Sequence modeling in language

Language modeling

- Language modeling is the task of assigning a probability to a sequence (for evaluating translation candidates).
- It is also the task of assigning probability of a word to follow a sequence of words (for language generation).
- Perfect performance in language modeling means being able to predict the next word in the sentence with a number of guesses that is fewer or less than that required by a human participant.

Language Modeling

Formally, the language model can be formulated as:

$$P(w_{1:M}) = P(w_1)P(w_2|w_1)P(w_3|\boldsymbol{w}_{1:2})P(w_4|\boldsymbol{w}_{1:3}) \cdots P(w_M|\boldsymbol{w}_{1:M-1})$$

With Markov assumption:

$$P(\boldsymbol{w}_{1:M}) \approx \prod_{m=1}^{M} P(w_m|\boldsymbol{w}_{m-n+1:m-1})$$

Text generated by GPT-2

The fact that I work at the university makes me really nervous about my ability to find more people. That's one of the things that we're doing. My university is a very small community. Those are things that are going to have the effect, that go all these years and have the impact that it has on my life."That's one of the things that makes me really nervous about being a student. If you're a computer programmer, there's almost always somebody who you want to study. This is not how I work. Because I'm learning, I have to be very focused. If I am going to get a good job, if I want to get a career, I don't have to go out into the kitchen, you know. Now, you know, I think if I don't get the job, then I just don't have the time. Maybe I feel like I can't do much, and all of these things are going to be my own business and I want to be able to get a job. I'd certainly be looking at that. I'm at a different time. I'm going to write about my business, my business, my business and I want to talk to you about the life that I've been working for the last 30 years or so. So that's why I've been able to be here for the last 15 years through the university. My life has been extremely productive and I've had a lot of great things, so I love it."

Try it yourself: https://gpt2.ai-demo.xyz

Noisy Channel Model for MT

- The Noisy channel model is the general framework for Statistical Machine Translation, with many variants
- Language model is a crucial component of the SMT model that is used to "select" the best translation:

$$
\begin{aligned}
P_{e|s}(\boldsymbol{w}^{(e)}|\boldsymbol{w}^{(s)}) &\propto P_{s,e}(\boldsymbol{w}^{(e)}, \boldsymbol{w}^{(s)}) \\
&= P_{s|e}(\boldsymbol{w}^{(s)}|\boldsymbol{w}^{(e)}) \times P_e(\boldsymbol{w}^{(e)})
\end{aligned}
$$

where $\boldsymbol{w}^{(e)}$ is generated from a language model, $\boldsymbol{w}^{(s)}$ is a Spanish sentence generated from a translation model $P_{s|e}(\boldsymbol{w}^{(s)}|\boldsymbol{w}^{(e)})$

Perplexity: a metric for evaluating language models

- Given a text corpus of M words (where M could be in the millions) $w_1, w_2, w_3, \cdots w_M$, a language model function LM assigns a probability to a word based on its history.

$$\ell(\boldsymbol{w}) = \sum_{m=1}^{M} \log_2 P(w_m | w_{m-1}, w_{m-2}, \cdots, w_1)$$

- The perplexity of the LM with respect to the corpus is:

$$Perplex(\boldsymbol{w}) = 2^{-\frac{\ell(\boldsymbol{w})}{M}}$$

What counts as a good language model?

- Good language models will assign high probability to the events in the corpus, resulting in low perplexity.
- Perplexities are corpus-specific, and perplexities for two language models are only comparable with respect to the same evaluation corpus.

Extreme Cases of Perplexity

- In the limit of a perfect language model, probability 1 is assigned to the held-out corpus, with $Perplex(\boldsymbol{w}) = 2^{-\frac{1}{M}\log_2 1} = 2^0 = 1$

- In the opposite limit, probability zero is assigned to the held-out corpus, which responds to an infinite perplexity: $Perplex(\boldsymbol{w}) = 2^{-\frac{1}{M}\log_2 0} = 2^{\infty} = \infty$

- Assume a uniform, unigram model in which $p(w_i) = 1/V$ for all words in the vocabulary. Then,

$$\log_2(\boldsymbol{w}) = \sum_{m=1}^{M} \log_2 \frac{1}{V} = -\sum_{m=1}^{M} \log_2 V = -M \log_2 V$$

$$Perplex(\boldsymbol{w}) = 2^{\frac{1}{M} M \log_2 V} = 2^{\log_2 V} = V$$

Traditional approaches to language modeling

- Based on a n-order markov property $P(w_{m+1}|\boldsymbol{w}_{1:m}) \approx P(w_{m+1}|\boldsymbol{w}_{m-n:m})$

- The estimates are usually derived from corpus counts

- The role of the language model is to provide good estimates of $\hat{P}(w_{m+1}|\boldsymbol{w}_{m-n:m})$

- The maximum likelihood (MLE) estimate of $\hat{P}(w_{m+1}|\boldsymbol{w}_{m-n:m})$ is then:

$$\hat{P}_{MLE}(w_{m+1}|\boldsymbol{w}_{m-n:m}) = \frac{\#(\boldsymbol{w}_{m-n:m+1})}{\#(\boldsymbol{w}_{m-n:m})}$$

Addressing the zero count problem

- Zero count for any of the n-grams resulting in zero probability for the entire corpus, meaning infinite perplexity!

- Add-α smoothing:

$$\hat{p}_{add-\alpha}(w_{m+1}|\boldsymbol{w}_{m-n:m}) = \frac{\#(\boldsymbol{w}_{m-n:m+1}) + \alpha}{\#(\boldsymbol{w}_{m-n:m}) + \alpha|V|}$$

- Another technique is to back off to a lower n-gram where there is a count. The Jelinek Mercer-interpolated smoothing:

$$\hat{P}_{int}(w_{m+1}|\boldsymbol{w}_{m-n:m})$$
$$= \lambda_{m-n:m}\frac{\#(\boldsymbol{w}_{m-n:m+1})}{\#(\boldsymbol{w}_{m-n:m})} + (1-\lambda_{m-n:m})\hat{P}_{int}(\boldsymbol{w}_{m+1}|\boldsymbol{w}_{m-(n-1):m})$$

Notice this is a recursive formulation.

Limitations of smoothed MLE based models

- Smoothing based on backoff to lower-order events, and the sequential nature of the backoff makes it hard to scale toward large n-grams.
- MLE-based language models suffer from lack of generalization across contexts

Neural language models

- Treat word prediction as a discriminative learning task, with the goal of computing the probability $P(w|u)$, where $w \in V$ is a word, and u is the context that depends on *previous* words
- Parametrize the probability $P(w|u)$ as a function of two K-dimensional dense vectors, $\boldsymbol{\beta}_w \in \mathbb{R}^K$ and $\boldsymbol{v}_u \in \mathbb{R}^K$:

  $$P(w|u) = \frac{\exp(\boldsymbol{\beta}_w \cdot \boldsymbol{v}_u)}{\sum_{w' \in V}(\exp(\boldsymbol{\beta}_{w'} \cdot \boldsymbol{v}_u))}$$

  The vector of probabilities can be computed by applying the SoftMax transformation to the vector of dot products:

  $$P(\cdot|u) = \text{SoftMax}([\boldsymbol{\beta}_{w_1} \cdot \boldsymbol{v}_u, \boldsymbol{\beta}_{w_2} \cdot \boldsymbol{v}_u, \cdots, \boldsymbol{\beta}_{w_V} \cdot \boldsymbol{v}_u])$$

- The word vectors β_w are parameters of the model and can be estimated directly, e.g., using the negative log likelihood of the training corpus as the objective

Computing the context vector

- There are different ways to compute the context vector $\boldsymbol{v}$, and one effective way is to use a **Recurrent Neural Network** or RNN. The basic idea is to recurrently update the context vector while moving through a sequence.

- Let $\boldsymbol{h}_m$ represent the contextual information at position m in the sequence. An RNN model can be defined as:

$$\boldsymbol{x}_m \triangleq \phi_{w_m}$$

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

$$P(w_{m+1}|w_1, w_2, \cdots, w_m) = \frac{\exp(\boldsymbol{\beta}_{m+1} \cdot \boldsymbol{h}_m)}{\sum_{w' \in V} \exp(\boldsymbol{\beta}_{w'} \cdot \boldsymbol{h}_m)}$$

where ϕ is a matrix of word embeddings, and $\boldsymbol{x}_m$ is the word embedding for w_m

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.

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

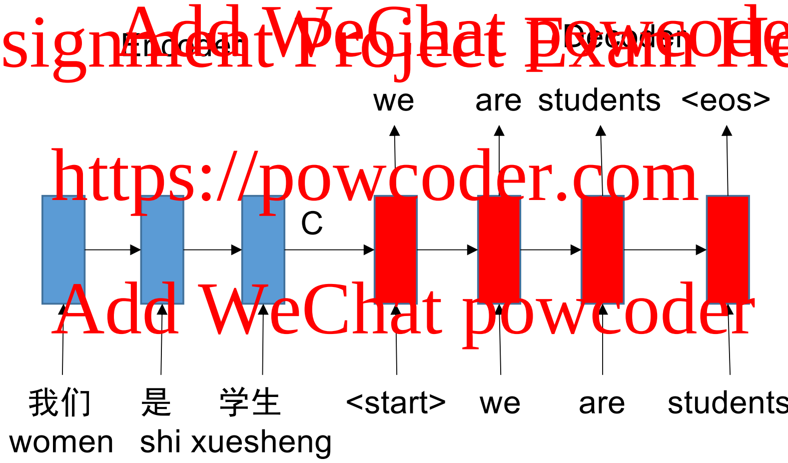
$$\boldsymbol{z} = \textsc{Encode}(\boldsymbol{w}^{(s)})$$

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

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

- The decoder is typically a recurrent neural network (e.g., LSTM) that generates one word at a time, while recurrently updating a hidden state.
- 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:

$$
\begin{aligned}
\boldsymbol{h}_m^{(s)} &= \mathrm{LSTM}(\boldsymbol{x}_m^{(s)}, \boldsymbol{h}_{m-1}^{(s)}) \\
\boldsymbol{z} &\triangleq \boldsymbol{h}_{M^{(s)}}^{(s)}
\end{aligned}
$$

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:

$$
\begin{aligned}
\boldsymbol{h}_0^{(t)} &= \boldsymbol{z} \\
\boldsymbol{h}_m^{(t)} &= \mathrm{LSTM}(\boldsymbol{x}_m^{(t)}, \boldsymbol{h}_{m-1}^{(t)})
\end{aligned}
$$

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

Tweaking the encoder decoder network

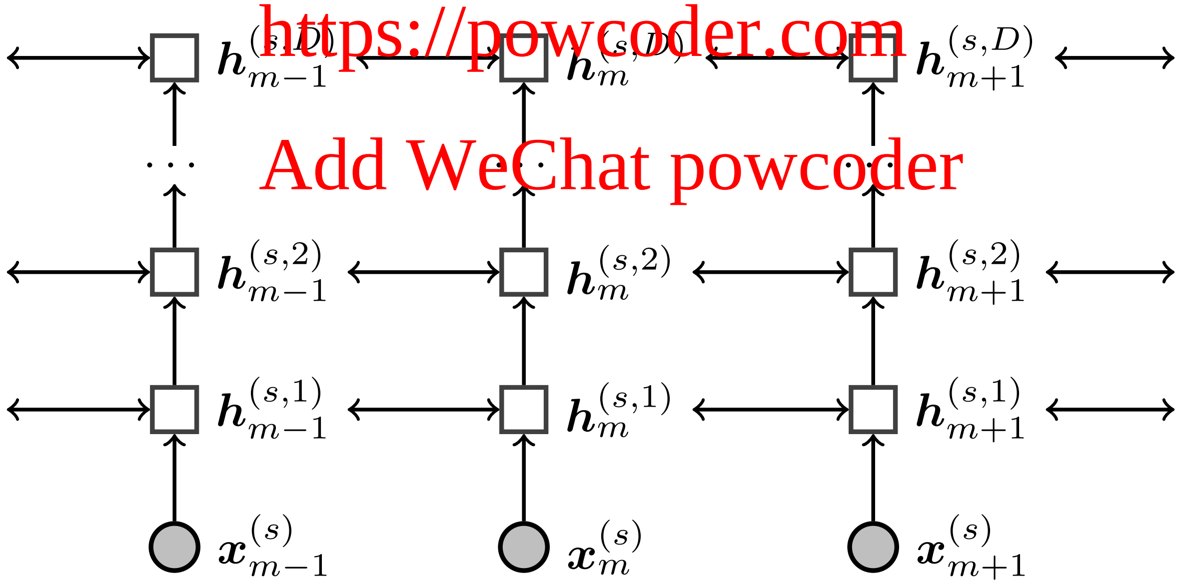
- Adding layers: The encoder and decoder network can be implemented as **deep LSTMs** with multiple layers of hidden state
- Adding **attention** to give more weight to particular word or words in the source language when generating a word in the target language

Multi-layered LSTMs

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

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

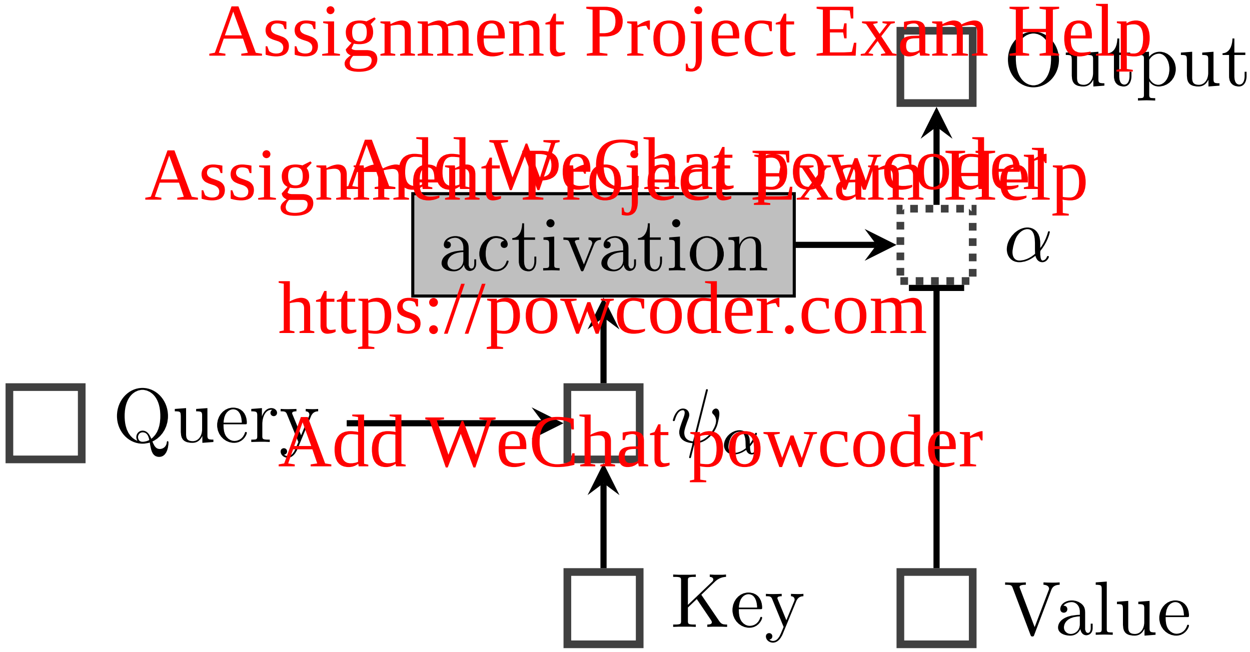
$$\boldsymbol{h}_m^{(s,i+1)} = \text{LSTM}(\boldsymbol{h}_m^{(s,i)}, \boldsymbol{h}_{m-1}^{(s,i+1)}), \forall i \geq 1$$



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:

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

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

Seq2seq: Initialization

- Word embeddings

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

- 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

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

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

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

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

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

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 Cs are used when generating target words at different time steps, e.g.,

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

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

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

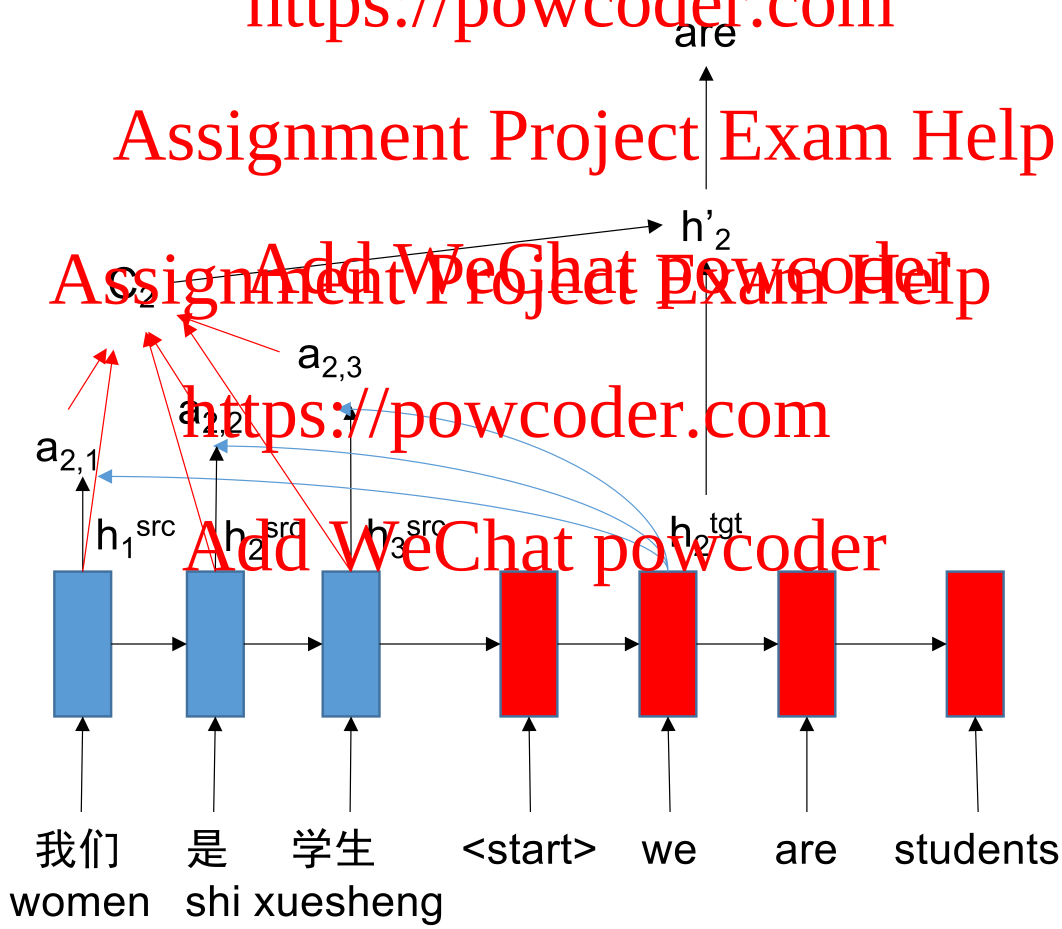
$$C_m = \sum_n \alpha_{m \to n} \times \boldsymbol{h}_n^{(s)}$$

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

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

- Other scoring variants exist

Computing attention



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Other attention variants

- Additive attention:

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

- 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 architecture 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.