MSc Artificial Intelligence.
Master Thesis

Transformer model for query suggestion.

by
Daniela Solis Morales
Student-ID:11392061

September 12, 2018

36 European Credits
February 2018 - September 2018

Supervisor:
Dr. Ilya Markov

Assessor:
Prof. Dr. Maarten de Rijke

University of Amsterdam
Abstract

Query suggestions are query proposals after one or more queries have been submitted. They help users refine their queries when using a search engine and they can guide the direction of the search within a session by either suggesting queries that dive deeper into the current search direction or by suggesting a change of direction into a different search space. So far, query suggestion models have focused on complex recurrent encoder-decoder architectures to solve the task. Such complex architectures require a great amount of computational power and hours to train.

This thesis proposes a novel neural model that reduces the complexity of current state-of-the-art models by using an encoder-decoder architecture that is based solely on attention. For this, it uses a Transformer model, that was first introduced for neural machine translation task, and it was shown to outperform state-of-the-art techniques (Vaswani et al., 2017).

The AOL dataset is used to compare the model proposed with current state-of-the-art query suggestion models (Sordoni et al. (2015) and Dehghani et al. (2017)). The empirical experiments show that it is possible to use a Transformer architecture for query suggestion task. Furthermore, results indicate that reducing the complexity of the architecture does not compromise the model performance. Simpler models are able to achieve good results.

This opens the door for future work to explore many different variants of Transformer models that are novel in the field of query suggestion task.

Keywords

Query Suggestion · Transformer Model · Sequence to Sequence Model
Acknowledgements

Firstly, I would like to thank my daily supervisor Ilya Markov. Throughout the thesis, he has always made time to give me valuable advice and feedback.

Many thanks to Mostafa Dehghani. Even though he was not officially my supervisor, he was always there to help me. His supervision kept the project feasible, without his great support and help, it would not have been possible.

Finally, I would like to thank my family, friends and boyfriend for their endless support. No words can express how grateful I am to have them.
1 Introduction

Query suggestions are related queries that happen after one or more queries have been submitted in a search engine. They help users stipulate the information they need. They improve the search process by making it more clear for both the user and the search engine. They can guide the direction of the search by either suggesting queries that dive deeper into the current search direction or by suggesting a change of direction into a different search space. Moreover, query suggestions might also help users when they do not know the specific terminology or technical vocabulary needed to formulate an adequate query that leads them to the required information. Unfortunately, users search intent and the context of the search cannot be observed directly. Hence, generating relevant suggestions is a challenging task.

Over the years different approaches have been taken to solve this task. For instance, one may think that finding relevant suggestions using co-occurring terms in documents retrieved during the search (Anick and Tipirneni, 1999; Jones and Staveley, 1999; Xu and Croft, 1996) or between queries in a session (Huang et al., 2003) may be reliable indicators to formulate adequate query suggestions. However, classical count-base methods are prone to data sparsity. Other methods take advantage of the "wisdom of crowds", using search engine logs and the clicked documents throughout user sessions (Huang et al., 2003; Boldi et al., 2009). With them, they identify patterns and find relationships between queries to generate suggestions. Another group of scientific research leverages not only search logs but also external resources. For instance, extracting query structures using Wordnet since it can be essential to understanding query reformulation behaviours (Szpektor et al., 2011) or applying filters to remove poor suggestions (Desautels et al., 2014).

More sophisticated methods use neural models. These models leave aside the assumption that the best recommendations have already been observed, as this might not be the case for rare queries. Therefore, neural models can produce synthetic suggestions1. Furthermore, having more complex models also allow us to achieve context awareness. Neural models use a Sequence-to-sequence (seq2seq) architecture (Sordoni et al., 2015; He et al., 2016; Dehghani et al., 2017; Chen et al., 2018). In this architecture, the sequence of queries issued previously in the session are encoded and then decoded to generate a query suggestion. Neural models have proven to be state-of-the-art solutions due to their efficiency in generating query suggestions.

Despite these recent developments in query suggestion task; there is still a substantial opportunity to improve. The most significant drawback of this methods is the complexity of current architectures. Current sequence-to-sequence (seq2seq) models use recurrence which results in an increase of computational power and time required to train the models. Correspondingly, this study aims to find a reliable query suggestion

1 Synthetic query suggestions are queries that have never been seen before by the model but whose words are in its vocabulary.
method that uses an encoder-decoder architecture without recurrence. Additionally, this study will compare its behaviour and performance with existing neural models.

### 1.1 Contributions

Aiming to improve current models’ architecture, this thesis proposes a novel neural model that introduces a new encoder-decoder architecture for the query suggestion task. Unlike previous neural models where recurrence is used in their sequence-to-sequence design, the proposed model is based solely on attention. The model uses a Transformer network architecture. This network architecture was first introduced for neural machine translation task, and it was shown to outperform state-of-the-art techniques (Vaswani et al., 2017). Thus, it falls from here the hypothesis that Transformers will achieve good results in query suggestions task. This architecture is less complex since it would reduce the number of sequential operations required which results in a decrease of computational power.

The performance of the proposed model is evaluated by comparing it with two models from the current state-of-the-art query suggestion models. The first model is a Hierarchical Recurrent Encoder-Decoder architecture with Recurrent Neural Networks similar to the one introduced by Sordoni et al. (2015). The architecture of this model introduces context by encoding, not only query level information but also session level information, resulting in a complex hierarchical architecture. The second model resembles the one introduced by Dehghani et al. (2017). A hierarchical seq2seq model as well, but it includes an additional attention mechanism to focus on specific parts of the input which are relevant to reformulate the query. The attention mechanism is used by the model encoder to capture the structure of the session context and handle the scope of the session to infer the next suggestion. The experimentation uses the AOL dataset².

In order to design an improved architecture, several aspects were explored considering previous models’ drawbacks. We formulate four research questions that address more specific aspects and that, as a whole, aim to explore the performance of the new architecture:

**RQ.1** Can a transformer architecture for query suggestion outperform current state-of-the-art query suggestions models?

**RQ.2** Is the Transformer model able to capture query-level information or does it still require a hierarchical encoder structure (Dehghani et al., 2017; Sordoni et al., 2015; Chen et al., 2018) to perform well?

**RQ.3** Will including user’s preference information by including the rank of the documents clicked in a search help improve the performance of the Transformer model?

**RQ.4** Does a Transformer model require information about the order of sequences to perform well in query suggestion task?

In addition, we make a contribution to the Tensor2Tensor (Vaswani
et al., 2018) library by including the model proposed as part of their components. We aim to make accessible our model in Tensor2Tensor so that researchers can reuse or continue working on extending the work presented in this thesis. Moreover, we wish to present the model in a platform that facilitates the reproducibility of the work presented.

1.2 Outline

The remainder of this thesis is structured as follows: Firstly, Chapter 2 details related work in query suggestion task. Subsequently, Chapter 3 covers an in-depth overview of the current state of the art algorithms in the field and their main contributions used as inspiration for the model proposed. This chapter also introduces the Transformer model (Vaswani et al., 2017) which is the base of the method presented in this thesis. In addition, Chapter 4 gives a detailed description of the method suggested, the different modifications examined and the intuition behind the model variations proposed. Chapter 5 describes the experiments conducted in this study. It also introduces a set of evaluation metrics to assess query suggestion models. Then, Chapter 6 shows the results from the experiments performed and presents the findings of the experiments conducted. Finally, Chapter 7 concludes the thesis by describing the contributions made with the proposed method and, also, offering future directions of research.
2 Related Work

Query suggestions try to suggest relevant queries with the aim of increasing the effectiveness of query submission and thus reducing unnecessary search steps. However, to provide appropriate query suggestions, several challenges have to be overcome. Thus, previous scientific research on query suggestions models use different approaches to handle the difficulties in solving this task.

A large group of methods are based on used co-occurrences. The first proposed models use co-occurring terms found in the highly ranked documents retrieved during the search. (Anick and Tipirneni (1999), Jones and Staveley (1999), Xu and Croft (1996)). These approaches struggled in determining which terms were representative of the search. Besides, high-ranked documents might not all be relevant to the queries. Another hurdle found in these methods is that they are unable to identify terms that are conceptually related but do not co-occur in the documents.

To tackle these difficulties, Huang et al. (2003) proposed a method that took leverage of the "wisdom of crowds" by using search engine logs. Search query logs capture the sequence of queries submitted by the users during a session and are an excellent source of information. One can identify query co-occurrence in a session which can be a strong indicator of relatedness and it may indicate what direction the suggestion should follow to satisfy users needs. Huang et al. (2003) tried to find co-occurrences within sessions and used their frequency to rank the query suggestions.

Boldi et al. (2009) proposed a different way to use the query logs. Analysing the logs to extract meaningful patterns and build a query flow graph. Then, using random walk over a query graph, they estimate, within a session, how likely a user would move to a particular query given the previous one.

Another group of methods focused on leveraging the clicked documents based on the assumption that similar queries will share documents selected by users, so it falls from here the idea of using overlapping clicked documents to find related queries. Mei et al. (2008) used a random walk algorithm over a bipartite graph of queries and document clicks with the intuition that this will provide semantic consistency between the suggested query and the original query.

Baeza-Yates et al. (2004) proposed a query clustering framework to group semantically similar queries. First, queries are represented as a term-weight vector of the clicked URLs for the query. Then, queries are grouped using a K-means algorithm. Finally, queries are ranked according to a relevance criterion. Despite being an efficient algorithm, it requires determining the number of clusters ahead of time.

Sadikov et al. (2010) combined the query-flow graphs with document click information to find query suggestions. Formulating the problem as a graph-clustering problem on a Markov graph that models user search
behaviour.

Unlike previously mentioned pair-wise query relations methods, where a single preceding query is used to predict the following query, He et al. (2009) proposed a way to increase the context awareness by considering a variable number of preceding queries to predict the following query. It uses a Variable Memory Markov Model (VMM) to automatically determine the optimal number of previous queries to use as context. The VMM builds a suffix tree to model user’s query sequences to generate query suggestions.

Santos et al. (2013) try to tackle data sparsity problem by considering query suggestions task as a ranking problem. First, they select unique terms present in the query log as candidates. Then the candidates are ranked using learning-to-rank methods taking into account features such as the number of clicks received, the candidate’s relative position in each session and the length of the suggestion in tokens among others.

Following the same lines, Ozertem et al. (2012) rank suggestions taking into account the position of URLs that appear in both search results sets, the original query set and the suggestion set. Moreover, their method uses weighted co-occurrence in the search logs based on the probability of them belonging to the same search.

One big disadvantage with these methods is that they have difficulty dealing with long-tail queries. To overcome this problem, Vahabi et al. (2013) introduced a technique that seeks out orthogonal queries. It comes from the assumption that long-tailed queries are probably quite vague and they will not bring the user close to the information they need. Thus, recommendations with small variations from the original query will not have good performance. Therefore, the approach searches for queries that have no common terms but that are still semantically similar.

Another group of scientific research focus on generating suggestions by leveraging search logs as well as external resources. Szpektor et al. (2011) used a template generation method along with Wordnet. They abstract the general structure of queries using templates (Agarwal et al., 2010). Rules are identified between the templates. These templates are used to generate suggestions falling from the idea that many queries share the same intent even if they have different entities. Nevertheless, using all token boundaries when segmenting the queries to build the templates leads to poor suggestion candidates. To tackle this shortage, Desautels et al. (2014) use Conditional Random Fields to identify non-critical terms in queries segmentation followed by a machine learning process that filters poor suggestions, yielding in better results.

The latest scientific research uses neural network models. It addresses the task of query suggestion as a sequence to sequence problem (Cho et al., 2014). The great advantage of this approach is that unlike co-occurrence methods, it is not prone to data sparsity. It introduces context by encoding previous inputs (sequences of queries) and then decodes it to output a query suggestion. This representation of context embedded in space avoids data sparsity due to similar context data being mapped close to each other.

Additionally these models are able to deal with rare queries. Sordoni

---

1 Orthogonal queries are queries which are related to a user’s queries but that they have a small amount or no common terms.

2 Wordnet is defined as “a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept.” (Kilgarriff, 2000)

3 To build the templates, for each query, they extract all the tokenizations possible by grouping all the terms of a query into sequences of their terms.
et al. (2015) proposed a context-aware method which uses a hierarchical recurrent encoder-decoder architecture to encode the information from the session and decode a sequence of words to output the next query suggestion. However, not all the context of the queries is equally important. Mechanisms such as attention allow focusing on specific parts of the input which are relevant to reformulate the query.

Dehghani et al. (2017) uses a similar architecture but improves the performance by introducing a hierarchical query-aware attention to automatically capture the context in the session. Moreover, it expands the model by adding a pointer network (Vinyals et al., 2015) that allows copying terms from previous queries in a session. This idea comes from the fact that on average 62% of the words used in the session are retained from the previous queries (Sloan et al., 2015). These terms show the information users need, so, they are usually discriminatory terms that help as filters.

Chen et al. (2018) also uses an attention based hierarchical architecture to capture user’s preferences. Unlike, previous neural based methods mentioned, it not only consider user’s current session, they also introduce context information by considering previous user sessions. The model has two neural networks to model long and short-term search history of users. While the short-term recurrent neural network (RNN) captures the queries in a session seen up to that point, the long-term recurrent neural network captures previous sessions of a given user. Besides this, they introduce attention in the hierarchy to capture user’s preference over the different queries using their click behaviour.

These approaches based on neural networks are the most similar to the one presented in this thesis. Yet, this thesis proposes a different encoder-decoder architecture. Unlike these models, where they use recurrence in their sequence to sequence design, the model proposed in this thesis is based solely on attention, resulting in a more parallelisable architecture that requires significantly less time to train.
3 Background

This chapter describes the current state-of-the-art models that served as an inspiration for the model architecture introduced in this thesis. Section 3.1 covers the general design of Sordoni et al. model that served as a baseline and as an inspiration for the hierarchical encoder structure studied in RQ.2. Then, Section 3.2 explains Dehghani et al. model which also has a hierarchical architecture and it introduces an attention mechanism. It will be used for RQ.2 and as a baseline. Lastly, section 3.3 explains the architecture of the transformer model, which forms the basis for the the method proposed.

3.1 Hierarchical Recurrent Encoder-Decoder for Generative Context-Aware Query Suggestion

Sequence-to-sequence models have had great success in reading and generating text. Thus, they can read previous queries in a session to create a query suggestion. However, there is a significant drawback in employing generic seq2seq models directly in query suggestion task. One cannot model information appropriately because seq2seq models consider the input to be a sequence of words without taking into account query level information.

To include query level information, Sordoni et al. (2015) proposed a context-aware seq2seq model with a hierarchical architecture to encode queries issued previously in the session and generate a query suggestion.

The hierarchical recurrent encoder-decoder (HRED) model runs with two parallel processes: the encoding and decoding of queries. First, a Recurrent Neural Network (RNN) encodes the sequence of words from a query seen up to that position into a compact order-sensitive encoding. Then, a second RNN encodes the context per session by learning a summary of the past queries in a session. Finally, a third RNN acts as a decoder to calculate a probability distribution on the space of possible suggestions given the encoded query. Then, this distribution is used to produce the next word in a sequence. This process lasts until the end-of-query symbol appears, resulting in the most likely following query. The architecture of the model is shown in Figure 3.1.

3.1.1 HRED Architecture

The model is comprised of three Gated Recurrent Units (GRU) RNN (Cho et al., 2014): Query-Level Encoder, Session-Level Encoder and Next-Query Decoder.

The Query-Level Encoder GRU receives as input a query \( Q_m = \{w_m,1, ..., w_m,N_m\} \) where words are represented as word embeddings of dimension \( d_e \) and \( N_m \) is the length of the query. Words are processed sequentially updating the GRU’s hidden state. The updates are done according to the following equation (3.1):

\[
d_{m,n} = GRU_{query}(d_{m,n-1}, w_{m,n}), n = 1, ..., N_m. \tag{3.1}
\]
After the entire query is read, the hidden state of the encoder becomes an order-sensitive representation of the query in form of a vector: \( q_m \in \mathbb{R}^{\text{enc}} \).

The Session-Level Encoder GRU repeats the same process as the Query-Level Encoder but using queries instead of words. The input is the sequence of query vectors \( q_1, ..., q_m \). The GRU’s hidden state is an ordered representation of the queries read up to that point: \( s_m \in \mathbb{R}^{\text{session}} \).

The hidden state is updated as follows:

\[
s_m = \text{GRU}_{\text{session}}(s_{m-1}, q_m), \ m = 1, ..., M. \tag{3.2}
\]

Following the idea of introducing context in the encoder by having a summary of the past queries in a session \( s_m \), a hierarchical encoder will be introduced in one of the models proposed, further explained in section 4.5.

The Next-Query Decoder GRU uses the previous queries as context to predict the next query \( Q_m \). The information of the previous queries is transferred to the decoder using \( s_{m-1} \) to initialise the recurrent state as follows:

\[
h_{m,0} = \tanh(D_0 s_{m-1} + b_0) \tag{3.3}
\]

\[
h_{m,n} = \text{GRU}_{\text{dec}}(h_{m,n-1}, w_{m,n}), \ n = 1, ..., N_m. \tag{3.4}
\]

Then, the probability of the next word \( w_{m,n} \) given the previous words and queries on each recurrent state \( d_{m,n-1} \in \mathbb{R}^{\text{dec}} \) is calculated as follows:

\[
P(w_{m,n} = v | w_{m,1:n-1}, Q_{1:m-1}) = \frac{\exp \theta_v^T \gamma(h_{m,n-1}, w_{m,n-1})}{\sum_k \exp \theta_k^T \gamma(h_{m,n-1}, w_{m,n-1})}. \tag{3.5}
\]
Being $o_v$ the embedding of the vocabulary that is used to learn to map related vocabulary words closer in the space to the context information encoded by $\gamma$. The linear transformation $\gamma$ on $d_{m,n-1}$ and the previous word $w_{m,n-1}$ is defined as:

$$\gamma(h_{m,n-1}, w_{m,n-1}) = H_{output}h_{m,n-1} + E_{output}w_{m,n-1} + b_{output}.$$  (3.6)

The $\gamma$ function uses an additional embedding space to map the words it receives as input. Thus, the model uses three embedding spaces: the input space, the vocabulary words space and the previous words space. By doing so, the model increases its expressiveness power. However, achieving as much expressive power as possible comes with a cost. Training three spaces implies longer training time since there are more weights to learn.

### 3.1.2 Learning

The model parameters comprise the parameters of the three GRU functions, the three embedding layers (input, $o_v$ and $\gamma$ function) and the fully connected layer weights to project the session state to the decoder dimensions. The model learns by maximising the log-likelihood of a session $S$ using back-propagation through time (BPTT) algorithm (Rumelhart et al., 1986). The loss of a session is:

$$L(S) = \sum_{m=1}^{M} \log P(Q_m|Q_{1:m-1})$$

$$= \sum_{m=1}^{M} \sum_{n=1}^{N} \log P(w_{m,n} = v|w_{m,1:n-1}, Q_{1:m-1}).$$  (3.7)

### 3.1.3 Query Generation

The generation of query suggestions is done through a beam-search algorithm. On each iteration, a set of $x$ words with the higher probability, calculated using equation (3.5), are considered as candidates. Then, each of them is extended using the next decoder recurrent state, computed using the previous state and the sampled word. The process repeats until the end-of-query symbol is sampled resulting in a new complete query.

### 3.1.4 Baseline

A Hierarchical Recurrent Encoder-Decoder architecture resembling the HRED architecture will be used as a baseline for this thesis (HRED-LSTM). Instead of using Gated Recurrent Units (GRU) RNNs, the baseline will be comprised of three Long Short-term memory (LSTM) RNNs.

Long-short term memory (LSTM) recurrent units were introduced as a way to mitigate the problem of vanishing and exploding gradients during learning. Gated Recurrent Units (GRU) were introduced afterwards with the same goal as LSTM: track long-term dependencies efficiently while avoiding vanishing and exploding gradients.

The LSTM unit tracks long term dependencies using the input, forget, and output gates. While the input gate controls how much of the new
information is kept, the forget gate regulates how much of the existing memory is forgotten. The output gate controls the output by deciding how much information of the new cell state is used. Similarly, GRU unit has two gates to control the unit’s memory, the reset gate and the update gate. The reset gate controls how much of the previous state is forgotten depending on the previous activation and the next candidate activation. The update gate decides how much of the candidate activation is used to update the cell state. Figure 3.2 illustrates a Vanilla RNN, an LSTM and a GRU unit.

Both recurrent units have the ability to save states from previous activations instead of substituting the entire activation like Vanilla RNNs. Thus, they have similar performance (Chung et al., 2014) and using LSTM will not have a significant change in the model’s performance.

Therefore, the new query level encoder will process queries as follows:

\[ \text{LSTM}_{\text{query}}(d_{m,n-1}, w_{m,n}), n = 1, ..., N_m. \quad (3.8) \]

Then the Session Level will repeat the process encoding the sequence of query vectors:

\[ \text{LSTM}_{\text{session}}(s_{m,n-1}, q_{m,n}), M = 1, ..., M. \quad (3.9) \]

and the Next-Query Decoder unit will decode the suggestions as follows:

\[ \text{LSTM}_{\text{dec}}(h_{m,n-1}, w_{m,n}), n = 1, ..., N_m. \quad (3.10) \]

In practice it can be implemented as follows:

Given a session \( S = \{Q_1, ..., Q_L\} \), at a given timestep \( t \), the query level encoder encodes up to that point the words in a query from the session \( Q_l = \{w_{l,1}, ..., w_{l,N_l}\} \) into word embeddings of dimension \( d_e = 128 \). Then, the session level information is encoded into a context vector \( c_t \) using the sequence of queries up to point \( t \). The context vector is calculated as follows:

\[ c_t = \sum_{i=1}^{t} q_i, \quad (3.11) \]

where \( q_i \) is the summary vector of the \( i \)-th query:

\[ q_i = \sum_{j=1}^{N_i} w_{i,j}, \quad (3.12) \]
Then the context vector, which is the session level encoding, is added to the query level encoding, it results in the following encoder representation for a given word in position $k$ in query $l$:

$$x_{l,k} = w_{l,k} + c_i. \quad (3.13)$$

### 3.2 Learning to Attend, Copy, and Generate for Session-Based Query Suggestion

As explained in the section before, generic seq2seq models are unable to generate context-aware suggestions because they do not capture query level information. Another hurdle with generic word-based seq2seq models is that they are less likely to produce queries with very low-frequency terms from the vocabulary. Moreover, they cannot deal with out-of-vocabulary words (OOV).

Different patterns can be observed from the way users change preceding queries during the query sessions. Some of these patterns include term addition, removal, and retention (Eickhoff et al., 2014; Sloan et al., 2015). Term retention is one of the most relevant query reformulation patterns. As outlined in (Dehghani et al., 2017): on average 62% of the terms in a query are retained from previous queries (Sloan et al., 2015). Also, more than 39% of the users repeat words from their previous query (Jiang et al., 2014). According to the AOL query log statistics (Pass et al., 2006), more than 67% of the words retained in a user’s session are from the 10% less frequent words in the vocabulary. A model that is not able to generate out-of-vocabulary words (OOV) will not be able to model term retention. Thus, it will not form query suggestions adequately.

To address the issue of not being able to generate context-aware suggestions, Dehghani et al. (2017) propose a seq2seq model with query-aware attention mechanism. By using a hierarchical attention mechanism, the model can capture context. Furthermore, by doing so with an attention mechanism, the model can determine what aspects of the context are relevant and attend over those to generate the next query suggestion.

Moreover, to solve the shortage of not being able to model term retention, Dehghani et al. (2017) introduced a copy mechanism in their model. When the model is generating the query suggestion, the copy component provides terms from previous queries in the session. This allows the model to deal with OOVs and therefore model term retention.

The seq2seq model with query-aware attention mechanism (ACG) is RNN-based encoder-decoder. The model runs with two parallel processes: the encoding and decoding of queries. First, a bidirectional recurrent neural network (RNN) encodes the words from a query seen up to that point. Also, a bidirectional recurrent neural network (RNN) encodes the queries seen up to that point. The encoded words and queries are summarised into a context vector. Then, a unidirectional RNN decodes the context vector into the target sequence. It does so by using an output projection layer to compute the probability distribution over the vocabulary to generate the next word. Additionally, a Pointer Network (Vinyals et al., 2015) works as a copy mechanism and calculates
a probability distribution over the input sequence to copy the next word. Then, these probabilities distributions are used to produce the next word in a sequence. The model decides whether to copy or generate a term in each time step. This process lasts until the end-of-query symbol appears resulting in the most likely following query. Figure 3.3 shows the architecture of the model.

\[ q_j = [q_{f,j}; q_{b,j}] \]  \hspace{1cm} (3.14)

### 3.2.1 ACG Architecture

The model extends Bahdanau et al. (2014) seq2seq model with a hierarchical attention architecture. The original implementation was done in a Neural Machine Translation model for which the inputs were the words of a given text. The implemented attention mechanism was used to pay attention to the entire text and determine which words were important for the translation. In this case, attention is used in a hierarchy to focus at different levels: pay attention over the words in a query and queries in a session.

The model is comprised of two Bidirectional RNNs: a word-level encoder; a query-level encoder, a Unidirectional RNN that works as a decoder and a Pointer Network that functions as a copy mechanism.

The word-level bidirectional RNN encoder receives as input a query \( Q_m = \{w_{m,1}, \ldots, w_{m,N_m}\} \) where words are represented as word embeddings of dimension \( d_e \) and \( N_m \) is the length of the query. Words are processed sequentially from left to right in the forward pass creating a sequence of RNN’s hidden states \( q_f \). Then words are processed in the reverse direction obtaining another sequence of hidden states \( q_b \). The forward and backward states for each time step are concatenated to create the encoder hidden states.

Figure 3.3: Learning to Attend, Copy, and Generate for Session-Based Query Suggestion model architecture.
The annotation \( q_j \in \mathbb{R}^{2mc} \) is a summary of the preceding and following words in order to capture the context for each word. Then, the encoded query is summarised using a function \( \Phi \) to generate a fixed length context vector \( c = \Phi(q_1, q_2, ..., q_m) \).

The query-level Bidirectional RNN encoder repeats the same process as the word-level Bidirectional RNN encoder but using queries instead of words. The input is the sequence of query vectors \( q_1, ..., q_m \) concatenated, each one followed by a special token \(</q>\). The encoder hidden states are again created by concatenating the forward \( g_f \) and backward \( g_b \) states for each time step.

\[
g_j = [g_{f,j}; g_{b,j}]. \quad (3.15)
\]

One of the models proposed in this thesis, also introduces a hierarchical encoder to include context. This is further explained in section 4.5.

The decoder is a Unidirectional RNN with hidden states \( h_t \) which uses a context vector on each time step to produce a target sequence. During each time step, a score \( \text{score}(h_{t-1}, x_t) \) is calculated using the hidden state of the decoder \( h_{t-1} \) to determine how well each annotation \( x_t \) in the source sequence matches the target before emitting output \( t \). Then, this score is used to calculate the attention weights \( a_t \) normalised by a softmax.

\[
a_{t,i} = \frac{\exp(\text{score}(h_{t-1}, x_i))}{\sum_{j=1}^{N} \exp(\text{score}(h_{t-1}, x_j))} \quad (3.16)
\]

The attention weights are calculated for the word level \( q \) annotations as well as for the query level annotations \( g \). To get the final query-aware attention weights, the word-level attention weights are multiplied with their corresponding query-level attention weight and normalised as follows:

\[
a_{t,i} = \frac{a_{t,i}^q a_{t,i}^g}{\sum_{t'=1}^{N} a_{t',i}^q a_{t',i}^g} \quad (3.17)
\]

The context vector for output \( i \) is calculated as follows:

\[
c_t = \sum_{i=1}^{N} a_{t,i} x_i \quad (3.18)
\]

Being \( N \) the number of tokens up to time \( i \). Giving a context vector \( c_i \) per output \( i \).

The hidden states in the decoder are calculated with the previous state \( h_{t-1} \), the context vector \( c_t \) and the previous output \( y_{t-1} \):

\[
h_{m,t} = \text{RNN}_{\text{dec}}(h_{m,t-1}, c_t, y_{t-1}) \quad (3.19)
\]

Then, an output layer is used to compute the probability over the vocabulary, given the previous words and queries on each recurrent state.

\[
p(y_{t}|y < t, X, \text{generate}) = f_y(h_t) \quad (3.20)
\]

However, the output projection layer will compute the probabilities for words in the vocabulary. To handle OOV words a Pointer Network is
introduced (Vinyals et al., 2015). The copy mechanism (Pointer Network) calculates the probability distribution over the input sequence using the encoder hidden states as follows:

\[
p(y_t|y < t, X, \text{copy}) = \frac{\exp(score(h_{t-1}, x_i)^p)}{\sum_{j=1}^{N} \exp(score(h_{t-1}, x_j)^p)}
\] (3.21)

Where \(x_0\) is the embedding of the unknown token \(<\text{UNK}>\), this is done to handle words that have to be generated instead of copied. Then, on each time step, a switch gate decides whether to generate or copy a word. Being \(w\) a weight vector and \(\sigma\) a sigmoid function the probabilities are calculated as follows:

\[
p(\text{copy}) = \sigma(w^{T}h_t)
\] (3.22)

\[
p(\text{generate}) = 1 - p(\text{copy})
\] (3.23)

The switch gate will favour the copy mechanism to copy as much as possible from the input and then let the generator create the rest of the words.

### 3.2.2 Learning

The model parameters comprise the parameters of the generator, the copier and the switch gate. The model uses back-propagation through time (BPTT) algorithm (Rumelhart et al., 1986). It has three different losses: One is the loss of the generator, which is the average cross entropy between the probability distribution \(p\) computed and the one-hot encoding of the target word \(q\) (target probability distribution).

\[
\text{loss}_{\text{generate}} = \frac{1}{|V|} H(p, q) = \frac{1}{|V|} \sum_{v \in V} p_v \log q_v
\] (3.24)

Where \(V\) is the vocabulary and \(|V|\) is its size.

Another one is the Pointer Network loss, which is the cross entropy over the probability distribution computed \(p\) and the one-hot encoding of the target word \(q\) averaged over the input length:

\[
\text{loss}_{\text{copy}} = \frac{1}{|X|} H(p, q) = \frac{1}{|X|} \sum_{x \in X} p_x \log q_x
\] (3.25)

Where \(X\) is the input sequence and \(|X|\) is its size.

Finally, the loss for the switch gate is:

\[
\text{loss}_{\text{switch}} = p(\text{copy}) - t_{\text{switch}}
\] (3.26)

To favour the copy mechanism over the generator and to avoid producing \(<\text{UNK}>\) tokens, \(t_{\text{switch}}\) is a variable that follows the following set of rules:

- Copy target is \(<\text{UNK}>\) and generator target is not \(<\text{OOV}>\), switch gate should choose the generator. \(t_{\text{switch}} = 0\)
- Copy target is not \(<\text{UNK}>\) and generator target is \(<\text{OOV}>\), switch gate should choose the copier. \(t_{\text{switch}} = 1\)
• Copy target is <UNK> and generator target is <OOV>, switch gate should choose the generator. $t_{\text{switch}} = 0$

• Copy target is not <UNK> and generator target is not <OOV>, switch gate should choose the copier. $t_{\text{switch}} = 1$

The update of parameters in the backward pass of the back-propagation step is done in three steps. First, the gradient for the copy loss ($\text{loss}_{\text{copy}}$) is calculated, and the copy mechanism parameters are updated. Then, the gradients for the generator loss ($\text{loss}_{\text{generate}}$) are determined, and the parameters of the generator are updated while the switch gate and the copy mechanism are frizzed. Finally, the switch parameters are updated propagating the gradients calculated with the switch loss ($\text{loss}_{\text{switch}}$).

### 3.2.3 Query Generation

The generation of query suggestions is done through a beam-search algorithm. On each iteration, a set of $x$ words with the higher probability are selected as candidates. For each candidate the top $x$ words are selected. Then, from those the $x$ most likely sequences are selected to produce the next word.

### 3.2.4 Baseline

Given the complexity of the model with the copy mechanism this thesis proposes a simplified version by removing the parts that are not necessary to compare the performance of traditional encoder-decoder architectures with transformer architectures. A Hierarchical Recurrent Encoder-Decoder with attention is proposed (HREDA-LSTM). The model will be comprised of two Bidirectional RNNs for the hierarchical encoder and a unidirectional RNN that works as the decoder. However, to do a fair comparison we remove the pointer network which the model proposed does not have. Consequently, the model will be trained only with the generator loss ($\text{loss}_{\text{generate}}$).

In practice the architecture of the model will be implemented as it was detailed in 3.1.4.

### 3.3 Transformer Model

RNN (Cho et al., 2014) have proven to be the choice in state-of-the-art sequence models. As seen in (Sordoni et al., 2015; Dehghani et al., 2017) models, the most competitive models in query reformulation task use RNNs in an encoder-decoder architecture.

The general process in encoder-decoder architectures is: the encoder receives as input a sequence of tokens and it maps it into an encoding sequence of continuous representations. Then, at each time step, the decoder generates an output from the encoding. The decoder uses the previous outputs as well as the encoded sequence up to that point to produce an output. The inherently sequential nature of this models restrains the possibility of using parallelisation within training examples. Being able to split training examples into several tasks to process independently, and combine the results at the end is essential when sequences are long since batching across instances is restricted due to
memory constraints.

Vaswani et al. (2017) proposed the Transformer model: a network design that removes recurrence and instead relies solely on attention mechanisms. By eliminating the recurrence, the number of sequential operations is reduced, and the computational complexity is decreased.

Notwithstanding the fact that the Transformer removes the recurrence mechanism, it uses positional embeddings, first introduced by Gehring et al. (2017), to maintain the order of the sequence. Thus, it is still able to encode sequences of continuous representations. Moreover, the Transformer still uses an "encoder-decoder" architecture that has shown to be useful in query suggestions; the Transformer matches this architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder. Figure 3.4 shows the model architecture.

Vaswani et al. (2017) applied the Transformer encoder-decoder architecture to neural machine translation task. This architecture not only reduced the complexity and training time but outperformed the state-of-the-art. Transformers have shown good results in other sequence modelling tasks, for example, in Relation Extraction task in the biomedical domain and Image generation on ImageNet.

Following this line, we can use a Transformer model for query suggestions. Given that it has shown to give good results in other seq2seq tasks, it falls from here the hypothesis that a Transformer network could be the way of simplifying the current query suggestions models without compromising its performance.

### 3.3.1 Transformer architecture

The model is comprised of an encoder and a decoder, each with six identical layers and each is composed of sub-layers. All sub-layers and embedding layers are of dimension \( d_{model} = 512 \).

Each encoder’s layer is composed of two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Around each sub-layer there is a residual connection and the following normalisation:

\[
\text{Norm}_{layer} = (x + \text{Sublayer}(x))
\]

Where function \( \text{Sublayer}(x) \) is the function implemented by the sub-layer.

Similar to the encoder, three sub-layers form each decoder layer. Besides the two layers found in the encoder layer, there is multi-head attention over the output of the encoder stack. Residual connections and normalisation are also applied around each sub-layer. To avoid attending to consecutive positions in the decoder stack and also to ensure the auto-regressive property, a mask is applied in the self-attention sub-layer.

As mentioned above, the model doesn’t contain recurrence therefore, it uses positional encodings to incorporate information about the order of the sequence. The positional encodings of dimension \( d_{model} \) are added to the input embeddings at the bottom of the encoder and decoder stacks.

The positional embeddings are created with sine and cosine functions.
of different frequencies:

\[ PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \]  

(3.28)

\[ PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \]  

(3.29)

Where pos is the position, and i is the dimension. Using positional embedding in the model proposed will be further discussed in 4.3. It is also used to study RQ.4.

\[ \text{Attention}: \]

Broadly speaking attention defines how much of each input state affects each output. To determine this, a score is calculated given for each input given an output. The score, based on context, weighs each value and combines them to determine to affect the output selectively.

The model uses a Scaled Dot-Product Attention, depict in Figure 3.5(left). Given a set of queries Q a set of keys K and their values V in matrix form, attention is calculated as follows:

\[ \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \]  

(3.30)

Where \( d_k \) is the dimension of the keys, and \( \frac{1}{\sqrt{d_k}} \) is used as a scaling factor to avoid vanishing gradients.

Figure 3.4: Transformer model architecture
Instead of performing a single Scaled Dot-Product Attention function, the model uses Multi-Head Attention: queries, keys and values are linearly projected \( h = 8 \) times with different linear projections. The model simultaneously attends to information from various representation sub-spaces at several positions. Multi-Head Attention is calculated as follows:

\[
\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O
\]  

\[
\text{head}_i = \text{Attention}(QW^Q_i, KW^K_i, VW^V_i)
\]

Where the dimensions used are \( d_k = d_v = \frac{d_{\text{model}}}{h} = 64 \) and the parameter matrices are: \( W^Q_i \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^K_i \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^V_i \in \mathbb{R}^{d_{\text{model}} \times d_v} \)

Figure 3.5 (right) shows the Multi-Head Attention.

**Position-wise Feed-Forward Networks:**
The Position-wise Feed-Forward Network used in the model is formed by two linear transformations with a ReLU activation function. Different parameters are used on each layer. It can be seen as two convolutions with kernel size 1. Given that the model has dimension \( d_{\text{model}} = 512 \), the dimensions inside the layer is \( d_{ff} = 2048 \).

**Embeddings and Softmax:**
The Transformer model transforms the input and output tokens into vectors of dimension \( d_{\text{model}} \) using an embedding layer. To predict the most likely output, vectors are transformed linearly and then passed through a softmax function. Weights in the input embedding layer, output embedding layer, and pre-softmax linear transformation are shared.

**3.3.2 Learning**
Unlike recurrent seq2seq models, were the backpropagation through all of the RNN states because they all happen in sequence. In Transformer architecture, all the training is done per example, that means that the output of one single token is one sample and the computation of the backpropagation is done for that single step. Thus there is no multi-step backpropagation like in RNN. Transformer learning is done using Kullback-Leibler divergence loss.
4 Query Suggestion Transformer

This chapter introduces a new neural model for the query suggestion task, building upon the state-of-the-art models described in Chapter 2, resembling those described in Chapter 3 and using a Transformer architecture outlined in section 3.3. First, section 4.2 presents the Vanilla transformer used now for query suggestions. Then, section 4.3 proposes a modification of this Vanilla transformer by having a non-ordered approach that eliminates the relative order between words in queries. Furthermore, section 4.4 proposes a structured transformer with a new input representation that implements an ordered structure, where the order is kept within the session but without having order between words in a query. Using this structured transformer, section 4.5 introduces query level information by implementing a hierarchical Transformer. Similarly, using the structured transformer, section 4.6 introduces, in this case, user’s intent information to it. These models will be empirically evaluated, and the results will be discussed in the remainder of this thesis.

4.1 Query Suggestion Data for Transformers

To use Transformer models in a query suggestion task, we defined an input representation to create the training examples. Query sessions were transformed in examples as follows: given a session with \( n \) queries, \( n - 1 \) examples were created. On each example, the input is composed by the queries up to that point in the session and the target is the following query in the session. An example of the data examples generation can be found below:

<table>
<thead>
<tr>
<th>Data examples generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Session:</strong></td>
</tr>
<tr>
<td>1: Italian Cuisine.</td>
</tr>
<tr>
<td>2: Pasta Carbonara Recipe.</td>
</tr>
<tr>
<td>3: Italian Pasta Recipe</td>
</tr>
<tr>
<td><strong>Examples created:</strong></td>
</tr>
<tr>
<td>1: <strong>Input:</strong> Italian Cuisine <strong>Target:</strong> Pasta Carbonara Recipe.</td>
</tr>
<tr>
<td>2: <strong>Input:</strong> Italian Cuisine Pasta Carbonara Recipe <strong>Target:</strong> Italian Pasta Recipe.</td>
</tr>
</tbody>
</table>

4.2 Vanilla Transformer

As explained in section 3.3, a Transformer (Vaswani et al., 2017) is a network that uses an "encoder-decoder" architecture based solely on attention. Given that neural models for query suggestions use a sequence to sequence architecture, one can naturally adapt this model for query suggestions task. The sessions that query suggestion tasks have are transformed in examples as it was previously detailed in 4.1. Using this input representation in the examples allows us to encode all the queries
up to that point in the session as an example. However, because the transformer learning is done per example, the session information is lost and the input representation does not delimit sessions. Once the sessions are processed as examples, they are contextually encoded with word embeddings.

The proposed model takes a sequence of \( n \) word embeddings. To model position information, a positional embedding (illustrated in Figure 4.1) is added to each input word embedding. This results in the following input representation: for a word \( x_i \) is: \( x_i = q_i + p_i \). Where \( q_i \) is the word embedding for \( x_i \) and \( p_i \) is the positional embedding for the \( i \)th position.

Examples are tokenized using sub-word representations. This allows the model to predict an output when it encounters rare words. A vocabulary of sub-word tokens is constructed with the train data-set. First it selects single characters. Then, the algorithm iteratively combines the most frequent co-occurring tokens to create vocabulary tokens. This process continues until the vocabulary reaches the size defined.

4.3 No Positional Embedding Transformer

Recurrent encoder-decoder architectures, encode inputs going from one input in the sequence to the next. However, the Transformer model doesn’t contain recurrence. As shown in figure 4.2, the Transformer model uses multihead attention to encode the input embeddings, when doing so, it attends in a forward and backward manner so the order in the input sequence is lost. Because of this, it relies on positional embeddings to introduce information about the order of the sequence. For seq2seq tasks, modelling order information of inputs in a sequence can be crucial for the model to learn. However, in the particular case of query suggestion task, words function as key words to filter information and find the documents required. Following this intuition, one can assume that the order does not affect the performance of the model substantially. For example searching Italian Cuisine or Cuisine Italian both will return the same output. Typing Italian first or after Cuisine will not have significant impact on the result as long as both have the same filter terms: Italian, Cuisine. In other words, we can assume that the modelling the syntactically correct order of words in a query is not significant.

Moreover, the data preprocessing (4.1) used to run the vanilla transformer (4.2), follows this intuition: data examples lack of session and query structure. Unlike previous query suggestion models, tokens to delimit queries within a session are not used. The input can be seen as a bag of words with words seen up to that point in the session.

Because of the aforementioned, one can assume that using positional embeddings with this input structure will not have any improvement in performance. Therefore, it is worth removing them from the Vanilla Transformer to obtain a non-positional embedding Transformer.

4.4 Structured Input Transformer

The previous model in section 4.3 assumes that in query suggestion task, words function as keywords to filter information and thus, the relative
order is disregarded and the inputs lack of order. However, despite the fact that word order inside a query might not affect the performance of the model substantially, this is not the case for the order of queries inside a session. By looking at how the queries in a session change, one can lead the direction of the search by either suggesting queries that dive deeper into the current search direction or by suggesting a change of direction into a different search space. In other words, the order of queries within a session is an indicator for generalisation or specialisation and therefore, it cannot be overlooked. Therefore, a new input structure is proposed, different to the flat structure used in previous models.

Tensor2Tensor\(^2\), the framework on which the Transformer is built upon allows to use two-dimensional inputs for models that use images as input. Hence, taking advantage of this, one can use bi-dimensional structures to model ordered queries within sessions.

This results in a new input representation. A session \( S = \{Q_1, ..., Q_L\} \) is represented as the set of queries in it, which in turn, is represented

\( S = \{Q_1, ..., Q_L\} \) is represented as the set of queries in it, which in turn, is represented
as the set of words in it $Q_l = \{w_{l,1}, ..., w_{l,N_l}\}$. The words are represented with word embeddings of dimension $d_e = 512$. The length of the query is $N_l = 10$ and the length of the session is $L = 10$. The aforementioned structure will result in a semi-ordered structure for queries within a session. The bag of words representation for words in a query is kept since we have removed the positional embeddings. Figure 4.3 shows the resulting input structure.

One disadvantage of using this new input representation is that inputs lengths need to be fixed. However, looking at the data exploration mentioned in section 5.1 one can see that 99% of the queries range from length 2 to 10 and in like manner 99% of sessions range from 2 to 10 queries. Consequently, building examples with a query length of 10 and a session length of 10 will not affect the majority of examples. Queries longer than this were truncated and the ones shorter were padded. The same criteria were used to fix the sessions’ length.

4.5 Structured Input Transformer With Hierarchical Architecture

In the previous section we introduced a Transformer model that is now able to model the order of sequences of queries in sessions. However, as explained before, generic encoder-decoder architectures are not able to model context aware suggestions because they capture word-level information but they do not capture query-level information. To address this issue, previous models (Sordoni et al., 2015; Dehghani et al., 2017) introduce a hierarchical encoder in their architecture. Following the same lines as previous models we introduce a hierarchy in the transformer encoder to encode the previously issued queries in the session.

Using the structured input transformer, we introduce hierarchy in the encoder. Resulting in the following:

First, given a session $S = \{Q_1, ..., Q_L\}$, at a given timestep $t$, the query level encoder encodes the words in a query from the session $Q_l = \{w_{l,1}, ..., w_{l,N_l}\}$ into word embeddings of dimension $d_e = 128$ up to that point. Then, the session level encoder encodes the sequence of queries up to point $t$ into a context vector $c_t$ calculated as follows:

$$c_t = \sum_{i=1}^{t} q_i,$$

where $q_i$ is the summary vector of the $i$th-query:

$$q_i = \sum_{j=1}^{N_l} w_{i,j}.$$

Then the context vector, which is the session level encoding, is added to the query level encoding, resulting in the following encoder representation for a given word in position $k$ in query $l$:

$$x_{l,k} = w_{l,k} + c_t.$$

Since the transformer training is done per example, it does not require to specify the session boundaries, they are automatically detected with
the input structure used. Thus, by employing this hierarchical encoder there is an ability to automatically model the context in a session. Figure 4.4 illustrates the input encoding of the model proposed.

4.6 Structured Input Transformer With Clicks

The transformer models proposed so far base their search context in all the previous queries generated by the user up to that point in a session. However, finding user’s preference among those previous queries might introduce more search context information in the model which in turn could result in generating better query suggestions. Assuming that queries within a session might show different aspects of the user’s search intent, we can use the rank of the documents clicked on the search engine result page that each query had to characterise the preference the user has for each query and capture user’s search intent. The fact that documents from the results page were clicked in a certain query could indicate that that query is closer to retrieving the information the user’s needs than queries without clicks.

These results in the following changes to the structured input transformer. The model’s input is comprised of session and the user’s intent for that session. A session $S = \{Q_1, ..., Q_L\}$ is represented as the set of queries in it, which in turn, are represented as the set of words in them $Q_l = \{w_{l,1}, ..., w_{l,N_l}\}$. The words are represented as word embeddings of dimension $d_e = 128$ and $N_l = 10$ is the length of the query and $L = 10$ is the length of the session. The user’s search intent information for a session is comprised by the click information $R = \{r_1, ..., r_L\}$ where for each query there is a list of ranks clicked by the user in that. The list of ranks of a query is comprised by the set of ranked positions the user clicked, thus, $r_l = \{c_{l,1}, ..., c_{l,M}\}$. The rank positions are represented as embeddings of dimension $d_e = 128$ and $M = 20$ is the length of the click list.

The aforementioned structure will result in two semi-ordered structures. The first one is an ordered structure for queries within a session but a bag of words representation for words in a query, as in previous models. The second one is an ordered structure for user’s search intent information within a session but a bag of words representation for clicked document ranks in a query. Figure 4.5 shows the resulting input structure.
Two vocabularies of tokens are constructed. One to encode the queries with the train data-set. This vocabulary iteratively selects the most frequent words in the training set. This process continues until the vocabulary reaches the size defined. Another vocabulary is constructed to encode the user’s search intent with a list of the possible click rankings, values ranging from 0 to 20.

The model will then contextually encode input embeddings with the following representation for a given word in position $k$ in query $l$:

$$x_{l,k} = w_{l,k} + e_{m,j} \quad (4.4)$$

where $e_l$ is the sum of clicked document ranks embeddings in $L_j$ and of dimension $d_e = 128$ as:

$$e_l = \sum_{i=1}^{M} c_{l,i} \quad (4.5)$$

Figure 4.6 illustrates the final encoding of inputs.
5 Experimental Setup

Several experiments were performed in this thesis to answer the four research questions stated in chapter 1. The following sections serve to describe the experiments conducted. Section 5.1 details the data used in the experiments. Section 5.2 describes the metrics used to evaluate the models. Then, section 5.3 discusses experiments carried out, the models used and intuition behind choosing them. Finally, section 5.4 explains the implementation details.

5.1 Dataset

The dataset used in this thesis is composed of queries from the AOL search log. This is a publicly available search log large used for current state-of-the-art models (Sordoni et al. (2015), Dehghani et al. (2017), Chen et al. (2018)). Moreover, it is large enough to train the models proposed in this thesis. The dataset is composed of query searches from 1st of March 2006 to 31st May 2006. It has 16,946,938 queries submitted by 657,426 unique users.

5.1.1 Dataset Analysis

First, an analysis of the dataset was conducted to understand how users reformulate their queries throughout the session. By gaining insight of the dataset, it is possible to make better decisions regarding the model’s design - i.e. adjusting the model parameters such as minimum session length\(^1\), the maximum length of a query\(^2\), etc.

The data was divided into sessions. Each session is considered finished once the user has been idle for 30 minutes (Jansen Bernard et al., 2007). Non-alphanumeric characters, single-query sessions, and consecutive repeated queries are eliminated from the dataset. Once removed, there are a total of 2,905,800 sessions and a total of 7,136,129 queries from which the analysis was conducted.

\(^1\) Session length is the number of queries submitted in the session.
\(^2\) Query length is the number of words in a query.

The session with maximum length is 28 queries. Figure 5.1 presents the frequency of the different session lengths. This Figure shows that 99% of sessions are less than 10 queries long. Therefore, the maximum

![Figure 5.1: Frequency of the different session lengths](image_url)
session length for the SIT (4.4) and its variations (4.6, 4.5) was set as 10 and sessions longer than this were truncated.

Queries range from length 1 to 245 terms. The average length is 1.58. Figure 5.2 presents the different query length frequencies. As shown, 99% percent of queries range from 2 to 10 queries. Therefore, the maximum query length for the Structured Input Transformer (4.4) and its variations (4.6, 4.5) was set as 10 and queries longer than this were truncated3.

5.1.2 Data preprocessing

Data was preprocessed by eliminating non-alphanumeric characters and lower casing. The minimum length of sessions was set at 4 queries long. Thus, any session with less than 4 queries were removed. Additionally, the minimum length of a query was set to 4 and therefore any query with less words was removed.

Furthermore, a large number of sessions repeat the same query consecutive (2,298,108 queries). Consecutive queries repetition prevents a fair comparison between the performance of the models; some might have an advantage over the rest if consecutive repeated queries are not removed. Therefore, the consecutive repeats from the same query were removed and replaced by a single query. The appearance of the same query several times throughout the session was allowed only when they did not appear next to each other.

The data was divided into sessions by considering the end of one after 30 minutes of the user being idle (Jansen Bernard et al., 2007). Sessions were sorted by the query time-stamp. A three moth period of data was further partitioned into training, validation and test sets: two months of data were taken as the training set, queries from 1/03/2006 until 30/04/2006. Two weeks of the time frame were used as the validation set, queries are ranging from the 01/05/2006 until 15/05/2006. Finally, the last two weeks were taken as a test set, queries from 15/05/2006 until 31/05/2006.

Queries will be input as embeddings, therefore a vocabulary needs to be compiled with all the words that the model will be able to use. Choosing the size of the vocabulary has a trade-off. On one hand, if the vocabulary is too small, examples will contain a lot of OOV words. As a result, the model looses expressiveness and it is not able to learn. On the other hand, if the size of the vocabulary is too big, the training will

---

3See appendix A for further data exploration

---
be too slow. The vocabulary was built based on the frequency of words in the dataset; it was cut up to the $2^{20} \sim 100K$ more frequent words, the rest of words were mapped to the unknown token <UNK>. The size of the vocabulary was established empirically after studying how the models reacted to different sizes.

5.2 Evaluation Metrics

In order to evaluate the model’s performance, we could evaluate the quality of the suggestions generated. This would require to analyse the set of all possible queries that retrieve the information needed by the user. Unsurprisingly, the amount of word combinations possible to achieve this set is infinite. Requiring at least a sample of probably correct suggestions for each target. Still, in practice, only one or a few of the possible queries can be observed making it hard to make a proper assessment.

Nonetheless, to evaluate the performance to some extent, this study could evaluate how similar the generated query is to the target query taken from the ground truth. During generation, the model tries to generate the semantically most probable terms for the query suggestion based on the learned representations. Thus, we could measure how well the most probable query suggestion matches the target query. More precisely, this would be measuring the Output Overlap: the number of overlapping terms between the suggestion and the target query normalised by the number of unique words in the query suggestion; and the Target Overlap: the number of overlapping terms between the suggestion and the target query normalised by the number of unique terms in the target query.

\[
\text{OutputOverlap} = \frac{\text{set of words in suggestion} \cap \text{set of words in target}}{\text{set of words in suggestion}}
\]

(5.1)

\[
\text{TargetOverlap} = \frac{\text{set of words in suggestion} \cap \text{set of words in target}}{\text{set of words in target}}
\]

(5.2)

To determine if certain query suggestion is relevant, several similarity metrics may be applied. This thesis will look at the matching words between the query suggestion and the target query bag of words representations (BoW), thus the order will not be considered. Non-ordered metrics have been established following the intuition that in the particular case of query suggestions, the order of words does not substantially affect the outcome, since words work more like keys to filter information. A more detailed explanation can be found in section 4.3.

Conversely, suggestions need to be semantically similar to the target, and even though a particular query does not match to the test query, it can still be relevant if they are semantically related. For this reason, the semantic similarity is assessed as a quality measure. To evaluate semantic similarity cosine similarity and Word Mover’s Distance (WMD) will be computed.

---

Cosine similarity measures the similarity between two vectors. In this case, the vectors correspond to the means of the word embeddings of the output query and the target query. It measures the cosine of the angle between them. If the cosine similarity is 1, queries have the same orientation, thus they are maximally "similar". On the contrary if the cosine similarity is 0, queries are maximally "dissimilar."

WMD measures the dissimilarity between two queries. It calculates the minimum distance that each embedded word in a query needs to "move" to obtain the embedded words of the test query. Queries are represented as weighted point clouds of embedded words, and the distance is calculated as the minimum cumulative Euclidean distance from words in one query to the point cloud of the other query. As shown in figure 5.3.

This metric leverages from the results of Mikolov et al. (2013) which have shown that semantic relationships are often maintained in word vector operations. However, to measure similarity, word2vec embeddings are used instead of the embeddings learned by the models. Using the models' learned embedding space is avoided because if models work as expected, word vectors will be close in space and this metric would result in a sanity check instead of an assessment.

5.2.1 Soft Evaluation Metrics

One of the most challenging aspects of query suggestion task is being able to understand which parts of the session context are useful to provide appropriate suggestions. It is challenging due to the noise that appears in the context. Noise can happen when users abruptly change the queries topics within a session. In these cases, the assumption of detecting session boundaries after 30 minutes of idle time is not accurate. Another type of noise is caused when users submit navigational queries during a session, like -i.e. google, ebay, MySpace.

Predicting noisy queries is quite difficult because the context provided is not enough to produce an output. Besides, models should be robust to noise and they should neglect it. Therefore, cases where the target query is noisy should not be taken into account when evaluating the quality of the suggestions. As an attempt of removing noisy target queries from the evaluation, a soft evaluation is proposed: models will be assessed only on queries were the target query has more than one term from the input query. The soft evaluation proposed will be done using the
5.3 Experiments

Research question RQ.1 directly entails comparison with state-of-the-art query suggestions algorithms. As a baseline for this research question the following models will be tested:

1. **Hierarchical Recurrent Encoder-Decoder (HRED-LSTM):** resembling the HRED architecture from Sordoni et al. (2015) with the difference of using Long Short-term memory (LSTM) RNNs instead of Gated Recurrent Units (GRU) RNNs. For further details refer to section 3.1.4.

   The baseline model will be trained using a batch size of 1024 and two hidden layers with dimension 128. The learning rate was set to 0.1 with a weight decay of $1e^{-6}$. It was optimised using Adam optimiser.

2. **Hierarchical Recurrent Encoder-Decoder With Attention (HREDA-LSTM):** It is a simplification of the model proposed in Dehghani et al. (2017). The model is comprised of two Bidirectional RNNs for the hierarchical encoder with an attention mechanism and a Unidirectional RNN that works as the decoder. Section 3.2.4 details the architecture used.

   The baseline model will be trained using a batch size of 1024 and two hidden layers with dimension 128 and an attention layer with one head and also with size 128. The learning rate was set to 0.1 with a weight decay of $1e^{-6}$. It was optimised using Adam optimiser (Kingma and Ba, 2014).

   It is expected that the HRDA-LSTM performs better than the HRED-LSTM because of the attention mechanism. The attention mechanism determines what parts of the session context are important when it comes to generating a new output.

   Research question RQ.2 RQ.3 RQ.4 directly involve analysing if a Transformer model can be applied for query suggestion task. Additionally, it is needed to assess if certain mechanisms or architectures can boost the performance. To answer these research questions, the following variants of Transformer model will be tested:

1. **Vanilla Transformer (VT):** Using the basic transformer architecture without any change in the architecture. Section 4.2 details the input representation used to adapt the transformer architecture to query suggestion task. It is expected that the model has a poor performance in comparison with the rest. This is because no adjustments have been done to the architecture. Consequently, the architecture is not able to model the information appropriately. A hypothesis of why this happens is that the performance is similar to employing a generic seq2seq model.

   The model has the following hyperparameters: The embeddings used have dimension 512. The number of transformer blocks used is $B = 6$. It was optimised using Adam optimiser. The learning rate was set to 0.2 with a Noam scheme. The batch size was set to 4096. Dropout

   “Noam” scheme is a particular way to set the learning rate warmup and decay together. It sets a linear warmup for a given number of steps, in this case 4000, followed by an exponential decay.
(Srivastava et al., 2014) is applied to the word embedding and position embedding, the interior layers, and final states with a rate of 0.1. The number of attention heads was \( h = 8 \).

2. **No Positional Embedding Transformer (T-NOPOS):** The model is a modification of the VT that has a non-ordered approach; it eliminates the relative order between words in queries following the intuition that words in query suggestions function as keywords and the order doesn’t substantially affect the performance of the model. Further details can be found in section 4.3.

   The model was trained using the same hyperparameters as the VT. It is expected that the performance of the model is similar to the VT. If this is the case, the intuition behind this change can be assumed as correct. Thus, a simpler version of the transformer can be used to build upon an architecture that better adapts to the task.

3. **Structured Input Transformer (SIT):** It introduces a Bidimensional input representation to model query session information appropriately. Information is represented in a semi-ordered structured, the order between queries in a session is kept however, queries have a bag of words representation. Section 4.4 further details the model characteristics.

   It is foreseeable that this model performs better than the VT and the T-NOPOS because the input representation includes information about the session, it delimits the sessions within the examples.

   The model was implemented using the following hyper-parameters: All embeddings have dimension 512. The model is composed by \( B = 6 \) transformer blocks and \( h = 8 \) attention heads. It was optimised using Adam optimiser. The learning rate was set to 0.2 with a Noam scheme. The batch size was set to 4096. Dropout (Srivastava et al., 2014) rate was set to 0.1 for all layers.

4. **Structured Input Transformer with Hierarchical Architecture (SIT-HER):** This model builds upon the SIT, it introduces query level information with a hierarchical architecture. This allows the model to have session context information. Thus, it is likely to perform better than previous mentioned models. Section 4.5 details the changes made.

   The embeddings in all the model have dimension 512. \( B = 6 \) where used for the architecture and attention heads. The learning rate was set to 0.2 with a Noam scheme and it was optimised using Adam optimiser. The batch size was set to 4096. Dropout (Srivastava et al., 2014) rate was set to 0.1 for all layers.

5. **Structured Input Transformer With Clicks (SIT-CLICKS):** This model extends the SIT by introducing user’s search intent information. The aim of this model is to introduce context information from another source. Previous models introduce search context by encoding session level information. However, assuming that users have different preferences from the previous queries in a session might help to have a broader context. Thus, it is foreseeable that the introduction of user’s search intent information will boost the performance of the SIT.
Further details can be found in section 4.6.

The model was trained using the following hyperparameters: All embeddings have dimension 512. The model is composed by $B = 6$ transformer blocks and $h = 8$ attention heads. The learning rate was set to 0.2 with a Noam scheme and it was optimised using Adam optimiser. The batch size was set to 4096. Dropout (Srivastava et al., 2014) rate was set to 0.1 for all layers.

5.4 Implementation Details

All models were implemented in Tensorflow (Abadi et al., 2016), using Tensor2Tensor library (Vaswani et al., 2018).

Tensor2Tensor (T2T) is a library of deep learning models that tries to make more accessible the design of deep neural networks with the goal of fastening research. This library allows to standardise the usage between the models presented in this thesis. With it, the same dataset can be used for all the models and the training and evaluation of them can be unified.

To be able to build the models in T2T, several components need to be specified. Among the components, the following were needed for the development of this thesis:

1. **Dataset**: To handle the preprocessing of the dataset, a Problem class was defined: aol.py. In it, the raw data is automatically downloaded and the dataset is generated. The training examples are generated with the input representation defined and the pipelines for the training and evaluation of our models are defined. Source code for our problem can be found in the following link: https://github.com/Danysolism/query-transformer/query-transformer/aol.py

2. **Hyperparameters**: The parameters for the model’s architecture as well as the ones for the training procedure are defined following classes: aol.py, lstm.py.

3. **Model**: The models presented in this thesis are defined in their corresponding Model class, there, the architecture of the different models is defined: Transformer.py is used for the VT and T-NOPOS. In Aol_transformer.py the models SIT, SIT-HER and the SIT-CLICKS are defined. The baselines are specified in aol_lstm.py. The source code for the transformer models can be found in the following link: https://github.com/Danysolism/query-transformer/query-transformer/aol_transformer.py. The source code for the baselines can be found in the following link: https://github.com/Danysolism/query-transformer/query-transformer/aol_lstm.py

Finally, the information to run the models and the library’s source code is presented in http://github.com/tensorflow/tensor2tensor.
6 Results

This chapter presents the experimental results from the evaluation outlined in Chapter 5. Firstly, Section 6.1 displays the general findings from the experiments carried out. Then, section 6.2 shows an analysis of the results with the aim of answering RQ.4. Section 6.3 presents a comparison between the different transformer models presented in this thesis. This comparison strives to answer RQ.2 and RQ.3. In addition, section 6.4 gives a comparison of VT and T-NOPOS to answer question RQ.4. Finally, Section 6.5 shows the weaknesses found in the models introduced and suggests changes that could lead to stronger results and a higher performance.

6.1 General results

Firstly, the quality of the models regarding the generation of the next query is assessed according to the metrics introduced in Chapter 5. They measure how well the most probable query suggestion matches the target query using strict metrics considering it is based on the exact word overlap between generated query and target query words. Two metrics are presented: word overlap normalised by the set of words in the generated query (Output Overlap) and the word overlap normalised by the set of words in the target query (Target Overlap). Also, results of embedding similarities are obtained using Cosine Similarity and Word Mover’s Distance (WMD). These metrics relax the hard assumption from Output Overlap and Target Overlap by taking semantic similarity into account. Results for these metrics are presented in Table 6.1. Results whose difference is statistically significant\(^1\) with respect to all the models are highlighted in bold. If for a metric, two or more results are highlighted in bold, then these are statistically significant with respect to all the models except between them.

<table>
<thead>
<tr>
<th>Model</th>
<th>Output Overlap</th>
<th>Target Overlap</th>
<th>Cosine Sim.</th>
<th>WMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRED-LSTM</td>
<td>0.090889</td>
<td>0.074154</td>
<td>0.56743</td>
<td>3.720417</td>
</tr>
<tr>
<td>HREALSTM</td>
<td>0.049178</td>
<td>0.057958</td>
<td>0.585388</td>
<td>3.695636</td>
</tr>
<tr>
<td>VT</td>
<td>0.059985</td>
<td>0.051739</td>
<td>0.480097</td>
<td>3.662450</td>
</tr>
<tr>
<td>T-NOPOS</td>
<td>0.066208</td>
<td>0.055036</td>
<td>0.487843</td>
<td>3.639959</td>
</tr>
<tr>
<td>SIT</td>
<td>0.028033</td>
<td>0.057559</td>
<td>0.528239</td>
<td>3.772297</td>
</tr>
<tr>
<td>SIT-HER</td>
<td>0.019104</td>
<td>0.044408</td>
<td>0.539484</td>
<td>3.779081</td>
</tr>
<tr>
<td>SIT-CLICKS</td>
<td>0.023567</td>
<td>0.046201</td>
<td>0.540472</td>
<td>3.777579</td>
</tr>
</tbody>
</table>

Secondly, a softer version of the evaluation is presented in Table 6.2. As explained in section 5.2.1, the assumption of sessions boundaries can be sometimes inaccurate or the user could be using navigational queries that are noise for this task. The softer version of the evaluation measures the quality of the query suggestion generation with a less noisy test set.

\(^1\) An alpha of 0.05 is used as the cutoff for significance.
It considers cases where the previous query submitted by the user and target query have more than one overlapping term.

<table>
<thead>
<tr>
<th>Model</th>
<th>Output Overlap</th>
<th>Target Overlap</th>
<th>Cosine Sim.</th>
<th>WMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRED-LSTM</td>
<td>0.172147</td>
<td>0.128593</td>
<td>0.613589</td>
<td>3.565822</td>
</tr>
<tr>
<td>HREA-LSTM</td>
<td>0.078425</td>
<td>0.087115</td>
<td>0.623497</td>
<td>3.618516</td>
</tr>
<tr>
<td>VT</td>
<td>0.158718</td>
<td>0.136445</td>
<td>0.598681</td>
<td>3.329679</td>
</tr>
<tr>
<td>T-NOPOS</td>
<td>0.166807</td>
<td>0.136529</td>
<td>0.568057</td>
<td>3.316989</td>
</tr>
<tr>
<td>SIT</td>
<td>0.046780</td>
<td>0.089986</td>
<td>0.563649</td>
<td>3.693490</td>
</tr>
<tr>
<td>SIT-HER</td>
<td>0.021988</td>
<td>0.048479</td>
<td>0.561047</td>
<td>3.724289</td>
</tr>
<tr>
<td>SIT-CLICKS</td>
<td>0.026406</td>
<td>0.054136</td>
<td>0.558181</td>
<td>3.764766</td>
</tr>
</tbody>
</table>

Table 6.2: Soft evaluation results for the five models proposed and the baselines. Output Overlap, Target Overlap, Cosine Similarity and WMD are assessed for queries were the input and target query have more than one overlapping term. The evaluation is done by calculating how well the most probable query suggestion matches the target query. Results whose difference is statistically significant with respect to all the models are highlighted in bold. If for a metric, two or more results are highlighted in bold, then these are statistically significant respect to all the models except between them.

As mentioned in previous section, evaluating the models’ performance using the ground truth is hard. By using the strict metrics like Output Overlap and Target Overlap we consider the exact word overlap for only one of the infinite possibilities that could be generated to retrieve the information needed by a user, at a given time. Therefore, it is not surprising that the results obtained from those metrics were low.

### 6.2 Comparison of Transformer models with Current-state-of-the-art models

Looking at the evaluation in 6.1, Output Overlap results show that HRED-LSTM performance is statistically significant better than the rest of models. This suggest that in the HRED-LSTM there is a higher proportion of overlapping words among the suggestions retrieved by the HRED-LSTM model. Also, the increase in Output Overlap among VT and HREA-LSTM is statistically significant, meaning that it is possible to apply a Transformer to query suggestion task. Target Overlap results also show that HRED-LSTM performs better than the rest of models and the difference is statistically significant. This implies that the HRED-LSTM model’s query suggestions have more overlapping terms with respect to the terms of the test query. The difference among the rest of the models is not statistically significant. The semantic analysis using Cosine Similarity reveals that HREDALSTM query suggestions are semantically more similar to the ground truth and the difference is statistically significant. However, when the suggestions are evaluated using WMD, the difference among the results of all models is not statistically significant. A possible explanation for this might be that HRED-LSTM outperforms in overlapping stop words\(^2\). Models might have more overlapping stop words, but when we analyse the results semantically, specially with WMD that does not consider stop words as part of the evaluation, the difference disappears.

Once noisy queries are removed from the evaluation, in Table 6.2, Output Overlap results show that HRED-LSTM performance is still better than the rest of the models and the difference is statistically significant\(^3\). However, observing the results for Target Overlapping, there is no statistically significant difference between HRED-LSTM, VT and T-NOPOS. These three models perform better than the rest with a statistical signifi-

---

\(^2\) Stop words are some of the most common words in a language. They are short function words, that is, words that have little lexical meaning. Some examples are: the, is, at, which and on.

\(^3\) An alpha of 0.05 is used as the cutoff for significance.
cance. This suggests that HRED-LSTM, VT and T-NOPOS have around the same amount of overlapping terms with respect to the terms of the test query when queries are not noisy. Moreover, the semantic evaluation shows that HREDA-LSTM query suggestions are semantically more similar to the ground truth when evaluated with Cosine Similarity metric and the distance is statistically significant. However, that difference disappears when the suggestions are evaluated using WMD. This might be because the Cosine Similarity averages the word embeddings to calculate the distance between queries. Thus, the influence of each word is smoothed out. WMD results show that, once the noisy queries are removed, VT and T-POS suggestions are semantically more similar to the ground truth than the ones from the other models. As mentioned before, this might be because HRED-LSTM outperforms in overlapping stop words and even though VT and T-POS have less overlapping words, the suggestions are semantically more relevant, outperforming the rest of the models with a statistically significant difference.

To conclude, results indicate that the transformer models introduced perform slightly worse or equally in strict metrics. Nevertheless, when looking at the less noisy dataset, two of the transformer models presented outperform HREDA-LSTM in three out of the four metrics. Furthermore, these transformer models also perform better or equal than HRED-LSTM in two out of four metrics used. Thus, answering RQ.1, results confirm that it is possible to use a Transformer architecture for query suggestion task. Moreover, it is possible to achieve similar results to current state-of-the-art query suggestions models.

### 6.3 Transformer models Comparison

Table 6.3 shows some examples of the query suggestions output by the five models proposed. Given a context (previous query in a session) the model outputs a suggestion. The table also shows the target query taken from the ground truth for each case. Examples 1, 4, 5 show cases where the models were able to match the target query. Examples 3, 6, 7, 13 show how sometimes the model is unable to match the target query because the context does not correspond to the target query. However, in these cases the query suggestions have semantic similarity with the context. In other cases, such as in 2, 8, 9, 10, 14 it is difficult to exactly match the target because it contains very specific terms that are out of the vocabulary of the models, such as: flys, past, flag, yuma, sedona, lotto, lifehouse, cabo san lucas, etc. Even so, in examples 9, 10, 14, 15 suggestions are semantically similar. These results justify the use of the softer evaluation and shows the issues with comparing query suggestions with only one target of the possible infinite ones.

Results form the evaluations in 6.1 and 6.2 show that in general the performance of the simpler architectures, VT and T-POS, is better than the Structured Transformers proposed (SIT, SIT-HER and SIT-CLICKS). VT and T-POS are significantly better in Output Overlap for both evaluations. A possible explanation for this is that Structured Transformers try to fit the length of the output suggestions to the length in the input structure.
That is, models output queries with a fixed size of 10. This results in long suggestions that move away from the target query’s space. To better show this behaviour, Table 6.4 presents an example for the five models given the same context.

<table>
<thead>
<tr>
<th>Model</th>
<th>Context</th>
<th>Query Suggestion</th>
<th>Target Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT</td>
<td>western school corporation russiaville indiana as the world turns</td>
<td>western school corporation russiaville indiana</td>
<td>western school corporation russiaville indiana</td>
</tr>
<tr>
<td></td>
<td>new york times official site</td>
<td>new york daily news</td>
<td>sonata arctica official site</td>
</tr>
<tr>
<td></td>
<td>trinity catholic high school ocala florida</td>
<td>catholic school in orlando florida</td>
<td>barlow girl lyrics never alone</td>
</tr>
<tr>
<td>T-NOPOS</td>
<td>free english to spanish translator</td>
<td>spanish to english translator</td>
<td>english to spanish translator</td>
</tr>
<tr>
<td></td>
<td>obituaries lee alan gaunt</td>
<td>rental homes in kissimmee fl</td>
<td>vacation rental houses california</td>
</tr>
<tr>
<td>SIT</td>
<td>invasive breast cancer prognosis</td>
<td>cancer staging cancer staging</td>
<td>pomeranian dogs for sale in tennessee</td>
</tr>
<tr>
<td></td>
<td>air force one fys past flag yuma arizona</td>
<td>flights from philadelphia to san francisco</td>
<td>day spas in the dallas fort worth area</td>
</tr>
<tr>
<td></td>
<td>new york city human resources</td>
<td>you re max executive</td>
<td>air force one flies</td>
</tr>
<tr>
<td>SIT-HER</td>
<td>flagstaff az realstate for sale</td>
<td>real estate prep test and yellow ink</td>
<td>sedona real estate for sale</td>
</tr>
<tr>
<td></td>
<td>online search for alabama vital records</td>
<td>how to find a woman s heart over we ll</td>
<td>vital records new orleans</td>
</tr>
<tr>
<td></td>
<td>hotels and motels in san antonio texas</td>
<td>homes for sale in knoxville tenn tenn</td>
<td>johnson inn suites san antonio</td>
</tr>
<tr>
<td>SIT-CLICKS</td>
<td>new york state lotto</td>
<td>new york state standards for art of</td>
<td>jobs to work off the book</td>
</tr>
<tr>
<td></td>
<td>map north carolina</td>
<td>education of public</td>
<td>hotel and beach resort in cabo san</td>
</tr>
<tr>
<td></td>
<td>hanging by a moment lifehouse lyrics</td>
<td>beverly hills hotels trip advisor</td>
<td>lucas mexico</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jalousie plantation in myrtle beach</td>
<td>myspace codes for music videos</td>
</tr>
</tbody>
</table>

Further looking at the experimental results, it can be seen that the difference between SIT, SIT-HER, SIT-CLICKS is not statistically significant for all metrics in 6.1. However, the soft evaluation showed that SIT performs better in Output Overlap and Target Overlap. This might imply that introducing a hierarchical architecture or user’s preference information to the transformer model not only is not beneficial but it might also affect the performance of the models when the target queries are not noisy (RQ.2, RQ.3).

6.4 Order information in Transformer models

The comparison between VT and T-NOPOS directly entails the study of removing information about the order of the sequence. The difference between the two models is the elimination of positional embeddings (4.1) in T-NOPOS.

The results obtained in both evaluations (Tables 6.1 and 6.2) show that the difference between VT and T-NOPOS was not statistically significant for all metrics. This implies that there is no difference in performance
between these models. Thus, answering RQ.4, the Transformer models do not require information about the order of sequences. As mentioned in 4.3, the intuition behind this is that words in a query act as key words to filter information and relative order does not affect the performance of the model.

For this task, it was assumed that modelling the syntactically correct order of words in a query is not significant for the task and, therefore, queries were evaluated using non-ordered metrics. However, it is interesting to see how examples from T-NOPOS from 6.3 show queries in which words are still in the syntactically correct order. This might be due to the fact that, even though the positional embeddings are not used and the positional information is not directly included, the words in the queries are fed in order, giving away some information about the position, which the model learns to use.

6.5 Improvements

After looking at the evaluation results we can see several aspects where there is still room for improvement.

Firstly, query suggestion task is very challenging and training models with noisy data increases the complexity of the task. Finding different approaches to preprocess data could lead to better results. For example the assumptions made until now where the session boundary is set after the user has been idle for 30 minutes leaves a lot of room for noisy queries and navigation queries. Perhaps narrowing the characterization of a query suggestion could lead to preprocessing data into training examples with the desire patterns in a query suggestion and would facilitate the models’ learning.

Secondly, results have shown that the input structure used for the Structured Transformers does not improve the performance in the task in comparison with VT and T-POS. Thus, a different way to input information could be explored, by either finding a way to include query level information in the T-NOPOS input representation or by trying a more complex input representation different than the one proposed in this thesis.

In addition, one of the biggest challenges in word-based seq2seq models is that they can not deal without-of-vocabulary words (OOV) and the transformer models presented are not an exception. Examples of the test data set using their vocabulary are:

- \textless LINK\textgreater \textless LINK\textgreater in \textless LINK\textgreater texas to \textless LINK\textgreater and \textless LINK\textgreater
- \textless LINK\textgreater your \textless LINK\textgreater lyrics
- \textless LINK\textgreater palm \textless LINK\textgreater test
- \textless LINK\textgreater \textless LINK\textgreater \textless LINK\textgreater center city \textless LINK\textgreater
- \textless LINK\textgreater for \textless LINK\textgreater with the \textless LINK\textgreater the \textless LINK\textgreater edge
- gold \textless LINK\textgreater \textless LINK\textgreater \textless LINK\textgreater recipe what is \textless LINK\textgreater vs \textless LINK\textgreater
As it be seen there is a great amount of <UNK> tokens, making it complicated to grasp the context, even for humans.

According to the AOL query log statistics (Pass et al., 2006), more than 62% of the words retained in a user’s session are from the 10% less frequent words. Since the vocabulary is built using the most frequent words, it is most likely that those terms are OOV. Thus, models that are not able to generate out-of-vocabulary words (OOV) will not be able to model term retention, which is present in more than half of the queries in the dataset. A possible solution for this problem is including a pointer network in the decoder as in Dehghani et al. (2017). Facilitating the access to terms from the session context during the decoding stage, specially when the term is OOV.

Lastly, it would be interesting to find a different set of metrics that could better highlight the difference between the models presented. Choosing adequate metrics for this task is challenging. One of the reasons for this is that it is hard to characterize a correct query suggestion. In previous literature, the metric used to compare state-of-the-art models is focused on the application this models will be deployed in (Sordoni et al., 2015), without considering the task itself. However, because this thesis studied a comparison between the models performance in the task, then the metrics proposed were more logical than application-wise metrics. However, the evaluation proposed only compares one of the infinite possible queries that could be generated to retrieve the information needed by a user. Finding different metrics could give a better insight on the difference between the models. This could lead to finding further changes in the models that could increase their performance.
7 Conclusion

This thesis proposed a novel method for query suggestion task. Existing models use complex recurrent architectures which require a great amount of computational power and hours to train them. As a solution, the proposed method introduces a Transformer architecture which is an encoder-decoder architecture based only on attention. By doing so, the computational complexity is decreased as well as the number of sequential operations required.

The models proposed were evaluated by comparing them to current state-of-the-art architectures (Sordoni et al. (2015), Dehghani et al. (2017)). An evaluation with experiments on the AOL dataset, showed that it is possible to use a Transformer architecture for query suggestion task. The Transformer models for query suggestion proposed can obtain results similar to current state-of-the-art query suggestions models (RQ.4). Furthermore, they do not require information about the order of sequences (positional embeddings) to perform well (RQ.4) and they do not require a hierarchical encoder to capture query level information (RQ.2).

Additionally, this thesis proposed an approach that additionally introduces information about the rank of the documents clicked on the search engine result page, for each query on the dataset, to characterise the preference the users. It has shown that including this user’s preference information in the models proposed does not boost the performance of the model (RQ.3).

Lastly, this thesis has made a contribution to the Tensor2Tensor (Vaswani et al., 2018) library by including the models presented as part of their components. By doing so, researches can reuse or continue working on extending the work presented in this thesis.

7.1 Future Work

In addition to the models examined in this thesis, future work could research other variations that also use a transformer architecture. Several improvements in this sense were outlined in section 6.5. This involve developing new input structures to refine the information passed to the T-NOPOS model; or narrowing the characterization of query suggestions to reduce the noise in the data set enabling the design of better metrics and better training.

It would also be interesting to explore how would the addition of a pointer network benefit the models presented here. This would address one of the biggest hurdles in word-base seq2seq models: Out Of Vocabulary (OOV) words. Models are required to deal with complex queries where the suggestions might not have been present in the previous examples. The transformer models presented in previous sections are able to produce synthetic suggestions1. However, the expressiveness of the models depends on the size of the vocabulary, and sometimes specific terminology or technical vocabulary needed to formulate an adequate

---

1 Synthetic suggestions are queries that the model has never seen before but whose words are in its vocabulary.
query are not part of the vocabulary (OOV). Following the study done by Dehghani et al. (2017), including a copy mechanism (pointer network) in the decoder to access the terms from the session context during the decoding stage would improve suggestions in the aforesaid case given that on average 62% of the words used in the session are retained from the previous queries (Sloan et al., 2015). Including this mechanism in the previous models presented could lead to a significantly better performance of the transformer model for the query suggestion task.
References


In Proceedings of the 19th international conference on World wide web, pages 841–850. ACM.


3.1 The hierarchical recurrent encoder-decoder (HRED) for query suggestion model. Each arrow is a non-linear transformation and each circle represents the end-of-query symbol. In this example, the first query the user types is "Italian Cuisine" followed by "Pasta Carbonara Recipe". When the model is trained, query "Italian Cuisine" is encoded and the session-level recurrent state are updated. Then, the output is determined by maximising the probability of seeing the following query "Pasta Carbonara Recipe". This process is repeated for all queries in a session. During testing, a contextual suggestion is created by encoding the previous queries, updating the session-level recurrent states accordingly and finally sampling a new query. In this example, the suggestion is Italian Carbonara Recipe.

3.2 Illustration of a Vanilla RNN, a Long-short term memory (LSTM) recurrent unit and a Gated Recurrent Unit (GRU).

3.3 Learning to Attend, Copy, and Generate for Session-Based Query Suggestion model architecture.

3.4 Transformer model architecture.

3.5 Multi-Head attention is composed of Scaled Dot-Product Attention layers running in parallel.

4.1 Positional encodings are created using sinusoidal functions, namely the sine and cosine function embeddings with different dimensions. Words are encoded with the pattern created by the combination of these functions; this results in a continuous binary encoding of positions in a sequence.

4.2 Transformer input encoding. The model uses multihead attention, it attends in a forward and backward matter to obtain context from similar items in a sequence regardless of their position in the sequence. To introduce information about the order of the sequence it relies on positional embeddings.

4.3 Structured input Transformer. It has an ordered structured for queries within a session but a bag of words representation for words in a query.

4.4 Input encoding of the Structured input Transformer with hierarchical architecture. Given a session $S_m$, it introduces a context vector $C_m$ with session level information.

4.5 Structured input Transformer with clicks. It has two ordered structures, one to represent the queries within a session and the other containing users search intent within a session.

4.6 Final encoding of inputs in the Structured Input Transformer with clicks.

5.1 Frequency of the different session lengths.

5.2 Frequency of the different query lengths.

5.3 Word mover’s distance. Non-stop words from both queries are embedded using word2vec embeddings. The distance between the queries is the minimum cumulative distance that all words in the output query need to travel to exactly match the target query.

1 Average number of query terms in each position of the session (for sessions with length 2 to 9).
## Tables

6.1 Evaluation results for the five models proposed and the baselines with metrics Output Overlap, Target Overlap, Cosine Similarity and WMD. The assessment is done by calculating how well the most probable query suggestion matches the target query. Results whose difference is statistically significant with respect to all the models are highlighted in bold. If for a metric, two or more results are highlighted in bold, then these are statistically significant respect to all the models except between them.

6.2 Soft evaluation results for the five models proposed and the baselines. Output Overlap, Target Overlap, Cosine Similarity and WMD are assessed for queries were the input and target query have more than one overlapping term. The evaluation is done by calculating how well the most probable query suggestion matches the target query. Results whose difference is statistically significant with respect to all the models are highlighted in bold. If for a metric, two or more results are highlighted in bold, then these are statistically significant respect to all the models except between them.

6.3 Examples of the query suggestions from the five models proposed. Given a context the model outputs a query suggestion. The target query taken from the ground truth is also shown.

6.4 Examples of the query suggestions from the five models proposed given the same context. The target query taken from the ground truth is also shown.
Appendices

A Data Exploration

When conducting the dataset analysis in section 5.1. Further analysis was done to analyse how queries change throughout the session. For the sessions with the same length, the average number of query terms in each position was calculated. Figure 1 presents the results along the sessions from length 2 to 9. One can see that the number of terms is reduced in the second position and then the number of terms gradually increase throughout the session. This could be because the search intent changes within the session, users start with a more complex query then in the second position, a decrease in the number of terms is observed, maybe to relax restrictions and then at the end of the session extra terms are added to narrow down the search and obtain the information they need. However further statistic analysis regarding the specialisation or generalisation of queries needs to be done to confirm the observation.

Figure 1: Average number of query terms in each position of the session (for sessions with length 2 to 9).