{w}^{(s)} \propto \exp\left(\beta_{w_{m+1}^{(t)}} \cdot \tilde{\boldsymbol{h}}_m^{(t)}\right)$$

Seq2seq: Initialization

- Word embeddings

$$\boldsymbol{E} = embed\left(\begin{bmatrix} women \\ shi \\ xuesheng \\ START \\ we \\ are \\ students \\ EOS \end{bmatrix}\right) = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \\ -0.5 & 0.5 \\ 0 & 0 \\ 0.1 & 0.2 \\ 0.2 & 0.1 \\ 0.1 & 0.2 \\ 1 & 1 \end{bmatrix}$$

- RNN parameters

$$\boldsymbol{W}^{(s)} = \begin{bmatrix} 0.3 & 0 \\ 0 & 0.3 \end{bmatrix} \boldsymbol{U}^{(s)} = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} \boldsymbol{b}^{(s)} = \begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix}$$

$$\boldsymbol{W}^{(t)} = \begin{bmatrix} 0.3 & 0 \\ 0 & 0.3 \end{bmatrix} \boldsymbol{U}^{(t)} = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} \boldsymbol{b}^{(t)} = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix}$$

Encoder

$$\boldsymbol{z} \triangleq \boldsymbol{h}^{(s)}$$

$$\boldsymbol{h}_n^{(s)} = \tanh(\boldsymbol{W}^{(s)} \times \boldsymbol{x} + \boldsymbol{U}^{(s)} \times \boldsymbol{h} + \boldsymbol{b})$$

$$\boldsymbol{h}_1^{(s)} = \tanh(\boldsymbol{W}^{(s)} \times \boldsymbol{E}[women] + \boldsymbol{U}^{(s)} \times 0 + \boldsymbol{b}^{(s)}) = \begin{bmatrix} 0.3364 \\ 0.5717 \end{bmatrix}$$

$$\boldsymbol{h}_2^{(s)} = \tanh(\boldsymbol{W}^{(s)} \times \boldsymbol{E}[shi] + \boldsymbol{U}^{(s)} \times \boldsymbol{h}_1^{(s)} + \boldsymbol{b}^{(s)}) = \begin{bmatrix} 0.3153 \\ 0.7289 \end{bmatrix}$$

$$\boldsymbol{h}_3^{(s)} = \tanh(\boldsymbol{W}^{(s)} \times \boldsymbol{E}[xuesheng] + \boldsymbol{U}^{(s)} \times \boldsymbol{h}_2^{(s)} + \boldsymbol{b}^{(s)}) = \begin{bmatrix} 0.0086 \\ 0.5432 \end{bmatrix}$$

$$C = \boldsymbol{h}_3^{(s)} = \begin{bmatrix} 0.0086 \\ 0.5432 \end{bmatrix}$$

where C is a context vector

Decoder

$$\boldsymbol{h}_m^{(t)} = \tanh(\boldsymbol{W}^{(t)} \times \boldsymbol{x} + \boldsymbol{U}^{(t)} \times \boldsymbol{h} + \boldsymbol{b})$$

$$\boldsymbol{h}_1^{(t)} = \tanh(\boldsymbol{W}^{(t)} \times \boldsymbol{E}[\textit{START}] + \boldsymbol{U}^{(t)} \times \boldsymbol{h} + \boldsymbol{b}^{(t)}) = \begin{bmatrix} 0.6929 \\ 0.2482 \end{bmatrix}$$

$$\boldsymbol{h}_2^{(t)} = \tanh(\boldsymbol{W}^{(t)} \times \boldsymbol{E}[\textit{we}] + \boldsymbol{U}^{(t)} \times \boldsymbol{h}_1^{(t)} + \boldsymbol{b}^{(t)}) = \begin{bmatrix} 0.6203 \\ 0.2123 \end{bmatrix}$$

$$\boldsymbol{h}_3^{(t)} = \tanh(\boldsymbol{W}^{(t)} \times \boldsymbol{E}[\textit{are}] + \boldsymbol{U}^{(t)} \times \boldsymbol{h}_2^{(t)} + \boldsymbol{b}^{(t)}) = \begin{bmatrix} 0.6039 \\ 0.1870 \end{bmatrix}$$

$$\boldsymbol{h}_4^{(t)} = \tanh(\boldsymbol{W}^{(t)} \times \boldsymbol{E}[\textit{students}] + \boldsymbol{U}^{(t)} \times \boldsymbol{h}_3^{(t)} + \boldsymbol{b}^{(t)}) = \begin{bmatrix} 0.6220 \\ 0.0980 \end{bmatrix}$$

Softmax over similarities between hidden layers and target embeddings

$$score_1\left(\begin{bmatrix} we \\ are \\ students \\ EOS \end{bmatrix}\right) = softmax\left(\begin{bmatrix} \boldsymbol{h}_1^{(t)} \times \boldsymbol{E}[we] \\ \boldsymbol{h}_1^{(t)} \times \boldsymbol{E}[are] \\ \boldsymbol{h}_1^{(t)} \times \boldsymbol{E}[students] \\ \boldsymbol{h}_1^{(t)} \times \boldsymbol{E}[EOS] \end{bmatrix}\right) = \begin{bmatrix} 0.1913 \\ 0.2000 \\ 0.1733 \\ 0.4354 \end{bmatrix}$$

$$score_2\left(\begin{bmatrix} we \\ are \\ students \\ EOS \end{bmatrix}\right) = softmax\left(\begin{bmatrix} \boldsymbol{h}_2^{(t)} \times \boldsymbol{E}[we] \\ \boldsymbol{h}_2^{(t)} \times \boldsymbol{E}[are] \\ \boldsymbol{h}_2^{(t)} \times \boldsymbol{E}[students] \\ \boldsymbol{h}_2^{(t)} \times \boldsymbol{E}[EOS] \end{bmatrix}\right) = \begin{bmatrix} 0.1999 \\ 0.2070 \\ 0.1826 \\ 0.4116 \end{bmatrix}$$

$$score_3\left(\begin{bmatrix} we \\ are \\ students \\ EOS \end{bmatrix}\right) = softmax\left(\begin{bmatrix} \boldsymbol{h}_3^{(t)} \times \boldsymbol{E}[we] \\ \boldsymbol{h}_3^{(t)} \times \boldsymbol{E}[are] \\ \boldsymbol{h}_3^{(t)} \times \boldsymbol{E}[students] \\ \boldsymbol{h}_3^{(t)} \times \boldsymbol{E}[EOS] \end{bmatrix}\right) = \begin{bmatrix} 0.2012 \\ 0.2098 \\ 0.1867 \\ 0.4023 \end{bmatrix}$$

$$score_4\left(\begin{bmatrix} we \\ are \\ students \\ EOS \end{bmatrix}\right) = softmax\left(\begin{bmatrix} \boldsymbol{h}_4^{(t)} \times \boldsymbol{E}[we] \\ \boldsymbol{h}_4^{(t)} \times \boldsymbol{E}[are] \\ \boldsymbol{h}_4^{(t)} \times \boldsymbol{E}[students] \\ \boldsymbol{h}_4^{(t)} \times \boldsymbol{E}[EOS] \end{bmatrix}\right) = \begin{bmatrix} 0.2037 \\ 0.2147 \\ 0.1959 \\ 0.3857 \end{bmatrix}$$

$$P(Y|X) = score_1 \times score_2 \times score_3 \times score_4$$

Attention

- The idea: Different context vector C_is are used when generating target words at different time steps, e.g.,

$$C_1 = 0.98 \times \boldsymbol{h}_1^{(s)} + 0.01 \times \boldsymbol{h}_2^{(s)} + 0.01 \times \boldsymbol{h}_3^{(s)}$$

$$C_2 = 0.01 \times \boldsymbol{h}_1^{(s)} + 0.98 \times \boldsymbol{h}_2^{(s)} + 0.01 \times \boldsymbol{h}_3^{(s)}$$

$$C_3 = 0.01 \times \boldsymbol{h}_1^{(s)} + 0.01 \times \boldsymbol{h}_2^{(s)} + 0.98 \times \boldsymbol{h}_3^{(s)}$$

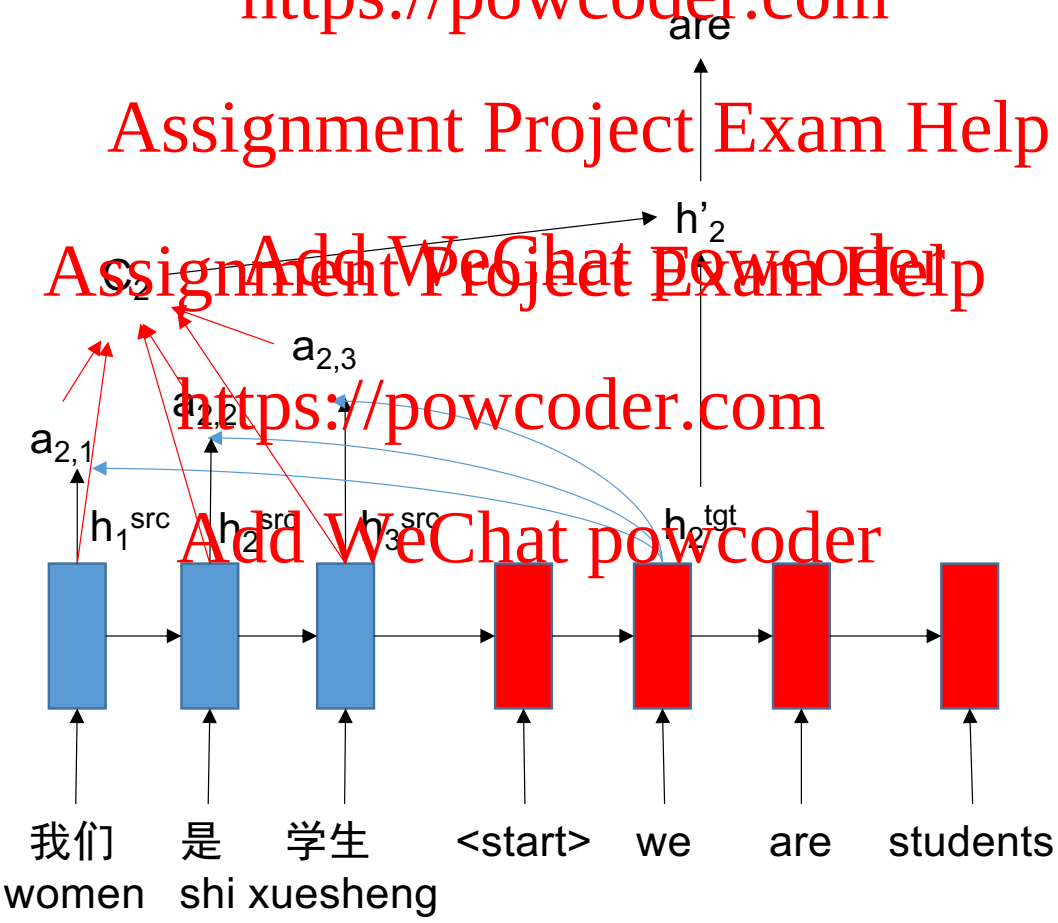
$$C_m = \sum_n \alpha_{m \to n} \times \boldsymbol{h}_n^{(s)}$$

$$\alpha_{m \to n} = \frac{\exp(score(\boldsymbol{h}_m^{(t)}, \boldsymbol{h}_n^{(s)}))}{\sum_{n'=1} \exp(score(\boldsymbol{h}_m^{(t)}, \boldsymbol{h}_{n'}^{(s)}))}$$

$$score(\boldsymbol{h}_m^{(t)}, \boldsymbol{h}_n^{(s)}) = \boldsymbol{h}_m^{(t)\top} \boldsymbol{h}_n^{(s)}$$

- Other scoring variants exist

Computing attention



Other attention variants

- Additive attention:

$$\psi_\alpha(m,n) = \boldsymbol{v}_\alpha \cdot \tanh(\boldsymbol{\Theta}_\alpha[\boldsymbol{h}_m^{(t)} + \boldsymbol{h}_n^{(s)}])$$

- Multiplicative attention

$$\psi_\alpha(m,n) = {\boldsymbol{h}_m^{(t)}}^\top \boldsymbol{\Theta}_\alpha \boldsymbol{h}_n^{(s)}$$

Drawbacks of RNNs

- For RNNs, input is processed as a sequence. The computation of each state ($\boldsymbol{h}_i$) depends on the previous state $\boldsymbol{h}_{i-1}$.
- This prevents parallel computation for all tokens in the input sequence simultaneously, making it difficult to take full advantage of modern computation architectures to increase speed.
- We can imagine a network in which each token in the sequence interacts with any other token in the sequence. Conceptually, this can be viewed as a fully-connected graph where each token is a node in the graph, and the computation of its hidden state depends on all other tokens in the graph.
- With this approach, the computation of the hidden state of a hidden state $\boldsymbol{h}_i$ does not depends on the computation of another hidden state. It only depends on the input sequence.
- With this approach, the order information would have to be captured separately, with position encoding.