Andrey Babushkin

babusand@fit.cvut.cz

presented:

Personalizing dialogue agents: I have a dog, do you have pets too?

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, & Jason Weston

https://arxiv.org/abs/1801.07243

Problems

Chit-chat (tlachání) models have several problems:

- A lack of specificity.
- No consistent personality.
- Are not very captivating.

The PERSONA-CHAT dataset

- Personas: 1155 persona descriptions, each having min. 5 sentences.
- Revised personas: a rewritten set of the same personas with no word-overlapping.
- Persona chat: 164,356 utterances over 10,981 dialogs.

Models

1 Definitions and notation

x	input sequence (previous utterances)
M^1	profile entries
M^2	interlocutor's profile entries
t	a term (one word)
d	a document (a collection of terms)
$tf\text{-}idf$	term-frequency, inverse document frequency
embedding	a vector representation of a word or sentence
IR	Information Retrieval

Four settings are compared: not having access to any profile information, having access only to M^1, having access only to M^2, or having access to both $M^1 \bigcup M^2$.

2 Ranking models

Models that rank the candidates from the training set and choose the best.

2.1 IR baseline

Finds the most similar message in the (training) dataset and outputs the response from that exchange. Similarity is measured by the $tf\text{-}idf$ weighted cosine similarity between the bags of words.

$$tf(t,d) = \frac{n_t}{\sum_k n_k}$$

$$idf(t,D) = \frac{|D|}{|\{d_i \in D | t \in d_i\}|}$$

$$idf'(t,D) = 1 + log_e(idf(t,D))$$

$$tf\text{-}idf(t,d,D) = tf(t,d) * idf(t,D)$$

$$sim(q,d) = \frac{\langle q,d \rangle}{||q|| \cdot ||d||}$$

2.2 StarSpace[1]

Performs information retrieval but by learning sentence embeddings that measure similarity between the dialog and the next utterance by optimizing the word embeddings directly for that task. Specifically, it optimizes the loss function:

$$\sum_{\substack{(q,c)\in E^+ \\ c^-\in E^-}} L^{batch}(sim(q,c), sim(q,c_1^-), \ldots, sim(q,c_k^-))$$

using margin ranking loss $\sum_{i=1}^{|E^-|} max(0, \mu - sim(q,c) + sim(q,c_i^-))$, where μ is a margin parameter. Denoting the dictionary of $\mathcal{D}$ word embeddings as W which is a $\mathcal{D} \times d$ matrix, it embeds a sequence s with $\sum_{i\in s} W_i$. One can then rank a candidate c' using $sim(q, c')$.

Similar to the IR baseline, to incorporate the profile we simply concatenate it to the query vector bag of words.

2.3 Ranking Profile Memory Network

The same concept as the StarSpace model but now we separate profile sentences and the dialog history, yielding a new query vector q^+:

$$q^+ = q + \sum s_i p_i, \quad s_i = Softmax(sim(q, p_i)), \quad Softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

One can then rank a candidate c' using $sim(q^+, c')$.

2.4 Key-Value Profile Memory Network[2]

An improvement of the previous model. We create a dictionary where keys are dialog histories, and the values are the next dialogue utterances. The first hop produces q^+ as described above, while in the second hop, q^+ is used to attend over the keys and output a weighted sum of values as before, producing q^{++}. This is then used to rank the candidates c' using $sim(q^{++}, c')$ as before.

3 Generative models

These models differ from the ranking models by the fact that they generate the next utterance word-by-word (hence the name *Generative*).

3.1 Seq2Seq[3]

The input is encoded by applying $h_t^e = LSTM_{enc}(x_t \mid h_{t-1}^e)$. GloVe is used for the word embeddings. The final state, h_t^e, or *Thought Vector*, is fed into the decoder $LSTM_{dec}$ as the initial state h_0^d. For each time step t, the decoder then produces the probability of a word j occurring in that place via the softmax, i.e.,

$$p(y_{t,j} = 1 \mid y_{t-1}, \ldots, y_1) = \frac{exp(w_j h_t^d)}{\sum_{j'=1}^{K} exp(w_{j'} h_t^d)}$$

The persona information can be included by prepending it to the input, i.e., $x' = \forall m \in M \mid\mid x$.

3.2 Generative Profile Memory Network

The model is an extension to the previous. The encoding part is the same. As for profile entries, each $m_i = \langle m_{i,1}, \ldots, m_{i,n} \rangle \in M$:

$$f(m_i) = \sum_{j}^{|m_i|} a_i m_{i,j}, \quad a_i = \frac{1}{1 + log(1 + tf)}, \quad tf = \frac{10^6}{idx^{1.07}}$$ [1]

Let F be the set of encoded memories. We generate the next output $\hat{x_t}$ by computing the mask a_t and context c_t as follows:

$$a_t = Softmax(FW_a h_t^d), \quad c_t = a_t^\intercal F, \quad \hat{x_t} = tanh(W_c [c_{t-1}, x_t])$$

[1] tf is computed from the GloVe index via Zipf's law

Appendix

1 GloVe - Global Vectors for word representation[4, 5]

1. Collect word co-occurence statistics in a form of word co-occurrence matrix X. Each element X_{ij} of such matrix represents how often word i appears in context of word j.

$$\ldots, \quad w_{i-2}, \quad w_{i-1}, \quad \overbrace{w_i}, \quad \ldots, \quad \underbrace{w_{j-1}}_{\text{window\_size}}, \quad \overbrace{w_j}, \quad w_{j+1}, \quad \ldots$$

   The weight we give for words is $decay = \frac{1}{offset}$

2. Create soft constraints for each word pair:

$$w_i^\intercal w_j + b_i + b_j = log(X_{ij})$$

   where b_i and b_j are scalar biases for the main word w_i and the context word w_j respectively.

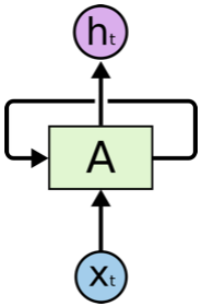
3. Define a cost function

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^\intercal w_j + b_i + b_j - log(X_{ij}))^2$$

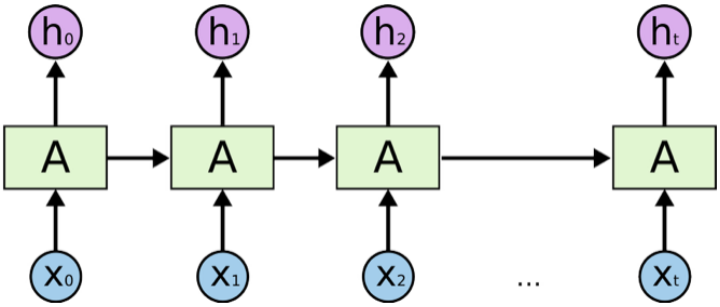
   Here f is a weighting function which help us to prevent learning only from extremely common word pairs. The GloVe authors choose the following function:

$$f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{x_{max}})^\alpha & \text{if } X_{ij} < XMAX \\ 1 & \text{otherwise} \end{cases}$$

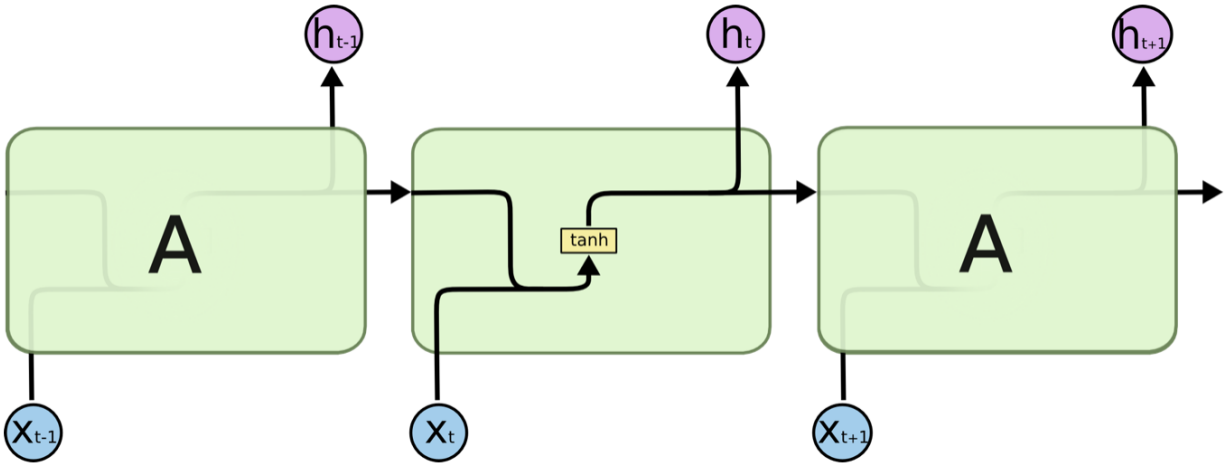
2 LSTM - Long short-term memory networks[6]



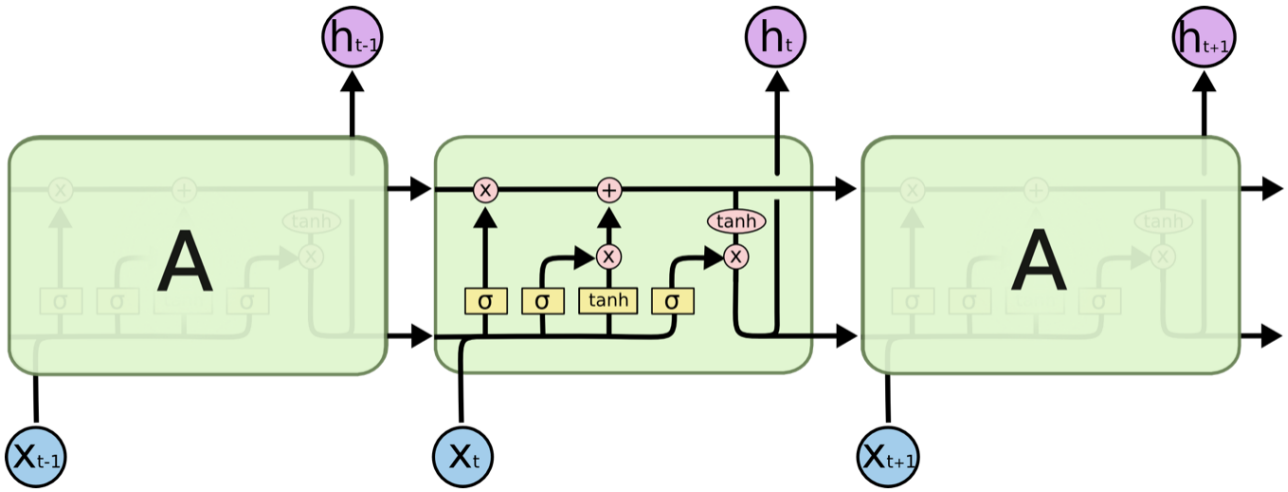
(a) Figure 1. A usual Recurrent Neural Network.



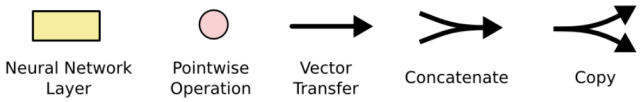
(b) Figure 2. An unrolled recurrent neural network.



(c) Figure 3. The repeating module in a standard RNN contains a single layer.



(d) Figure 4. The repeating module in an LSTM contains four interacting layers.



(e) Figure 5. The notation we'll be using.

Before we go deeper, we shall remember two important functions and their nice properties.

1. Sigmoid function $\sigma(x)$.
$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}, \quad \sigma : \mathbb{R} \to (0,1)$$
2. Hyperbolic tangent $\tanh x$.
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}, \quad \tanh x : \mathbb{R} \to (-1,1)$$

So, let's start. The main advantage of the LSTM is the ability not to forget information with time. The key to LSTMs is the *cell state*, the horizontal line running through the top of the diagram. The LSTM network contains 4 independent layers:

1. The *Forget Gate* layer. Decides what information will be removed from the cell state.
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] \; + \; b_f)$$
2. The *Input Gate* layer. Decides what information will be added to the cell state.
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] \; + \; b_i)$$
3. The *New Candidates* layer. Creates new candidate values for the cell state.
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
4. The *Output* layer. Decides what values should be outputted.
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] \; + \; b_o)$$

Finally, we compute a new cell state C_{t+1} and the output h_{t+1} ($\otimes$ denotes a pointwise multiplication operation):

$$C_{t+1} = i_t \times \tilde{C}_t \; + \; C_{t-1} \times f_t$$

$$h_{t+1} = \tanh(C_{t+1}) \times o_t$$

Well done!

References

[1] L. Wu et al. "StarSpace: Embed All The Things!" In: *ArXiv e-prints* (Sept. 2017). arXiv: 1709.03856 [cs.CL].

[2] A. Miller et al. "Key-Value Memory Networks for Directly Reading Documents". In: *ArXiv e-prints* (June 2016). arXiv: 1606.03126 [cs.CL].

[3] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to Sequence Learning with Neural Networks". In: *ArXiv e-prints* (Sept. 2014). arXiv: 1409.3215 [cs.CL].

[4] Dmitriy Selivanov. *GloVe Word Embeddings*. URL: https://cran.r-project.org/web/packages/text2vec/vignettes/glove.html.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[6] Christopher Olah. *Understanding LSTM Networks*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs.