The first sentence the second sentence

a smaller subtitle

Dewi E. Timman
12419273

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 900
1098 XH Amsterdam

Supervisor
Dr. V. Niculae

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 900
1098 XH Amsterdam

Semester 1, 2023-2024

Abstract

keywords:

Contents

Chapter 1

Introduction

What if machines can read our mind? If we can give a machine a few keywords and let the machine generate a sentence from these keywords, our lives would become more productive and efficient. This is what autocomplete systems are trying to achieve. The way in which we choose the keywords is also important. Taking just the first or the last few words of a sentence as keywords usually does not capture the full meaning of the sentence. For example, if someone want to capture the meaning of *'I live in Amsterdam'* in a few keywords, the words *'live Amsterdam'* would probably be choosen. Thus, the keywords come from multiple places in the sentence. Therefore, autocomplete systems need to use more complex models to be more efficient and accurate.

1.1 Literature review

1.1.1 Autocomplete communication game

The same autocomplete communication game is considered as in Lee et al. (2019). In this game, a human (called user) encodes a sentence into keywords. These keywords are then decoded by a machine (called system) to retrieve the full, initial sentence. A schematic overview is given in figure 1.1. The communication game is succesfull if the retrieved sentence is the same as the initial sentence.

More formally, a target sentence $x = (x_1, \ldots, x_m)$ is communicated by a user through the keywords $z = (z_1, \ldots, z_n)$. Note that z is a subsequence of x. The system then tries to retrieve the target sentence by decoding the keywords. The target sentence is described by the keywords using encoding strategy $q_\alpha(z|x)$ and the system decodes the keywords by using decoding strategy $p_\beta(x|z)$.

For a model to be efficient, the number of keywords needs to be as low as possible. In addition, for a model to be accurate, the probability of reconstructing

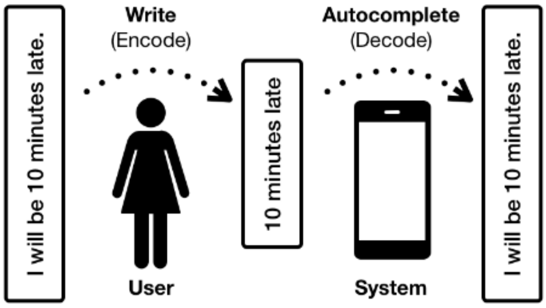


Figure 1.1: schematic overview of the communication game. Figure from Lee et al. (2019).

x from z needs to be as high as possible. Therefore, a cost and a loss, respectively, can be defined:

$$\mathrm{cost}(x, \alpha) = \mathbb{E}_{q_\alpha(z|x)}[\mathrm{length}(z)] \tag{1.1}$$

$$\mathrm{loss}(x, \alpha, \beta) = \mathbb{E}_{q_\alpha(z|x)}[-\log p_\beta(x|z)] \tag{1.2}$$

1.1.2 Segmentation model

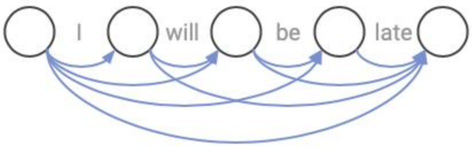
1.1.2.1 General idea

With a segmentation model all possible segmentation can be made. A segmentation model scores every possible segmentation. With these scores, the model can determine what the best possible segmentation is.

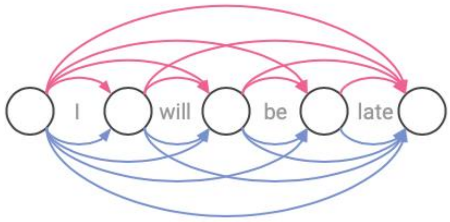
1.1.2.2 Segmentation model for text

So how does the segmentation model work for text? If we have a sentence, e.g. *'I will be late'*, we can use fencepost indexing and represent the fenceposts as nodes in a directed acyclic graph (DAG). We can then draw edges between those nodes that represent segments. Those segments can be seen as (groups of) words. In figure 1.2a, a DAG can be seen in which all the possible segments are showed. In the case of the autocomplete communication model described before, a segment is either kept or not. Therefore, we can have one edge representing 'keep' and one representing 'do not keep', resulting in figure 1.2b. If the pink edges are taken as 'do not keep' and the blue ones as 'keep', two possible segmentations can be seen in figure 1.2c and 1.2d. Both segmentations result in the keywords *'will be late'*.

All segments can be scored with the help of the Forward algorithm. After scoring each segment, the model can then choose the segments with the highest scores to retrieve the best possible segmentation.



(a) Segments as a DAG



(b) More segments as a DAG

(c) Possible segmentation

(d) Another possible segmentation

Figure 1.2: Segmentation model.

1.1.3 Structured latent variables

How does the model work?

1.2 Current research

Previous research did not take structure into account (Lee et al., 2019; Bar-Yossef & Kraus, 2011; Svyatkovskiy, Zhao, Fu, & Sundaresan, 2019). Since language is structured, it makes sense to use a structured model as an autocomplete model. In this research, we look at how we can use a latent segmentation model to retrieve keywords from a sentence. The segmentation model will be implemented in the encoder of the encoder-decoder model in order to choose the best keywords from the sentence.

Chapter 2

Experiment 1 Replication

In the first experiment the autoencoder model from Lee et al. (2019) was replicated.

2.1 Method

2.1.1 Data

The same data was used as in Lee et al. (2019). The data used to train the model consisted of 500K randomly sampled sentences from the Yelp restaurant reviews corpus (Yelp, 2017). Another 10K sentences were used to test the model. The sentences had at most 16 tokens. The reviews were segmented into sentences following the same procedure as in Guu, Hashimoto, Oren, and Liang (2018).

2.1.2 Experimental Design

The model used is an encoder-decoder model. The encoder, using the encoding strategy $q_\alpha(z|x)$, takes as input a target sequence $x = (x_1, \ldots, x_m)$ and outputs a sequence of keywords $z = (z_1, \ldots, z_n)$. The decoder, using the decoding strategy $p_\beta(x|z)$, then takes these keywords as input and outputs a predicted sequence $y = (y_1, \ldots, y_k)$. The better the autoencoder works, the more likely it is that x and y are equal.

2.1.3 Model description

Encoder. The encoder embeds the tokens and uses a uni-directional LSTM to score the tokens. An additional linear layer followed by sigmoid function is used to determine the probability of keeping each token. From these probabilities, a mask is sampled from a Bernoulli distribution. Finally, the sequence of kept tokens and the log probability of the mask are returned.

Decoder. The decoder itself is also an encoder-decoder model (the encoder of this model is referred to as encoder*). The encoder* first embeds the tokens of the subsequence. It then encodes the embedding using a bi-directional LSTM.

The decoder decodes the full sentence. It therefore embeds the already decoded sequence (or just the <sos> symbol if there is none) into a 300-dimensional vector and concatenates the last hidden state of the encoder* to it. This embedding is the input for another uni-directional LSTM. Finally, the probability of the next word is calculated using global attention and the full sentence and its log probability are returned.

Optimization. The goal of the model is to be as efficient and accurate as possible. If equation 1.1 and 1.2 are merged and a parameter λ is added to represent the trade-off between the two, the goal becomes the following:

$$\min_{\alpha,\beta} \mathbb{E}[\text{cost}(x,\alpha)] + \lambda\mathbb{E}[\text{loss}(x,\alpha,\beta)]. \tag{2.1}$$

Here the expactation is taken over x. Since the gradients of equation 2.1 cannot be calculated, it is approximated with Monte Carlo. The gradients can then by calculated as following (see appendix A for the exact derivations):

$$\nabla_\alpha F(\alpha,\beta) = \mathbb{E}_{q_\alpha(z|x)}[\nabla_\alpha \log q_\alpha(z|x) f(z,\beta)], \tag{2.2}$$

$$\nabla_\beta F(\alpha,\beta) = \mathbb{E}_{q_\alpha(z|x)}[\nabla_\beta f(z,\beta)]. \tag{2.3}$$

Where, f and the score function estimator F (and its Monte Carlo approximation) are:

$$f(z,\beta) = \text{length}(z) + \lambda(-\log p_\beta(x|z)), \tag{2.4}$$

$$F(\alpha,\beta) = \mathbb{E}_{q_\alpha(z|x)}[f(z,\beta)], \tag{2.5}$$

$$F(\alpha,\beta) \overset{\text{M.C.}}{\approx} \frac{1}{M}\sum_i f(z^{(i)},\beta). \tag{2.6}$$

Here M is the amount of samples drawn from $q_\alpha(z|x)$ and x is dropped for clarity.

2.1.3.1 Hyperparameters

What are the hyperparameters?

2.2 Results

Chapter 3

Experiment 2 Segmentation Model

3.1 Method

3.2 Results

Chapter 4

Results

Chapter 5

Conclusion

Chapter 6

Discussion

References

Bar-Yossef, Z., & Kraus, N. (2011). Context-sensitive query auto-completion. In *Proceedings of the 20th international conference on world wide web* (p. 107-116). ACM.

Guu, K., Hashimoto, T. B., Oren, Y., & Liang, P. (2018). Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, *6*, 437-450.

Lee, M., Hashimoto, T. B., & Liang, P. (2019). Learning autocomplete systems as a communication game.

Svyatkovskiy, A., Zhao, Y., Fu, S., & Sundaresan, N. (2019). Pythia: Ai-assisted code completion system.

Yelp. (2017). *Yelp dataset challenge, round 8.* Retrieved from `https://www.yelp.com/dataset_challenge`

Appendix A

Optimization derivations

Using equations 1.1 and 1.2, the goal is:

$$\begin{aligned}
&\min_{\alpha,\beta} \mathbb{E}[\mathrm{cost}(x,\alpha)] + \lambda \mathbb{E}[\mathrm{loss}(x,\alpha,\beta)] \\
&= \min_{\alpha,\beta} \mathbb{E}[\mathrm{cost}(x,\alpha) + \lambda \mathrm{loss}(x,\alpha,\beta)] \\
&= \min_{\alpha,\beta} \frac{1}{D} \sum_{x \in D} [\mathrm{cost}(x,\alpha) + \lambda \mathrm{loss}(x,\alpha,\beta)].
\end{aligned}$$

f can then be defined as:

$$\begin{aligned}
f(z,\beta) &= \mathrm{cost}(x,\alpha) + \lambda \mathrm{loss}(x,\alpha,\beta) \\
&= \mathrm{length}(z) + \lambda(-\log p_\beta(x|z)).
\end{aligned}$$

And F as:

$$\begin{aligned}
F(\alpha,\beta) &= \mathbb{E}_{q_\alpha(z|x)}[[\mathrm{length}(z) + \lambda(-\log p_\beta(x|z))]] \\
&=^* \sum_{z \in Z} [q_\alpha(z|x) f(z,\beta)] \\
&= \mathbb{E}_{q_\alpha(z|x)}[f(z,\beta)]
\end{aligned}$$

where Z consists of all the possible masks of size x. In the step marked with *, the following rule is used: $\mathbb{E}_{P(A)}[f(A)] = \sum_{a \in A} P(a) f(a)$.

Thus, the goal then becomes:

$$\min_{\alpha,\beta} F(\alpha,\beta).$$

Since

$$F(\alpha, \beta) \overset{\text{M.C.}}{\approx} \frac{1}{M} \sum_{i} f(z^i, \beta)$$

we also have that:

$$\nabla_\alpha F(\alpha, \beta) \overset{\text{M.C.}}{\approx} \nabla_\alpha \sum_{i} f(z^i, \beta)$$
$$\nabla_\beta F(\alpha, \beta) \overset{\text{M.C.}}{\approx} \nabla_\beta \sum_{i} f(z^i, \beta).$$

Where M.C. stands for Monte Carlo estimation.

Therefore, we can calculate the gradients with respect to α as following:

$$\begin{aligned}
\nabla_\alpha F(\alpha, \beta) &= \nabla_\alpha \mathbb{E}_{q_\alpha(z|x)}[f(z, \beta)] \\
&= \nabla_\alpha \sum_z [q_\alpha(z|x) f(z, \beta)] \\
&= \sum_z \nabla_\alpha (q_\alpha(z|x) f(z, \beta)) \\
&=^* \sum_z (q_\alpha(z|x)[\nabla_\alpha \log q_\alpha(z|x) f(z, \beta)]) \\
&= \mathbb{E}_{q_\alpha(z|x)}[\nabla_\alpha \log q_\alpha(z|x) f(z, \beta)].
\end{aligned}$$

In the step marked with *, a log-derivative trick is used: $\nabla_t \log h(t) = \frac{\nabla_t h(t)}{h(t)}$.

And the gradient with respect to β:

$$\begin{aligned}
\nabla_\beta F(\alpha, \beta) &= \nabla_\beta \mathbb{E}_{q_\alpha(z|x)}[f(z, \beta)] \\
&= \nabla_\beta \sum_z [q_\alpha(z|x) f(z, \beta)] \\
&= \sum_z \nabla_\beta (q_\alpha(z|x) f(z, \beta)) \\
&= \sum_z (q_\alpha(z|x)[\nabla_\beta f(z, \beta)]) \\
&= \mathbb{E}_{q_\alpha(z|x)}[\nabla_\beta f(z, \beta)].
\end{aligned}$$