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

To make tasks more efficient, an autocomplete system can predict a full sentences from a partial sentence. To do this, the system needs to be as accurate and efficient (as few tokens as possible) as possible. In order to train such an autocomplete system, an encoder decoder model is made. The encoder extracts keywords from a sentence, and the decoder tries to predict the full sentence from these keywords. First, the model of Lee et al. (2019) was replicated. This work was one of the first to extract keywords from across the full sentence instead of the first or the last few words. However, this was done in an unstructured manner. Since human language is structured, this calls for a structured manner of extracting keywords. Therefore, the second model in this paper is a structured model. In order to achieve this, the encoder was adjusted to extract keywords using a segmentation model. With the help of dynamic programming algorithms the keywords could be extracted resulting in ....

TODO:

1. add results
2. add conclusion
3. add discussion

Keywords: segmentation, dynamic programming, autocompletion, autoencoder, communication, latent variables

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

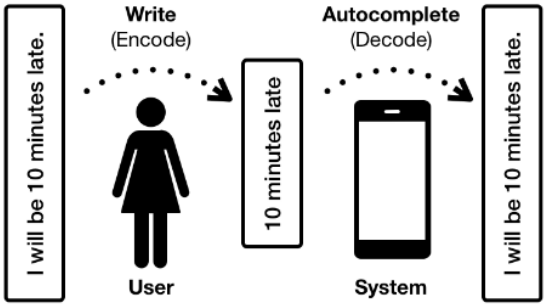


Figure 1.1: schematic overview of the communication game. Figure from Lee et al. (2019).

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

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

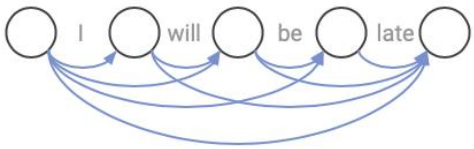
$$\text{loss}(x, \alpha, \beta) = \mathbb{E}_{q_\alpha(z|x)}[-\log p_\beta(x|z)] \tag{1.2}$$

1.1.2 Segmentation model

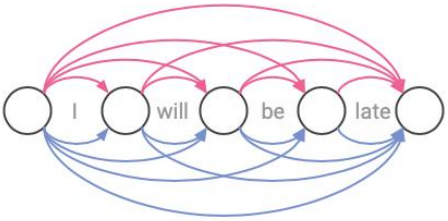
General idea. If there is a rod of length n and we can cut this rod at every marker, how can we best find the maximal total value of the resulting pieces? This is called the rod cutting problem. The segmentation model gives a solution to the rot cutting problem. The segmentation model takes the scores of all pieces of the rod, called segments. Those segments can be of length 1, 2 or even n. With these scores, the model determines what the best possible segmentation is. To find the best segmentation and the probability of a segmentation, the model makes use of dynamic programming algorithms such as the Viterbi algorithm (Rabiner, 1989) and the forward algorithm. The segmentation model is essentially a simpler version of a hidden semi-Markov model (K. Murphy, 2002).

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 to represent a sentence as a rod which can be cut at the fenceposts (see also figure 1.2a). The fenceposts can also represent 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 1.2b, 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 1.2c. If the pink edges are taken as 'do not keep' and the blue

(a) Fencepost indexing



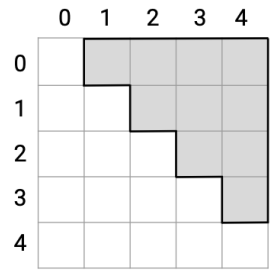
(b) Possible segments in a DAG



(c) Possible segments in a DAG when each segment can either be true or false

(d) A possible segmentation

(e) Another possible segmentation



(f) Score matrix A

Figure 1.2: Segmentation model

ones as 'keep', two possible segmentations can be seen in figure 1.2d and 1.2e. Both segmentations result in the keywords *'will be late'*.

The score of a segmentation can be calculated by summing up all scores of its segments. The assigned score of a segment can visualized as a score matrix A and has size $(m+1) \times (m+1)$. A segment, e.g. segment $i-j$, only has a score if it goes to a node further in the sequence, i.e. $i < j$. Therefore, only the upper triangle of the matrix is used. This is denoted in figure 1.2f by the gray squares. The scores of the segmentations can then be used to calculate probabilities and to sample from a distribution over segments.

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 (Bar-Yossef & Kraus, 2011; Lee et al., 2019; Svyatkovskiy et al., 2019). Since language is structured, it makes sense to use a structured model as an autocomplete model. Furthermore, previous work has shown that the prediction of segmentations can lead to better results than models that do not explicitly represent segments (Kong et al., 2016; Sarawagi & Cohen, 2004). Therefore, in this research, we look at how a latent segmentation model can be used 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 et al. (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 $x = (x_1, \ldots, x_m)$ 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 (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 et al., 2015; Bahdanau et al., 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 λ is added to represent the trade-off between the two, the goal becomes the following:

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

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):

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

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

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

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

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

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

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

2.1.4 Hyperparameters

During training, the sentences start with a <sos> 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 λ.

2.1.5 Optimizations

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

Copy mechanism. The copy mechanism (Gu et al., 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:

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

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_{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 vocobulary 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 et al., 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 et al., 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 α or β). 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, $\boldsymbol{A}$. 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. The matrix A_1 represents the scores for when the segment is true, and A_0 for when the segment is false. The 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 W with dimensions $d \times d$ is used. This weight matrix is learned by the encoder. Matrix A_1 is then calculated as following:

$$A_{ij1} = h_i^T W h_j \tag{3.1}$$

And more efficiently, with matrices:

$$A_1 = HWH^T \tag{3.2}$$

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 (K. P. 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: score tensor A with shape $(m+1) \times (m+1) \times 2$
Output: sampled segmentation

$n \leftarrow A.\text{shape}[0]$
$q_0 \leftarrow 0$
for $i = 1, \ldots, n$ **do**
  $q_i \leftarrow \log \sum_{0 \le j < i} (\exp(q_j + a_{ji0}) + \exp(q_j + a_{ji1}))$
end for

$y \leftarrow []$
$i \leftarrow n - 1$
while $i > 0$ **do**
  sample $j < i$ and k with probability $p_j \leftarrow \exp(a_{ji0} + q_j - q_i) + \exp(a_{ji1} + q_j - q_i)$
  $y \leftarrow (ji, k) \frown y$
  $i \leftarrow j$
end while

return y

Algorithm 2 Forward algorithm

Input: score tensor A with shape $(m+1) \times (m+1) \times 2$
Output: log-normalizer

$n \leftarrow A.\text{shape}[0]$
$q_0 \leftarrow 0$
for $i = 1, \ldots, n$ **do**
  $q_i \leftarrow \log \sum_{0 \le j < i} (\exp(q_j + a_{ji0}) + \exp(q_j + a_{ji1}))$
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`

Kong, L., Dyer, C., & Smith, N. A. (2016). *Segmental recurrent neural networks.* Retrieved from `https://arxiv.org/abs/1511.06018`

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. (2002). Hidden semi-markov models (hsmms). , 1-13.

Murphy, K. P. (2012). *Machine learning : a probabilistic perspective.* Cambridge, Massachusetts: MIT Press.

Niculae, 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

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257-286.

Rennie, S. J., Marcheret, E., Mroueh, Y., Ross, J., & Goel, V. (2017). Self-critical sequence training for image captioning.

Sarawagi, S., & Cohen, W. W. (2004). Semi-markov conditional random fields for information extraction. In L. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in neural information processing systems* (Vol. 17). MIT Press. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2004/file/eb06b9db06012a7a4179b8f3cb5384d3-Paper.pdf

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:

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

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 α 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 β:

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