

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

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Literature review . . . . .	4
1.1.1	Autocomplete communication game . . . . .	4
1.1.2	Segmentation model . . . . .	5
1.1.3	Structured latent variables . . . . .	6
1.2	Current research . . . . .	6
<b>2</b>	<b>Experiment 1</b>	
	<b>Replication</b>	<b>7</b>
2.1	Method . . . . .	7
2.1.1	Data . . . . .	7
2.1.2	Experimental Design . . . . .	8
2.1.3	Model description . . . . .	8
2.1.4	Hyperparameters . . . . .	9
2.1.5	Optimizations . . . . .	9
2.2	Results . . . . .	10
<b>3</b>	<b>Experiment 2</b>	
	<b>Segmentation Model</b>	<b>11</b>
3.1	Method . . . . .	11
3.1.1	Score matrix . . . . .	11
3.1.2	Model description . . . . .	12
3.2	Results . . . . .	12
<b>4</b>	<b>Results</b>	<b>14</b>
<b>5</b>	<b>Conclusion</b>	<b>15</b>
<b>6</b>	<b>Discussion</b>	<b>16</b>
	<b>References</b>	<b>17</b>



# 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 chosen. 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 successful if the retrieved sentence is the same as the initial sentence.

More formally, a target sentence  $x = (x_1, \dots, x_m)$  is communicated by a user through the keywords  $z = (z_1, \dots, z_n)$ . 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

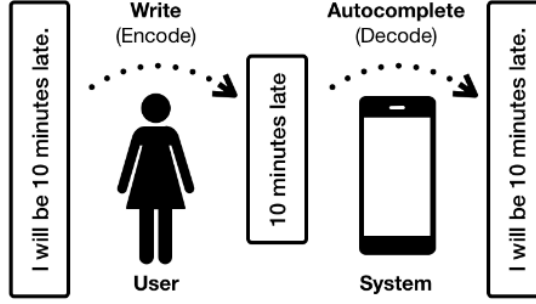


Figure 1.1: schematic overview of the communication game. Figure from Lee et al. (2019).

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

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

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

### 1.1.2 Segmentation model

**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.

**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'*.

The best segmentations can be found with the help of dynamic programming algorithms such the Viterbi and the forward algorithms.

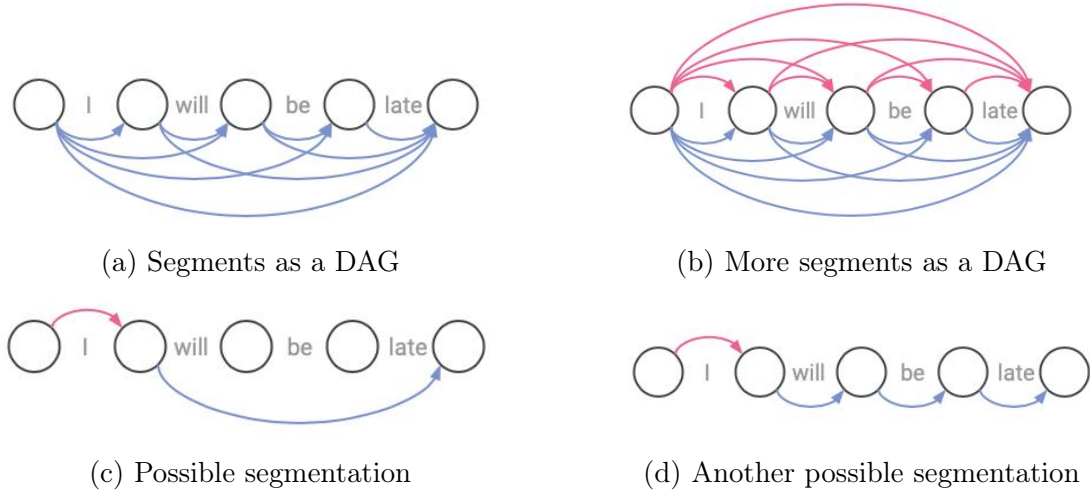


Figure 1.2: Segmentation model.

### 1.1.3 Structured latent variables

A latent variable is an unobservable variable. For these variables, there are no labels available. Therefore, it is not possible to use supervised learning on them. In the case of this model, we try to recover the full sentence by inferring what is a good mask. The mask determines what words are good keywords and thus is a latent variable. In addition, the segmentation model makes this a structured variable.

## 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

To replicate the model, an encoder-decoder model was made. The encoder chooses which words to keep as keywords, and the decoder tries to retrieve the full sentence of the keywords.

#### 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 50K sentences were used to evaluate 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).

The model of Lee et al. (2019) also predicts capital letters and whitespaces. However, for English we can just assume that after every word a space is used. Therefore, the data was adjusted to leave out the characters for whitespaces. Something similar is also true for capital letters: usually these only occur at the start of a sentence or in names. It is thus not necessary to also predict capital letters.

**TODO:**

1. Table with example sentences
2. Adjust the number of sentences used for the training and the validation set (Due to computational resources ...)



### 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  $x = (x_1, \dots, x_m)$  and outputs a sequence of keywords  $z = (z_1, \dots, z_n)$ . The decoder, using the decoding strategy  $p_\beta(x|z)$ , then takes these keywords as input and outputs a predicted sequence  $y = (y_1, \dots, y_k)$ . 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 (Hochreiter & Schmidhuber, 1997) 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 32-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 (Luong, Pham, & Manning, 2015; Bahdanau, Cho, & Bengio, 2016) and the full sentence and its log probability are returned.

During training, the decoder is given the keywords and at every step the model takes the correct word from the target sequence and calculates the probability of that sequence. During evaluation, however, the decoder uses a greedy decoding strategy and thus takes the word with the highest probability as the next word.

**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  $\lambda$  is added to represent the trade-off between the two, the goal becomes the following:

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

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

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

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

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

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

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

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

Here  $M$  is the amount of samples drawn from  $q_{\alpha}(z|x)$ .

### 2.1.4 Hyperparameters

During training, the sentences start with a <eos> symbol and end with a <eos> symbol. During evaluation of the model, those symbols are removed.

The model is implemented using PyTorch (Paszke et al., 2019). After each token is embedded, ReLu is used. During training, dropout is used to prevent overfitting. The hidden dimensions of the LSTM layers are all 32. To optimize the model an Adam optimizer (Kingma & Ba, 2017) is used with a learning rate of 0.01. Multiple models were trained for 10 epochs with different values for  $\lambda$ .

### 2.1.5 Optimizations

To let the model predict better results faster, a few optimizations were done.

**Copy mechanism.** The copy mechanism (Gu, Lu, Li, & Li, 2016) in the decoder determines if it wants to copy the current token or wants to generate a new token. It therefore calculates the probability of the new word as following:

$$p(w) = (1 - p_{\text{gen}}) \cdot p_{\text{copy}}(w) + p_{\text{gen}} \cdot p_{\text{word}}(w). \quad (2.7)$$

For the calculation of the probability of the to be copied word,  $p_{\text{copy}}(w)$ , the global attention mechanism from the decoder is used. To calculate the probability of generating a new word,  $p_{\text{gen}}$ , the last hidden decoder state, the attention mechanism and the token embedding of the new to generated word are used in a linear layer followed by a sigmoid.  $p_{\text{word}}(w)$  is the probability of the to be generated word if a word is being generated.

**Adjusted vocabulary.** The model makes a vocabulary to translate words to numbers. During the evaluation of the model, a lot of unknown tokens were encountered. This was due to words that were not present in the training set but were in the validation set. To solve this problem, the vocabulary was made using the whole dataset, instead of just the training set. Lee et al. (2019) used the copy mechanism and a dynamic vocabulary for this, but for the task at hand and the amount of data this also works.

**Variance reduction.** Because of the use of a SFE the gradient can be more prone to noise (Niculae, Corro, Nangia, Mihaylova, & Martins, 2023). To solve this problem, its variance can be reduced in a couple of ways. One of those solutions is to sample a second subsentence in the encoder (Rennie, Marcheret, Mroueh, Ross, & Goel, 2017). With this second subsentence,  $f(z')$  can be determined and the loss can be updated as following:

$$\mathbb{E}_{q_{\alpha}(z|x)}[\nabla_w \log q_{\alpha}(z|x) \cdot (f(x, z, \beta) - f(x, z', \beta))] + \mathbb{E}_{q_{\alpha}(z|x)}[\nabla_w f(x, z, \beta)], \quad (2.8)$$

where  $w$  is a parameter (i.e. either  $\alpha$  or  $\beta$ ). Note that the gradient is not changed when subtracting  $f(z')$  from  $f(z)$  since it can be seen as constant because it is independent of  $z$ .

## 2.2 Results

# Chapter 3

## Experiment 2 Segmentation Model

In the second experiment, the model from chapter 2 is expanded with a segmentation model.

### 3.1 Method

An autoencoder model is made in which the encoder uses a segmentation model to choose the sequence of keywords  $z$ . The same data was used as described in section 2.1.1.

#### 3.1.1 Score matrix

In order for this model to work, it needs a score tensor,  $a$ . This tensor consists of two upper triangle matrices of  $m + 1 \times m + 1$ , called  $a_0$  and  $a_1$ . Both matrices represent the score of segments in the DAG made for each sentence as described in 1.1.2. Matrix  $a_1$  represents the scores for when the segment is true, and  $a_0$  for when the segment is false. Matrix  $a_0$  consists of only zeros to simplify the model.

To construct  $a_1$ , an embedding matrix  $H$  with dimensions  $m \times d$  is used. Here  $m$  is the amount of tokens in  $x$  and  $d$  the hidden dimension of the LSTM. In addition, a weight matrix with dimensions  $d \times d$  is used. This weight matrix is learned by the encoder. Matrix  $a_1$  is then calculate as following:

$$a_{ij1} = h_i^T w h_j \quad (3.1)$$

And more efficiently, with matrices:

$$a_1 = H w H^T \quad (3.2)$$

Note that the scores learned for  $a_1$  can also be negative. Therefore, the model can prefer to leave out a segment when needed.

### 3.1.2 Model description

The same model as in section 2.1.3 was used. Only the encoder part was adjusted. Instead of using an LSTM and a linear layer to score each token, the segmentation model is used.

**Encoder.** The encoder embeds each token. It then uses a one-directional LSTM. The output of this LSTM is the matrix  $H$ , which is used to make the score matrix  $a$ , as described in the previous section. Then, using dynamic programming algorithms, a segmentation is sampled and its probability is calculated. As described in section 2.1.5, variance reduction is applied and thus also a second segmentation is sampled. These segmentations are then converted into masks.

**Dynamic programming.** To sample a segmentation, the forward filtering, backward sampling algorithm (Murphy, 2012) was used. This algorithm first calculates the logsumexp of all possible segmentations and then samples from this distribution. Pseudocode for this algorithm can be found in algorithm 2.

To calculate the probability of the segmentation, the forward algorithm was used. This algorithm calculates the logsumexp of all possible segmentations. Pseudocode for this algorithm can be found in algorithm 1. The logsumexp is then used in combination with the score of the segmentation (which can be calculated using  $a$ ) to calculate the probability of the segmentation

## 3.2 Results

---

**Algorithm 1** Forward filtering, backward sampling algorithm

---

**Input:**  $a$ **Output:** sample

```
 $n \leftarrow a.shape[0]$ 
 $q_0 \leftarrow 0$ 
for  $i = 1, \dots, n$  do
     $q_i \leftarrow \log \sum_{0 \leq j < i} \exp q_j + a_{ji}$ 
end for

 $y \leftarrow []$ 
 $i \leftarrow n - 1$ 
while  $i > 0$  do
    sample  $j < i$  with probability  $p_j \leftarrow \exp(a_{ji} + q_j - q_i)$ 
     $y \leftarrow ji \frown y$ 
     $i \leftarrow j$ 
end while

return  $y$ 
```

---

---

**Algorithm 2** Forward algorithm

---

**Input:**  $a$ **Output:** log-normalizer

```
 $n \leftarrow a.shape[0]$ 
 $q_0 \leftarrow 0$ 
for  $i = 1, \dots, n$  do
     $q_i \leftarrow \log \sum_{0 \leq j < i} \exp q_j + a_{ji}$ 
end for

return  $q_n$ 
```

---

## Chapter 4

# Results

Chapter 5

Conclusion



## Chapter 6

# Discussion

# References

- Bahdanau, D., Cho, K., & Bengio, Y. (2016). *Neural machine translation by jointly learning to align and translate*. Retrieved from <https://arxiv.org/abs/1409.0473>
- 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.
- Gu, J., Lu, Z., Li, H., & Li, V. O. K. (2016). Incorporating copying mechanism in sequence-to-sequence learning. *CoRR*, *abs/1603.06393*. Retrieved from <http://arxiv.org/abs/1603.06393>
- 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.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.
- Kingma, D. P., & Ba, J. (2017). *Adam: A method for stochastic optimization*. Retrieved from <https://arxiv.org/abs/1412.6980>
- Lee, M., Hashimoto, T. B., & Liang, P. (2019). Learning autocomplete systems as a communication game.
- Luong, M., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *CoRR*, *abs/1508.04025*. Retrieved from <http://arxiv.org/abs/1508.04025>
- Murphy, K. P. (2012). *Machine learning : a probabilistic perspective*. Cambridge, Massachusetts: MIT Press.
- Niculescu, V., Corro, C. F., Nangia, N., Mihaylova, T., & Martins, A. F. T. (2023). *Discrete latent structure in neural networks*. Retrieved from <https://arxiv.org/abs/2301.07473>
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024-8035). Curran Associates, Inc. Retrieved from <http://papers.neurips.cc/paper/9015-pytorch-an>

- `-imperative-style-high-performance-deep-learning-library.pdf`
- Rennie, S. J., Marcheret, E., Mroueh, Y., Ross, J., & Goel, V. (2017). Self-critical sequence training for image captioning.
- 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](https://www.yelp.com/dataset_challenge)

# Appendix A

## Optimization derivations

Using equations 1.1 and 1.2, the goal is:

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

Here  $D$  is the set of all target sentences.

Then  $f$  can be defined as:

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

And  $F$  as:

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

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

Thus, the goal then becomes:

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

The gradient with respect to  $\alpha$  can be calculated as following:

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

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  $\beta$ :

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