POLITECNICO DI TORINO

Master's degree in ICT for Smart Societies

Master's Thesis

Towards Zero-shot Dialogue State Tracking

Supervisors
Prof. Maurizio Morisio
Dr. Giuseppe Rizzo

Candidate
Yu Xiahou

March 2020

Abstract

Goal-oriented dialogue systems are designed to help people in daily life to accomplish various tasks, such as restaurant reservation, movie ticket booking. Such systems assist user through a series of conversations, therefore it is important to keep track of the user goal. Dialogue State Tracking (DST) is such a component that extracts the user goal at each user turn by interpreting the user utterance. DST encodes the user goal as a set of dialogue states, i.e. slot-value pairs. The dialogue states are then used by the dialogue system to issue the corresponding API call to the backend database, in such a manner to accomplish the tasks.

Many state-of-the-art DST approaches operate on a predefined ontology, performing classification over a full candidate-value list, i.e. all slots and their corresponding values. However, such method is infeasible or impractical in the real-world system. First, oftentimes the backend database is maintained by service providers, and exposed through some external APIs, it is infeasible that DST can access to all the possible values for all slots. Second, even the entire database is available, the values may keep changing over time or simply countless, for example the showtimes in a theatre changes every once in a while, or the number of movies in an online streaming website can be huge. In addition, the model size of these systems is proportional to the number of slots, with an ever-increasing number of slots, it leads to the performance deficit, and it also requires the model to perform constantly updating or re-training. Furthermore, dialogue systems need to support an increasing number of services, as a consequence DST may encounter unseen slots. A DST that is built under such an approach can only label them as unknown without further classification.

In this work, we present a Schema-Guided approach based on pretrained BERT model to address the zero-shot state tracking challenge. In particular, we attend to a special setting where target slot value can be found as word segments in the dialogue context. We construct the input sequence as a pair

of sentences where the former is the dialogue context consists of preceding system utterance (if exists) and current user utterance, and the latter is the description of a slot in natural language, such description is obtained through the guided schema. Our model consists of BERT as the encoder, a classification module and a span prediction module. We decouple the dialogue state tracking problem into a Next Sentence Prediction like task and a Question Answering task. Each input sequence is encoded by BERT to obtain contextualized sentence-level and token-level representations. The sentence-level representation is then used by the classification module to produce a Boolean value indicating whether the value of the described slot is present in the dialogue context. Likewise, the token-level representations are used by the span prediction module to generate the start and end positions for this slot value. In the end, the dialogue state update mechanism derives the output. Such procedure iterates over all the input sequences at each conversation turn, yield the final outputs.

Empirical evaluation shows the proposed model achieves better performance than the baseline model. Rather than treating the slots as labels, the model extracts slot value from the dialogue context by interpreting the semantics of the slots. Thanks to BERT's contextualized representation, it allows our approach to be effective. Furthermore, it demonstrates transfer learning with language models has become an integral part of many language understanding problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Goal-oriented Dialogue Systems

Goal-oriented dialogue systems are designed to help people in daily life to accomplish various tasks, such as restaurant reservation, movie ticket booking. Such systems assist user through a series of conversations, therefore it is important to keep track of the user goal. Dialogue State Tracking (DST) is such a component that extracts the user goal at each user turn by interpreting the user utterance. DST encodes the user goal as a set of dialogue states, i.e. slot-value pairs. The dialogue states are then used by the dialogue system to issue the corresponding API call to the backend database, in such a manner to accomplish the tasks. Which API/service to be triggered depends on Intent Classification, a component that detects the user's intention, while it is out of the scope of this work. For what concerns this work, we focus on the DST problem.

1.2 Motivation and Objective

Many state-of-the-art DST approaches operate on a predefined ontology, performing classification over a full candidate-value list, i.e. all slots and their corresponding values. However, such method is infeasible or impractical in real-world systems. First, oftentimes the backend database is maintained by service providers, and exposed through some external APIs, it is infeasible that DST can access to all the possible values for all slots. Second, even the entire database is available, the values may keep changing over time or simply countless, for example the showtimes in a theatre changes every once in a while, or the number of movies in an online streaming website can be huge. In addition, the model size of these systems is proportional

to the number of slots, with an ever-increasing number of slots, it leads to the performance deficit, and it also requires the model to perform constantly updating or re-training. Furthermore, dialogue system need to support an increasing number of services, as a consequence DST may encounter unseen slots. A DST that is built under such an approach can only label them as unknown without further classification.

Recently, in the 8th Dialogue System Technology Challenge, Google introduced a Schema-Guided Dialogue State tracking task whose goal is to develop dialogue state tracking model suitable for large scale dialogue systems, with a focus on zero-shot generalization to new APIs. This task provides a new dataset, namely the Schema-Guided Dialogue (SGD) dataset [20] containing 16k multi-domain dialogues across 17 domains in train and development sets. It is the largest corpus of annotated task-oriented dialogues, especially, evaluation sets contain many unseen domains and services. Consequently slots which are not present in the training set ideally serve the purpose to test the model's ability to generalize in zero-shot setting. We are highly motivated by this challenge, our objective is to build a DST able to perform zero-shot dialogue state tracking, utilizing the knowledge acquired in training and apply to the unseen slots at test time.

1.3 Task Description

In this work, we present our proposed approach to address the zero-shot state tracking challenge. In particular, we attend to a special setting where target slot value can be found as word segments in the dialogue context. The task we perform in this work is described as follows: at each user turn, the propose DST model extracts the dialogue state update from the dialogue context consists of the current user utterance and preceding system utterance. The state update is considered to be the difference between the slot values present in the current turn and the ones present in the previous user turn (if exists). Related work is introduced in the following chapter.

Chapter 2

State-of-the-art

In this chapter, we conduct literature review on the background knowledge. First, we review some state-of-the-art models from word embedding to language modeling. Then, we introduce the concept of transfer learning, a key factor to recent success in NLP. Finally, we briefly report the related work on dialogue state tracking for goal-oriented dialogue systems.

2.1 Word Embedding

How to present words is the first and arguably most important thing we need to deal with in natural language processing (NLP) world, as words are the input in almost all the NLP tasks. The quality of the representations is essentially a key factor to downstream tasks in various NLP research.

In traditional NLP works, words are treated as discrete symbols, in other words one-hot vectors, it's a simple and robust approach. However, the fundamental limitation of this technique is the lack of notion of similarity, every vector is orthogonal to another one.

> "You shall know a word by the company it keeps"
>
> — J.R.Firth

Underlying the idea popularized by Firth in 1950s: a word's meaning is characterized by the words frequently co-occur, words represented by distributed vectors on the other hand, makes similarity measure trivial as each value in a vector captures a dimension of the word's meaning, so that similar words eventually have similar vectors. In the following section, 3 models will be discussed, namely Word2Vec [13], GloVe [18] and fastText [3].

2.1.1 Word2Vec

Introduced by Mikolov et al. in 2013, word2vec is a framework for learning word embeddings. It demonstrates how high-quality word embeddings can be learned from huge datasets with billions of words through a shallow network in a computationally efficient way. Two model architectures are proposed, Continuous Bag-of-Words (CBOW) and continuous Skip-gram [13]. In addition, Negative Sampling training strategy and the subsampling of the frequent words significantly accelerate the whole learning process and improve the quality of the vectors as well [12].

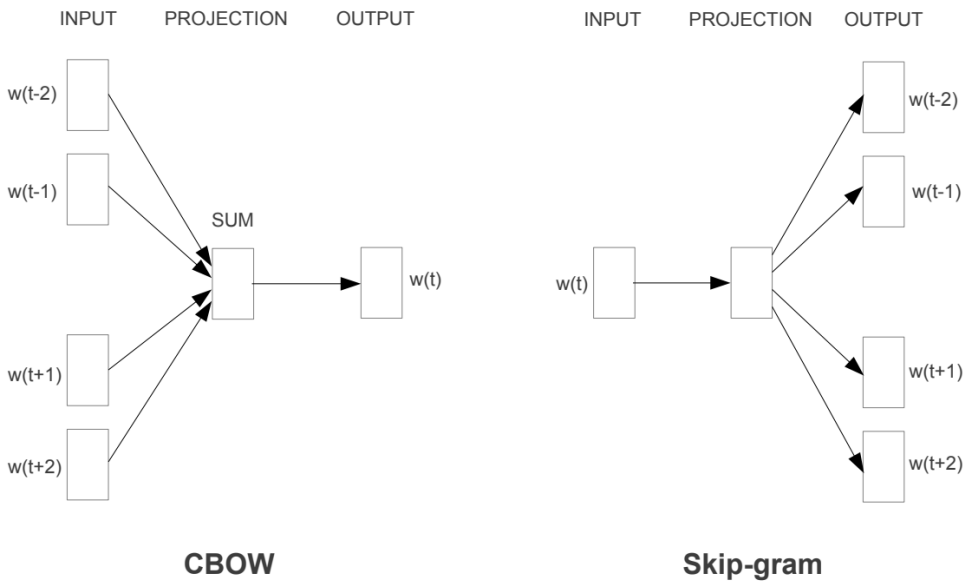

Figure 2.1: Two model architectures introduced by Mikolov et al. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word [13].

CBOW

CBOW is built upon the idea that a word is to be predicted from its surrounding context words.

Consider following example sentence:

"The cat jumped over the puddle"

CBOW model aims to predict the word "jumped" from its context "The", "cat", "over", "the", "puddle".

First, the input sentence can be represented by one-hot vectors $x^{(i)}$, and the output is represented as y. Then we created two matrices, $\mathcal{V} \in \mathbb{R}^{n \times |V|}$ and $\mathcal{U} \in \mathbb{R}^{|V| \times n}$, where n is an arbitrary number which defines the size of out embedding space. $\mathcal{V}$ is the input word matrix such that i-th column of $\mathcal{V}$ is the n-dimensional embedded vector for word w_i when it is an input word. Likewise, $\mathcal{U}$ is the output word matrix such that j-th column of $\mathcal{U}$ is the n-dimensional embedded vector for word w_j when it is an output word. In fact we learn these two vectors for every word w_i, i.e. the input vector v_i and the output vector u_i. The model works in following steps:

1. We generate one-hot vector for input context words with size m: ($x^{(c-m)}$, ..., $x^{(c-1)}$, $x^{(c+1)}$, ..., $x^{(c+m)} \in \mathbb{R}^{|V|}$)
2. We get embedded word vectors for the context ($v_{c-m} = \mathcal{V}x^{(c-m)}$, ..., $v_{c+m} = \mathcal{V}x^{(c+m)} \in \mathbb{R}^{|n|}$)
3. Average these vectors we get $\hat{v} = \frac{v_{c-m}+...+v_{c+m}}{2m} \in \mathbb{R}^{|n|}$
4. We can then generate a score vector $z = \mathcal{U}\hat{v} \in \mathbb{R}^{|V|}$
5. Convert the score to probabilities $\hat{y} = \text{softmax}(z) \in \mathbb{R}^{|V|}$
6. We expect our generated probabilities $\hat{y} \in \mathbb{R}^{|V|}$ to match the true probabilities $y \in \mathbb{R}^{|V|}$, which is also the one-hot vector of the actual word.

We use cross-entropy to be the objective function for the model to learn the two matrices, i.e $\mathcal{V}$ and $\mathcal{U}$.

$$H(y, \hat{y}) = -\sum_{j=1}^{|V|} y_j \log \hat{y}_j$$

As the output is an one-hot vector, the loss function can be simplified

$$H(y, \hat{y}) = -y_i \log \hat{y}_i$$

Let c be the index of the correct word, its one-hot vector is 1 thus if the prediction is perfect we have $y_c = 1$, the loss is calculated as $H(y, \hat{y}) = -1 \log 1 = 0$. In other words, if the prediction is correct, there is no loss. To the contrary, if the prediction goes very bad that $y_c = 0.01$, then we have the

loss $H(y, \hat{y}) = -1 \log 0.01 \approx 4.605$ to be very high. Thus we can formalize the optimization objective as:

$$\begin{aligned}
\text{minimize } J &= -\log P(w_c | w_{c-m}, ..., w_{c-1}, w_{c+1}, ..., w_{c+m}) \\
&= -\log P(u_c | \hat{v}) \\
&= -\log \frac{\exp(u_c^T \hat{v})}{\sum_{j=1}^{|V|} \exp(u_j^T \hat{v})} \\
&= -u_c^T \hat{v} + \log \sum_{j=1}^{|V|} \exp(u_j^T \hat{v})
\end{aligned}$$

We use stochastic gradient descent to update all relevant word vectors u_c and v_j.

Skip-gram

Skip-gram, instead, is built upon the idea that surrounding words are able to be generated given the center word.

Thus consider the same example, skip-gram model aims to predict the context words "The", "cat", "over", "the", "puddle" given the word "jumped". The input one-hot vector can be represented as x, so do the output vectors as $y^{(j)}$. Similarly, we define $\mathcal{V}$ and $\mathcal{U}$ the same as in CBOW. The model works the way as follows [5]:

1. We generate one-hot vector for input center word $x \in \mathbb{R}^{|V|}$
2. We get embedded word vector for the center word $v_c = \mathcal{V}x \in \mathbb{R}^{|n|}$
3. Generate a score vector $z = \mathcal{U}v_c$
4. Convert the score to probabilities $\hat{y} = \text{softmax}(z)$, so we have $\hat{y}_{c-m}$, ..., $\hat{y}_{c-1}$, $\hat{y}_{c+1}$, ..., $\hat{y}_{c+m}$ as the probabilities for each context word.
5. We expect our generated probabilities to match the true probabilities y_{c-m}, ..., y_{c-1}, y_{c+1}, ..., y_{c+m} which are one-hot vectors for the actual output.

We can formalize the optimization objective as:

$$\begin{aligned}
\text{minimize } J &= -\log P(w_{c-m}, ..., w_{c-1}, w_{c+1}, ..., w_{c+m}|w_c) \\
&= -\log \prod_{j=0, j\neq m}^{2m} P(w_{c-m+j}|w_c) \\
&= -\log \prod_{j=0, j\neq m}^{2m} P(w_{c-m+j}|v_c) \\
&= -\log \prod_{j=0, j\neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|} \exp(u_k^T v_c)} \\
&= -\sum_{j=0, j\neq m}^{2m} u_{c-m+j}^T v_c + 2m \log \sum_{k=1}^{|V|} \exp(u_k^T v_c)
\end{aligned}$$

We compute the gradients with respect to unknown parameters at each iteration and update them via stocastic gradient descent.

Negative Sampling

Observe that the loss function J for CBOW and Skip-gram are computationally expensive because of the softmax normalization, where we have to sum over all $|V|$ scores. Negative sampling is the technique to approximate it in a computationally efficient way. The main idea is to turn the softmax normalization to logistic regression, train a true pair (the center word and its context) versus several noise pairs (the center word paired with some random word). Let $P(D = 1|w, c)$ denotes the probability that a pair of word and context (w, c) is a true pair, likewise, $P(D = 0|w, c)$ be the probability that (w, c) is a noise pair. We model $P(D = 1|w, c)$ with the sigmoid function:

$$P(D = 1|w, c, \theta) = \sigma(v_c^T v_w) = \frac{1}{1 + e^{(-v_c^T v_w)}}$$

The corresponding objective function is then to maximize the probability of a word and context being a true pair if it indeed is, and maximize the probability of a word and a random word being a false pair as well. We take

a maximum likelihood approach of these two probabilities.

$$
\begin{aligned}
\theta &= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta) \\
&= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1-P(D=1|w,c,\theta)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log\left(1-P(D=1|w,c,\theta)\right) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log\left(1-\frac{1}{1+\exp(-u_w^T v_c)}\right) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log \frac{1}{1+\exp(u_w^T v_c)}
\end{aligned}
$$

Such optimization is the same as minimizing the negative log likelihood:

$$
J = -\sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} - \sum_{(w,c)\in \tilde{D}} \log \frac{1}{1+\exp(u_w^T v_c)}
$$

where $\tilde{D}$ are the noise pairs that we can generate on the fly by random sampling.

For CBOW the new objective function for observing the center word u_c given the context vector $\hat{(v)} = \frac{v_{c-m}+...+v_{c+m}}{2m}$ would be

$$
-\log \sigma(u_c^T \cdot \hat{v}) - \sum_{k=1}^{K} \log \sigma(-\tilde{u}_k^T \cdot \hat{v})
$$

For Skip-gram the new objective function for observing the context word $c-m+j$ given center word c word be

$$
-\log \sigma(u_{c-m+j}^T \cdot v_c) - \sum_{k=1}^{K} \log \sigma(-\tilde{u}_k^T \cdot v_c)
$$

$\{\tilde{u}_k | k = 1...K\}$ are sampled from the noise distribution $P_n(w)$, as per the author, the unigram distribution raised to the 3/4rd power makes the best approximation.

Subsampling of Frequent Words

In addition, we also observe that frequent words such as "a", "the"" easily appear hundreds of millions of times, while they usually provide much less information than the rare words do. To counter the imbalance between rare and frequent words, the subsampling approach is applied: each word w_i in the training set is discarded with a probability

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

where $f(w_i)$ is the frequency of word w_i and t is the chosen threshold, typical value is around 10^{-5}. Such setup significantly subsamples the words whichever occur greater than t while preserving the ranking of the frequencies. It accelerates the training, furthermore it improves the representation quality of the rare words.

2.1.2 GloVe

In spite of the success in capturing fine-grained semantic, shallow window-based models like word2vec suffer from the disadvantage of their insufficiency in global word co-occurrences statistics usage. Pennington et al. proposed GloVe, a log-bilinear regression model combines the advantages of both global matrix factorization and local context window methods. In a way that training only non-zero elements in a word-word matrix rather than the entire sparse one, the crucial insight is that ratios of co-occurrence probabilities can encode meaning components.

Co-occurrence Matrix

We construct a word-word co-occurrence matrix by collecting the statistics over the entire corpus. Let X denote the matrix, where X_{ij} is the number of times that word j being the context of word i. Let $X_i = \sum_k X_{ik}$ be the number of times that any word k being the context of word i. And $P_{ij} = P(w_j|w_i) = \frac{X_{ij}}{X_i}$ is the probability of j appearing in the context of word i.

GloVe model

GloVe model use the global statistics of word co-occurrence in a corpus, consider the example shown in Table 2.1, suppose word $i = ice$, word $j = steam$

Probability and Ratio	k = solid	k = gas	k = water	k = fashion
P(k—ice)	1.9*E-4	6.6*E-5	3.0*E-4	1.7*E-5
P(k—steam)	2.2*E-05	7.8*E-4	2.2*E-3	1.8*E-5
P(k—ice)/P(k—steam)	8.9	8.5*E-2	1.36	0.96

Table 2.1: Co-occurrence probabilities for target words ice and steam with selected context words from a 6 billion token corpus. Only in the ratio does noise from non-discriminative words like water and fashion cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam [18].

are of our interest. Their relationship can be examined by checking the ratio of their co-occurrence probabilities with probe words. If a word related to ice but not steam, we expect the ratio P_{ik}/P_{jk} to be large, for example $k = solid$. Vice versa, a small ratio to be expect for word relate to steam but not ice, like $k = gas$. And for words either related to both, or related to none, the ratio is close to 1, such as $k = water$ or $k = fashion$.

We can describe above argument in a formal way.

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

where $w \in \mathbb{R}^d$ are word vectors and $\tilde{w} \in \mathbb{R}^d$ are separate context word vectors.

Consider the vector difference of two target words, we can modify the left hand side of the equation by taking the dot product.

$$F(w_i, w_j, \tilde{w}_k) = F(w_i - w_j, \tilde{w}_k) = F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

$$\implies \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)} = \frac{P_{ik}}{P_{jk}}$$

Meanwhile the probability can be found by co-occurrence matrix.

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

Then we can solve F by taking the logarithm.

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log X_{ik} - \log X_i$$

Notice the term $\log X_i$ is independent of k, then we can introduce a bias term to modify the equation.

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log X_{ik}$$

Compare to the original formulation, this is much simpler. While it becomes ill-conditioned if $X_{ik} = 0$, therefore we can start the word co-occurrence count from 1 instead of 0, introduce a shift in the logarithm $\log X_{ik} \to \log(1 + X_{ik})$.

For the objective function, a weighted least squares regression model is proposed here, the weighting function $f(X_{ij})$ takes the co-occurrence frequency into consideration.

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_k + b_i + \tilde{b}_k - \log X_{ik})^2$$

In particluar, the weigthing function depicted in Figure 2.2 follows below 3 properties:

1. f(0) = 0.
2. f(x) should not decrease so that rare co-occurrences are not overweighted.
3. f(x) should be relatively small for large value of x, so that frequent co-occurrences are not overweighted.

The author find a class of weighting function works well that can be parameterized as

$$f(x) = \begin{cases} (x/x_{max})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise.} \end{cases}$$

where they assign α the value of 3/4 after empirical evaluation.

2.1.3 fastText

But, is it really a good idea to represent each word of the vocabulary as a distinct vector? In 2015, Bojanowski et al. proposed fastText, a model that learns representations for character n-grams and represents words as the sum of the n-gram vectors. They argue such representations take word morphology into consideration, rare or unseen words during training can be easily computed, result in an improved performance.

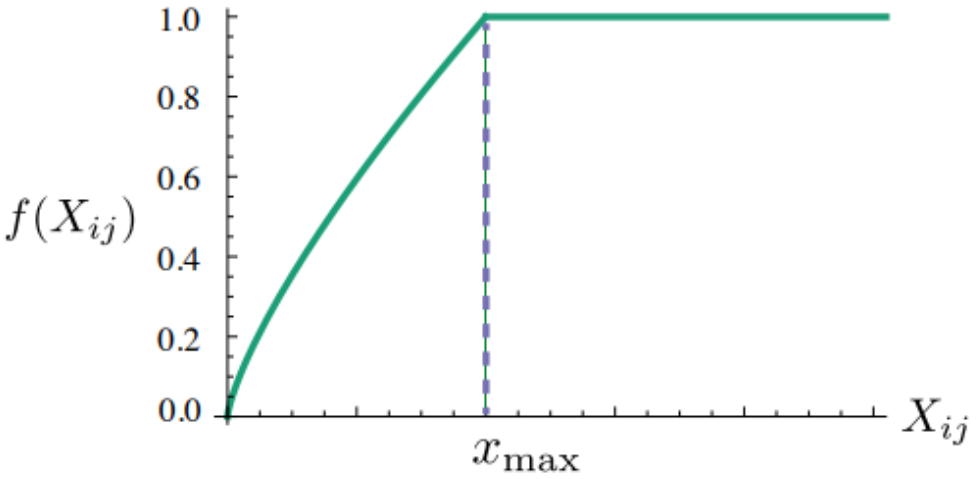

Figure 2.2: Weighting function f with $\alpha = 3/4$ [18].

Subword model

As presented in word2vec skip-gram model, each word uses a distinct vector representation, whereas in subword model, each word w is represented by a bag of characters n-gram. And the word itself is also included in the n-gram set. For instance, the word *where* represented by character n-gram when n = 3:

$$<wh, whe, her, ere, re>, <where>$$

Let $g_w \subset \{1, ..., G\}$ denote the set of n-grams appearing in word w, we associate a vector representation z_g to each n-gram g. Then the word w is represented by the sum of vector representations of its n-gram. Thus the scoring function can be obtained.

$$s(w, c) = \sum_{g \in g_w} z_g^T v_c$$

This model allows representations sharing among words, such a robust and reliable approach can easily handle rare words and out-of-vocabulary words.

2.2 Language Model

Regardless of the widespread use, one major limitation of aforementioned embeddings is handling polysemy. The meaning of a word may change upon its context, however, for context-free embeddings, there is only one single representation for a word. In this section, we will introduce two language models, ELMo [19] and BERT [6], whose contextual word representations are the perfect solution.

2.2.1 ELMo

ELMo is a deep contextualized word representation introduced by Peters et al., it models both complex characteristics of word use (e.g., syntax and semantics), and how these uses vary across linguistic contexts (i.e., to model polysemy). ELMo word representations are functions of the entire input sentence. A two-layer, bi-directional, LSTM [9] based language model with character convolutions is pre-trained on a large text corpus, a weighted sum of the hidden states that extracted from each layer in the biLM is then computed to obtain the embedding for each word in the sentence. ELMo representations can be easily added to existing models and significantly improve the state of the art across a broad range of challenging NLP problems, including question answering, textual entailment and sentiment analysis.

Bidirectional Language Model

Given a sequence of N tokens, $(t_1, t_2, ..., t_N)$, a forward language model computes the probability of token t_k given history $(t_1, t_2, ..., t_{k-1})$, whereas a backward language model computes the probability of token t_k given its future context $(t_{k+1}, t_{k+2}, ..., t_N)$.

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1})$$

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ..., t_N)$$

A bidirectional language model (biLM) combines both a forward and backward language model. The formulation below jointly maximize the log likelihood of both directions.

$$\sum_{k=1}^{N}(\log p(t_k|t_1, t_2, t_{k-1}; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s) + \log p(t_k|t_{k+1}, t_{k+2}, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s))$$

where Θ_x represents token representation and Θ_s represents softmax layer in the LSTMs.

ELMo

ELMo is a task specific combination of the intermediate layer representations in the biLM. For each token t_k in a L-layer biLM, its representation can be computed as:

$$\begin{aligned} R_k = & \{x_k^{LM}, \vec{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, ..., L\} \\ = & \{h_{k,j}^{LM} | j = 0, ..., L\} \end{aligned}$$

where $h_{k,0}^{LM}$ is the token layer and $h_{k,j}^{LM} = [\vec{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM}]$ are each biLM layer.

To use ELMo in any downstream model, a task specific weighting of all biLM layers can be computed:

$$ELMo_k^{task} = E(R_k, \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}$$

where s^{task} are softmax-normalized weights and the scalar parameter γ^{task} allows the task model to scale the entire ELMo vector.

Simply adding ELMo to existing architectures across six benchmark NLP tasks: question answering, textual entailment, semantic role labelling, coreference resolution, named entity extraction, and sentiment analysis, we can observe 6 - 20% performance improvement, as shown in Table 2.2.

2.2.2 BERT

Nevertheless, because of the unidirectional nature of standard language models, the power of pre-trained representations are still limited. Introduced by Devlin et al., BERT alleviates the unidirectionality constraint by using a masked language model, thus it allows the model to see the context from both direction, such innovation leads to a great leap forward in NLP world.

Task	Previous SOTA	ELMo	Improvement
SQuAD	84.4	85.8	4.7 / 24.9%
SNLI	88.6	88.7	0.7 / 5.8%
SRL	81.7	84.6	3.2 / 17.2%
Coref	67.2	70.4	3.2 / 9.8%
NER	91.93	92.22	2.06 / 21%
SST-5	53.7	54.7	3.3 / 6.8%

Table 2.2: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F1 for SQuAD, SRL and NER; average F1 for Coref [19].

Transformer

BERT makes use of the encoder of Transformer [26], an encoder-decoder [24] model enhanced with self attention [2] mechanism introduced by Vaswani et al, the architecture is shown in Figure 2.3. For what concerns BERT is the encoder part, it is composed of $N = 6$ identical layers. Each layer has 2 sub-layers, the first is a multi-head self-attention mechanism and the second is a fully connected feedforward network. A residual connection [7] at each of the two sub-layers, followed by layer normalization.

We can view an attention function as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

The author name the attention as scaled dot-product attention, which can be formalized as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where the input queries and keys are of dimension d_k, and values of dimension d_v.

Not only perform a single attention function, they actually project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of the projection, the attention is

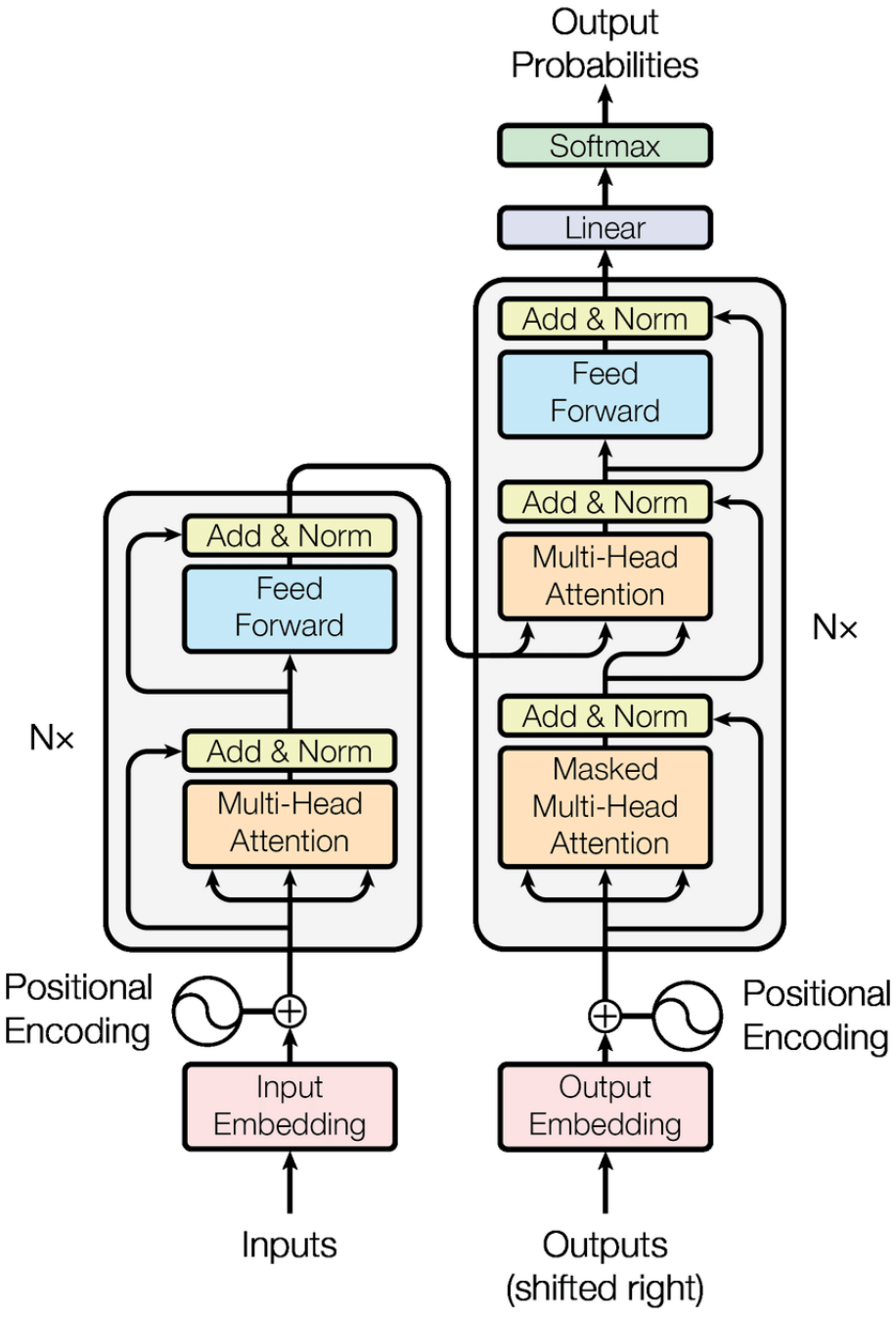

Figure 2.3: Model architecture of the Transformer [26].

performed in parallel yielding d_v dimensional output values. These values are then concatenated and projected again to obtain the final output, as shown in Figure 2.4.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$
$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

where $W_i^Q \in \mathbb{R}^{d_{model}} \times d_k$, $W_i^K \in \mathbb{R}^{d_{model}} \times d_k$, $W_i^V \in \mathbb{R}^{d_{model}} \times d_v$.

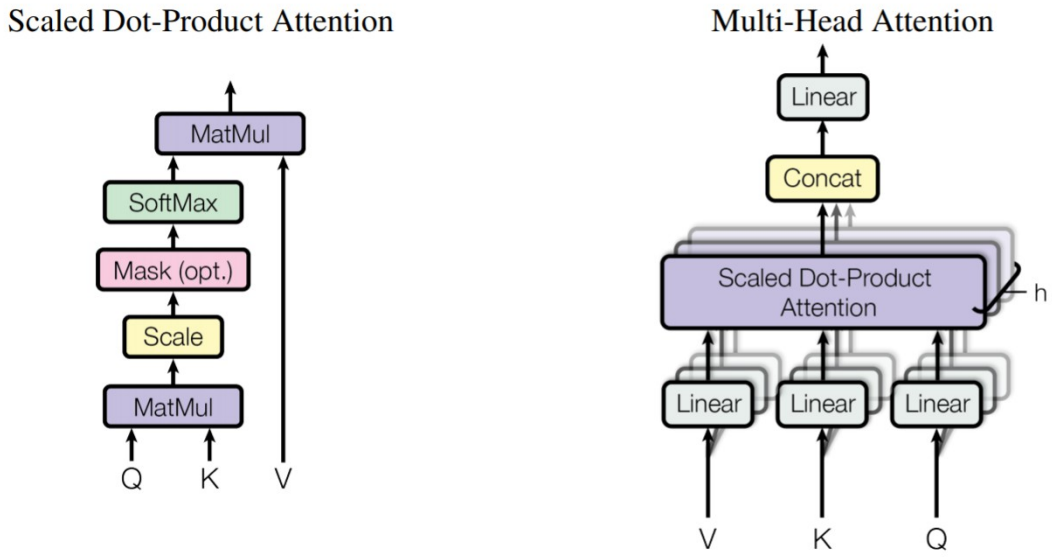

Figure 2.4: Scaled Dot-Product Attention (left). Multi-Head Attention (right) consists of several attention layers running in parallel [26].

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions, as depicts in Figure 2.5.

BERT

BERT's model architecture is a multi-layer bidirectional Transformer encoder. Two model sizes are reported, let L denote the number of transformer block, H be the hidden size, and self attention heads as A:

- $\text{BERT}_{\text{BASE}}$: L=12, H=768, A=12, parameters 110M;

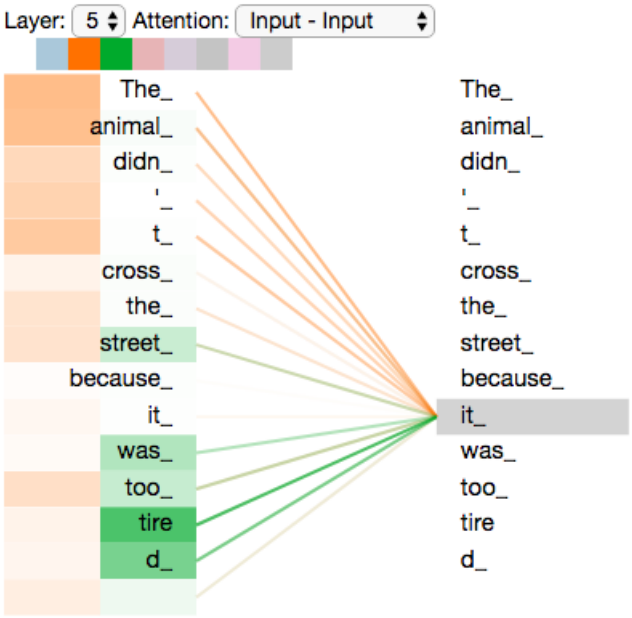

Figure 2.5: As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" – in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired" [1].

- $\text{BERT}_{\text{LARGE}}$: L=24, H=1024, A=16, parameters 340M.

The input representations is the sum of the corresponding token, segment, and position embeddings, it is illustrated in Figure 2.6. Both single sentence and a pair of sentences are accepted, while every sequence always starts with a special classification token [CLS], and uses token [SEP] as the separator.

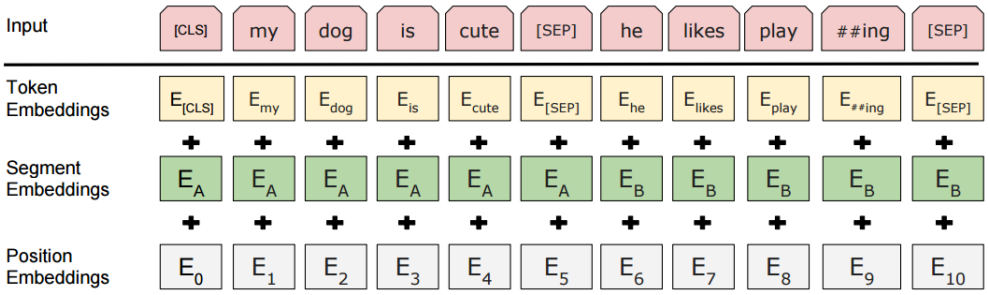

Figure 2.6: BERT input representation [6].

Bidirectional conditioning would allow each word to see itself indirectly, which makes prediction trivial, how does BERT work? BERT achieve bidirectionality by using a mask language model. In particular, during the pre-training, 15% of the tokens are randomly masked. Then the task becomes predicting the masked words given context in both direction. To illustrate in more details, the masking procedure follows:

- 80% of the time: replace the word with [MASK] token, e.g. my dog is hairy → my dog is [MASK].
- 10% of the time: replace the word with a random word, e.g. my dog is hairy → my dog is apple.
- 10% of the time: keep the word unchanged, e.g. my dog is hairy → my dog is hairy.

This forces BERT to keep a distribution contextual representation for every input token so to perform the prediction task.

In addition, in order to train the model for understanding the relationship between two sentences, BERT is also trained with a next sentence prediction task which requires the model to predict if the second sentence is the next sentence of the first one. Such task is very beneficial to downstream task

such as Question Answering (QA) and Natural Language Inference (NLI).

Fune-turning stage for BERT is pretty straightforward. For each task, simply plug task specific inputs and output onto BERT and fine-tuning all the parameters end to end, as is shown in Figure 2.7.

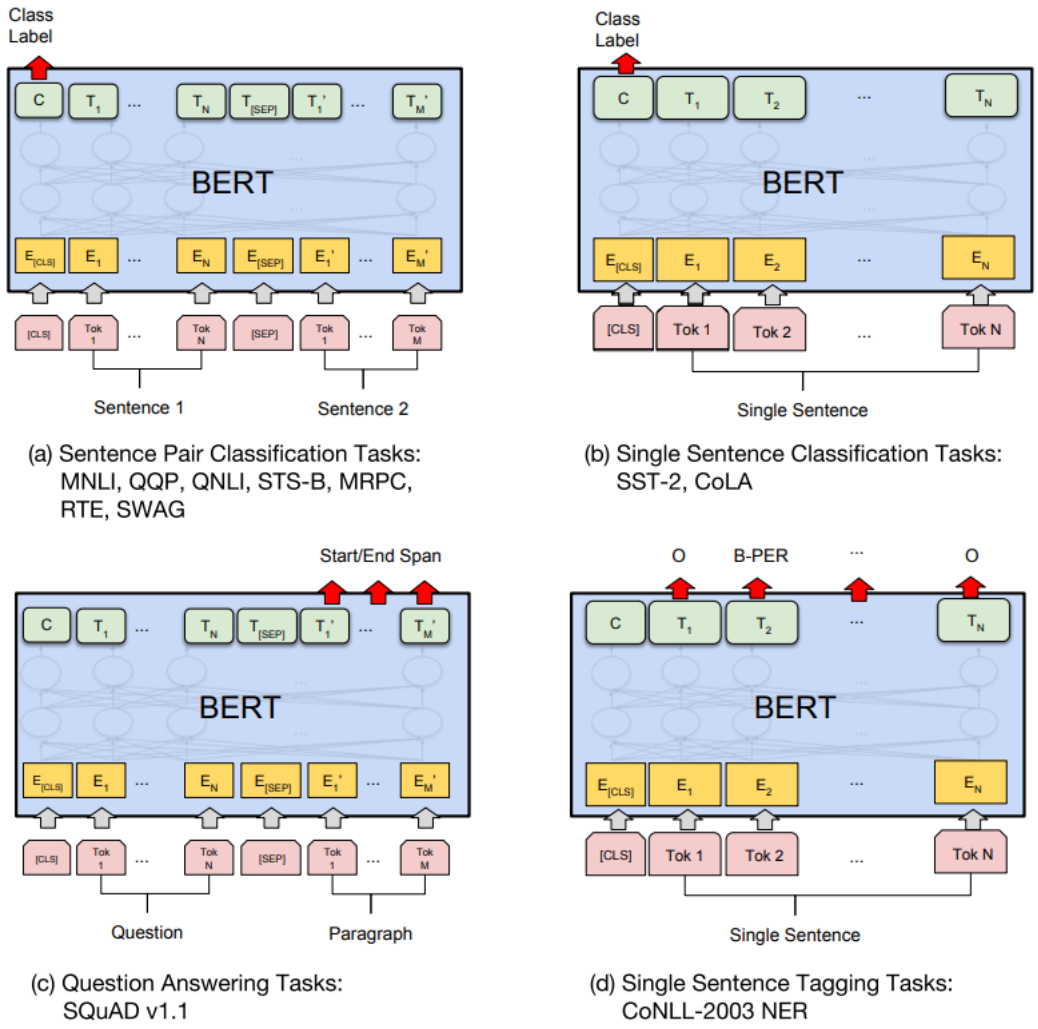

Figure 2.7: Illustrations of Fine-tuning BERT on Different Tasks [6].

2.3 Transfer Learning

Transfer learning can be defined as a means to extract knowledge from a source setting and apply it to a different target setting [22], an illustration

is in Figure 2.8.

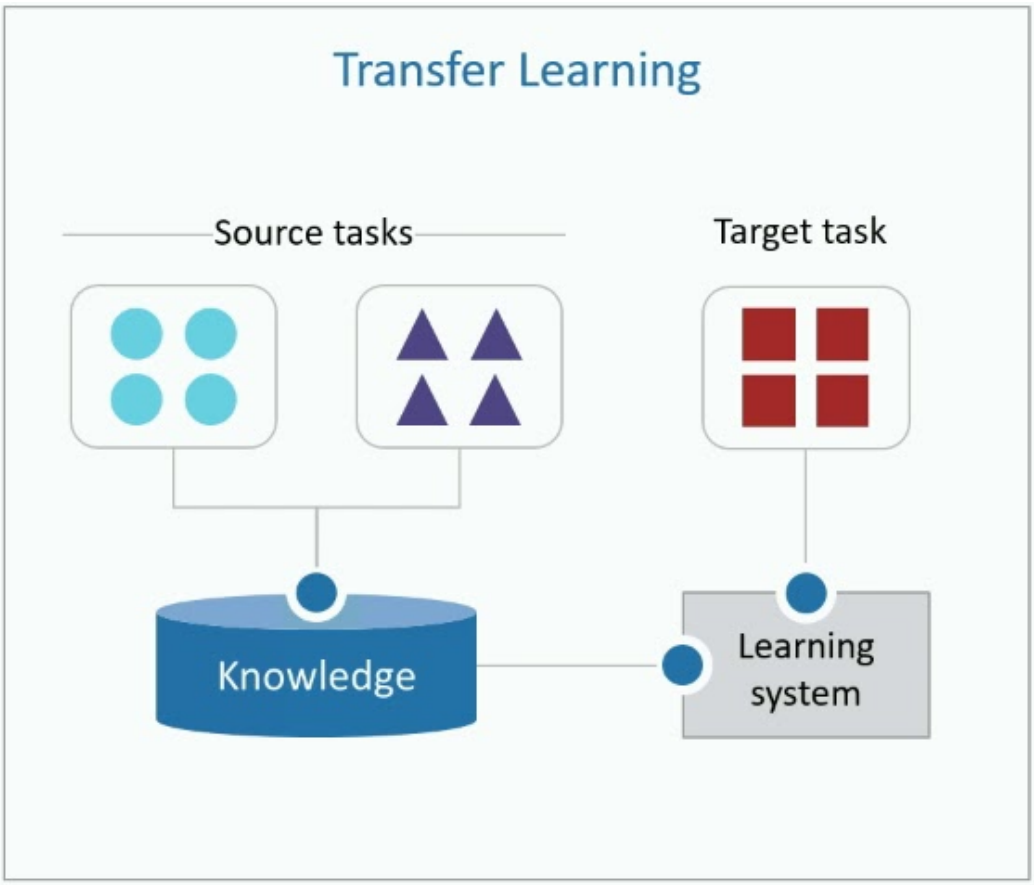

Figure 2.8: An illustration of the transfer learning process [23, 15].

As deep learning models are usually data-hungry whereas getting enough labeled data for supervise learning is oftentimes infeasible. Thus use a language model that is pre-trained using a large corpus as basis and fine-tune such model for specific task becomes the default approach across various NLP tasks. Like what we described in earlier section, ELMo and BERT well present such concept.

2.4 Dialogue State Tracking

The goal of dialogue state tracking is to keep track of the user goal, so that dialogue systems accomplish task according to the user's needs. The initial motivation for dialogue state tracking came from the uncertainty such as homophones in speech recognition [28], as well as to provide a comprehensive

input to a downstream dialogue policy component deciding the next system action. Proposed belief tracking models have ranged from rule-based [27], to generative [25]. and discriminative [8]. The common drawback of such approaches is that they rely on hand-crafted features and complex domain-specific lexicons (besides the ontology), and are difficult to extend and scale to new domains.

To overcome these problems, Mrksic et al. propose a Neural Belief Tracking framework which utilize pre-trained word vectors to learn the semantics of user utterance and dialogue context [14]. Zhong et. al use global modules to share parameters between estimators for different slots, use local modules to learn slot-specific features, so to improve the performance on rare states [33]. Ren et al. propose StateNet that generates a representation for dialogue history and compares the distances between this representation and value vectors in the candidate set [21]. However, these approaches often require a predefined domain ontology, which makes it impractical for dialogue systems that operate over real-world services having a large number and fairly dynamic set of possible values.

To address these concerns, Wu et al. propose a TRAnsferable Dialogue statE generator that generates dialogue states from utterances using a copy mechanism, facilitating knowledge transfer when predicting (domain, slot, value) triplets not encountered during training [30]. On the other hand, Chao and Lane focus on a specific condition, where the ontology is unknown to the state tracker, but the target slot value can be located in the dialogue context, they use BERT as dialogue context encoder to extract the value [4]. To the best of our knowledge, it is the most similar setting and approach to ours, whereas the difference lies on the inputs.

For what concerns schema-guided dialogue state tracking challenge, Rastogi et al. propose a baseline model [20] inspired by BERT-DST [4], two kinds of embeddings are obtained separately, an embedded representation of intents and slots using each service schema and the representation of user utterance and preceding system utterance. The utterance and schema embeddings are then used together to predict the dialogue state through a set of projection. Meanwhile the best performed model is proposed by Ma et al., their framework consists of a machine reading comprehension module based on XLNet [32] for non-categorical slot prediction and a RoBERTa [10] based model with hand-craft features for categorical slot prediction [11]. They encode the full dialogue history as well as natural language description of slots and intents. Moreover, data augmentation by back translation between English

and Chinese significantly improve the overall performance.

Chapter 3

Dataset

In this chapter, we describe the dataset. First, we give the information about the schema-guided dataset in all aspects. Then we go into the details of data pre-processing, explaining how we reconstruct the dialogue structure.

3.1 Dataset Overview

Schema-Guided Dialogue dataset, to the best of our knowledge, is the largest publicly available corpus of annotated task-oriented dialogues. There are more than 16000 dialogues spanning over 26 services within 16 domains in the training set (the detailed statistics is in Table 3.1). Furthermore, many unseen services and domains are in the evaluation sets, consequently slots which are not present in the training set ideally serve the purpose to test the model's ability to generalize in zero-shot settings.

The dataset consists of conversations between a virtual assistant and a user, these conversations can be categorized into two types: single-domain and multi-domain dialogues. The single domain dialogues associate with interactions in a single service from one domain, whereas in the multi-domain dialogues, interactions possibly span two or more different services across different domains. Furthermore, the transfer of dialogue state values from one service to another is also involved as long as it is necessary, for instance, if a user asks for a restaurant reservation and then asks for the direction, the dialogue state of location slot in navigation service is already initialized by the one for restaurant service.

Dialogue Representation

The dialogue is represented by a set of turns, where each turn contains either a user or a system utterance. The annotations of a turn are grouped into frames, each frame is corresponding to a single service, an example is shown in Figure 3.1. Hence, in the single-domain dialogues, each turn contains exactly one frame, however, multiple frames may exist in a turn within the multi-domain dialogues.

Each dialogue is a json object contains the following fields.

- ***dialogue_id***: an unique identifier for a dialogue.
- ***services***: a list of services present in the dialogue.
- ***turns***: a list of annotated system or user utterances.

Each turn contains following fields.

- ***speaker***: the speaker of the turn, either "USER" or "SYSTEM".
- ***utterance***: the utterance in natural language.
- ***frames***: a list of frames, each consists of annotations for a single service.

Each frame contains different field according to its speaker.

- ***service***: the name of service corresponding to the frame.
- ***slots***: a list of slot spans in the utterance, each consists of:
  - *slot*: the name of the slot.
  - *start*: the index of the starting character.
  - *exclusive_end*: the index of the character just after the ending character.
- ***actions***(system turns only): a list of actions, each consists of:
  - *act*: the type of action.
  - *slot*(optional): the name of the slot for the action.
  - *value*(optional): a list of values assigned to the slot.
- ***state***(user turns only): the dialogue state till current turn, it consists of:

- *active_intent*: the intent corresponding the service.
- *requested_slots*: a list of slots requested by the user.
- *slot_values*: a dictionary of slot-value pairs.

Schema Representation

In addition of the dialogues, each service used in the dataset provides a schema listing the supported intents and slots along with their descriptions in natural language, an example is shown in Figure 3.2. Each schema consists of following fields.

- ***service_name***: a unique service name.
- ***description***: a natural language description of the tasks supported by the service.
- ***slots***: a list of slots/attributes in the service, each consists of:
  - *name*: the name of the slot.
  - *description*: a natural language description of the slot.
  - *is_categorical*: a Boolean value.
  - *possible_values*: list of possible values of the slot.
- ***intents***: a list of intents/tasks in the service, each consists of:
  - *name*: the name of the intent.
  - *description*: a natural language description of the intent.
  - *is_transactional*: a Boolean value.
  - *required_lots*: a list of slot names whose values must be provided before making a call to the service.
  - *optional_lots*: a dictionary mapping slots and their default values.

```
{
  "dialogue_id": "13_00000",
  "services": [
    "Movies_1"
  ],
  "turns": [
    {
      "frames": [
        {
          "service": "Movies_1",
          "slots": [],
          "state": {
            "active_intent": "FindMovies",
            "requested_slots": [],
            "slot_values": {}
          }
        }
      ],
      "speaker": "USER",
      "utterance": "A movie will help me overcome the monotonous of the day. Is there any good movie?"
    },
    {
      "frames": [
        {
          "actions": [
            {
              "act": "REQUEST",
              "slot": "location",
              "values": []
            }
          ],
          "service": "Movies_1",
          "slots": []
        }
      ],
      "speaker": "SYSTEM",
      "utterance": "Where do you want to see?"
    },
    {
      "frames": [
        {
          "service": "Movies_1",
          "slots": [
            {
              "exclusive_end": 22,
              "slot": "location",
              "start": 19
            }
          ],
          "state": {
            "active_intent": "FindMovies",
            "requested_slots": [],
            "slot_values": {
              "location": [
                "SFO"
              ]
            }
```


Figure 3.1: Example dialogue.

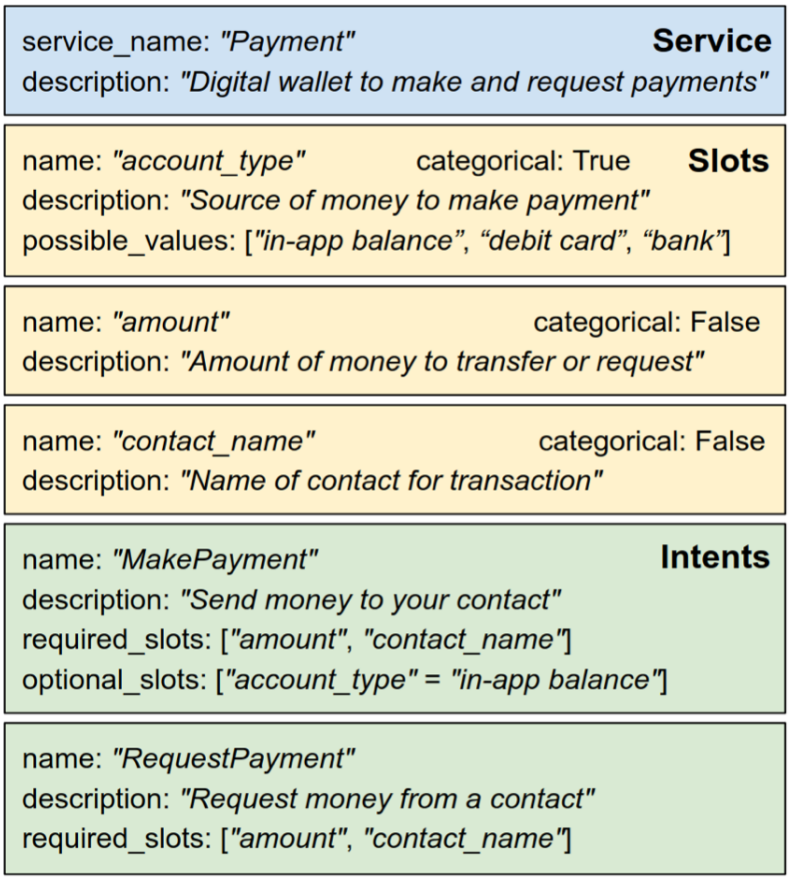

Figure 3.2: Example schema for a digital wallet service [20].

Dataset Statistics

The overall statistics of the train and dev sets are given in Table 3.1. In the train set, the single domain dialogues have an average of 15.3 turns whereas 23 turns on an average for the multi-domain dialogues. The detailed distribution of dialogues in the train and dev sets among different domains is given in Table 3.2. Note that Alarm domain is only present in the dev set, meanwhile in Banks, Events, Flights, Hotels, Media, Movies, Restaurants, and Services domains, the services in train set are different than those in the dev set. This is to test the generalization of models on unseen domains and services, in a more former way, this is the zero-shot dialogue state tracking task. The number in parenthesis indicates the unique services belonging to the corresponding domain.

	Train			Dev		
	Single-domain	Multi-domain	Combined	Single-domain	Multi-domain	Combined
No. of dialogues	5403	10739	16142	836	1646	2482
No. of turns	82588	247376	329964	11928	36978	48726
Avg. turns per dialogue	15.286	23.035	20.441	14.268	22.356	19.632
Avg. tokens per turn	9.778	9.741	9.751	9.85	9.603	9.664
Total unique tokens	16353	25459	30352	6803	10533	12719
Total domains	14	16	16	16	15	16
Total services	24	26	26	17	16	17
Total no. of slots	201	214	214	134	132	136

Table 3.1: Schema-Guide Dialogue dataset statistics [20].

3.2 Data Pre-processing

It is necessary to carry out data manipulation in order to proceed. We reconstruct the original dialogues into a set of conversation turns. Every conversation turn consists of an user utterance and its preceding system utterance as the dialogue context. And each turn is corresponding to a single service and an active intent. That being said, there are turns associated with multiple services exist in multi-domain dialogues, in such case more than one conversation turn will be generated, whereby each of them consists of the same text but different service, and they also share the same dialogue_id and turn_id, while can be differentiate by the _id. The service item is used as a key word for retrieving the natural language description from the schema for slot whichever belongs to it, so to form the input sequence. Other annota-

	Train			Dev		
	Single-domain	Multi-domain	Combined	Single-domain	Multi-domain	Combined
Alarm (1)	NA	NA	NA	37	NA	37 (1)
Banks (2)	207	520	727 (1)	42	252	294 (1)
Buses (2)	310	1970	2,280 (2)	44	285	329 (1)
Calendar (1)	169	1433	1,602 (1)	NA	NA	NA
Events (2)	788	2721	3,509 (1)	73	345	418 (1)
Flights (3)	985	1762	2,747 (2)	94	297	391 (1)
Homes (1)	268	579	847 (1)	81	99	180 (1)
Hotels (4)	457	2896	3,353 (3)	56	521	577 (2)
Media (2)	281	832	1,113 (1)	46	133	179 (1)
Movies (2)	292	1325	1,617 (1)	47	94	141 (1)
Music (2)	394	896	1,290 (2)	35	161	196 (1)
RentalCars (2)	215	1370	1,585 (2)	39	342	381 (1)
Restaurants (2)	367	2052	2,419 (1)	73	263	336 (1)
RideSharing (2)	119	1584	1,703 (2)	45	225	270 (1)
Services (4)	551	1338	1,889 (3)	44	157	201 (1)
Travel (1)	NA	1871	1,871 (1)	45	238	283 (1)
Weather (1)	NA	951	951 (1)	35	322	357 (1)

Table 3.2: No. of dialogues belongs to each domain in SGD dataset. In the first column, it indicates the number of unique services for the domain in Train and Dev datasets combined. In the fourth column, it indicates the number of such unique services in the Train dataset only. In the last column, it indicates the number of such unique services in the Dev dataset only [20].

tions including slots and belief. Slots represents the dialogue state update, whereas belief represents the constant slot values of previous and current turns. Figure 3.1 and Figure 3.3 depict the same dialogue before and after data pre-processing. Consequently, the processed data are then tokenized and converted to tensors that can be used by BERT, the details is discussed in the next chapter.

```
{
  "_id": "13_00000_00",
  "dialogue_id": "13_00000",
  "turn_id": "00",
  "text": "A movie will help me overcome the monotonous of the day . Is there any good movie ?",
  "service": "Movies_1",
  "intent": "FindMovies",
  "slots": {},
  "belief": {},
  "sys_utt": "",
  "usr_utt": "A movie will help me overcome the monotonous of the day. Is there any good movie?"
},
{
  "_id": "13_00000_01",
  "dialogue_id": "13_00000",
  "turn_id": "01",
  "text": "Where do you want to see ? ; It will be fine in SFO",
  "service": "Movies_1",
  "intent": "FindMovies",
  "slots": {
    "location": "SFO"
  },
  "belief": {},
  "sys_utt": "Where do you want to see?",
  "usr_utt": "It will be fine in SFO"
}
```


Figure 3.3: An example of the same dialogue shown in Figure 3.1 after data preprocessing. Each turn is corresponding to a single service and an active intent. Text is composed of system utterance and user utterance. Slots represents the dialogue state update, while belief represents the constant slot value pairs from previous turn.

Chapter 4

Approach

In this chapter, we describe in full details the proposed model, first the overall architecture is introduced, followed by all the individual modules.

4.1 Model Overview

The proposed model architecture is illustrated in Figure 4.1, it consists of a pretrained multilingual $\text{BERT}_{\text{BASE}}$ as the encoder and a classification module and a span prediction module. The input sequence is a pair of sentences where the former is the dialogue context, and the latter is the description of a slot in natural language. At each turn, all slots from the service that current dialogue context belongs to, their respective descriptions will be used to construct the input sequence one after another, as it is shown in Table 4.1.

	Sequence A	**Sequence B**
Input Sequences	dialogue context	slot^s description ($\forall s \in S$)

Table 4.1: All input sequences of a conversation turn, where S is the complete set of slots at the turn.

The proposed model is inspired by two tasks, namely BERT for Next Sentence Prediction and BERT for Question Answering. We decouple the dialogue state tracking problem into a Next Sentence Prediction like task and a Question Answering task. Each input sequence is encoded by the encoding module to obtain contextualized sentence-level and token-level representations. The sentence-level representation is then used by the classification module to produce a Boolean value indicating whether the value of the described slot is present in the dialogue context. Likewise, the token-level representations are used by the span prediction module to generate the start

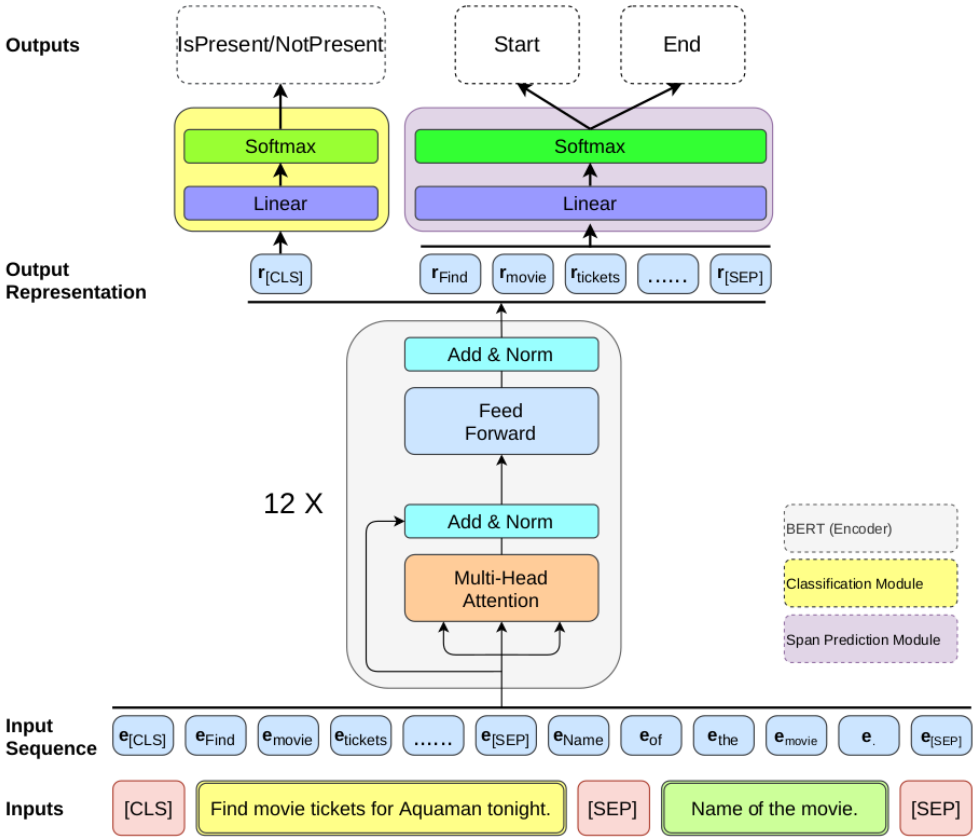

Figure 4.1: Architecture of proposed model, it consists of a BERT encoder, a classification module and a span prediction module. Each element in the input sequence is constructed as the sum of its corresponding token, segment and position embeddings.

and end positions for the slot value. In the end, the dialogue state update mechanism derives the output. Such procedure iterates over all the input sequences at each conversation turn, yield the final outputs.

4.2 Encoding Module

We use BERT as the encoder in our proposed model to obtain contextualized representations. To comply with BERT input format, the elements in the input sequence are first tokenized, then summed by their corresponding segment and position embeddings to form the input representation, as shown in Figure 4.2.

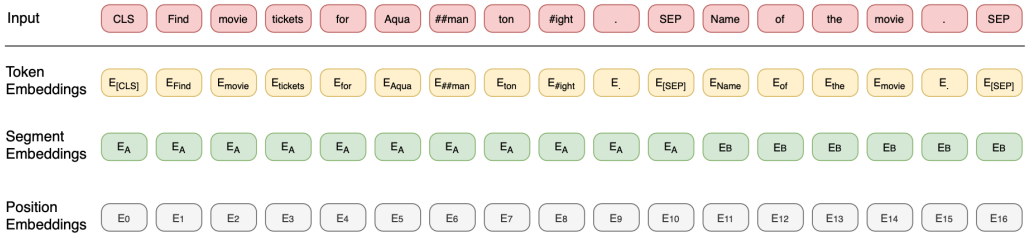

Figure 4.2: Input representation for the proposed model.

Let $X = [x_0, ..., x_n, x_{n+1}, ..., x_m]$ denotes the entire input token sequence, in which $[x_0, ..., x_n]$ belongs to dialogue context, whereas $[x_{n+1}, ..., x_m]$ belongs to slot description, the corresponding embedding of each token x_i in BERT embedding layer is presented as

$$\mathbf{e_i} = E_{token}(x_i) + E_{segment}(i) + E_{position}(i), \quad \forall i \in [0, ..., n, ..., m] \tag{4.1}$$

where $E_{token}(x_i)$ is the WordPiece [31] embedding for token x_i, $E_{segment}(i) \in \{E_A, E_B\}$ is the segement embedding whose value depends on whether token x_i is an element of dialogue context or an element of slot description, and $E_{segment}(i)$ is the position embedding whose value is the absolute position in the sequence, i.e. equal to the value of i.

We obtain the last hidden state at BERT output layer, the contextualized representations

$$\mathbf{u_{CLS}}, [t_1, ..., t_n, ..., t_m] = BERT([e_0, ..., e_m]) \tag{4.2}$$

where the first output token $\mathbf{u_{CLS}}$ is the sentence-level representation, and remaining ones $[t_1, ..., t_m]$ are token-level representations.

$\mathbf{u_{CLS}}$ is then passed to classification module, whereas $[t_1, ..., t_n]$ are feed into span prediction module for the subsequent tasks respectively.

4.3 Classification Module

We use a classification module to perform a binarized Next Sentence Prediction like task, it trains the model to learn the relation between the dialogue context and the described slot.

The input is initialized by contextualized sentence-level embedding $\mathbf{u_{CLS}}$ from encoding module, a linear layer is used to obtain the state score, softmax is then applied to produce a probability distribution. The classification module predicts the state value to be either 1 or 0 that indicates whether the value of the described slot is present in the dialogue text.

$$\mathbf{z} = \mathbf{W_{cls}}\, \mathbf{u_{CLS}} + \mathbf{b_{cls}} = [z_{present},\ z_{not\_present}] \tag{4.3}$$

$$\mathbf{p} = softmax(\mathbf{z}) = [p_{present},\ p_{not\_present}] \tag{4.4}$$

$$state = argmax(\mathbf{p}) \tag{4.5}$$

4.4 Span Prediction Module

We use span prediction module to perform question answering task, work in a reading comprehension setting whereby the question is the slot description, while the answer may be found in the dialogue context. In such a way to obtain the slot value by means of start and end positions.

The input are contextualized token-level representations $\mathbf{t_i}$ from encoding module, and linearly projected to obtain start and end span scores. Softmax is then applied respectively to produce the probability distribution over all input tokens. The span prediction module outputs two values indicating start span and end span of the slot value.

$$[\alpha_i, \beta_i] = \mathbf{W_{span}}\, \mathbf{t_i} + \mathbf{b_{span}}, \quad \forall i \in [1, 2, ..., n] \tag{4.6}$$

$$\mathbf{p}_\alpha = softmax(\alpha_i) \tag{4.7}$$

$$\mathbf{p}_\beta = softmax(\beta_i) \tag{4.8}$$

$$start = argmax(\mathbf{p}_\alpha) \tag{4.9}$$

$$end = argmax(\mathbf{p}_\beta) \tag{4.10}$$

4.5 Dialogue State Update Mechanism

We deploy a dialogue state update mechanism to combine the outputs from classification and span prediction modules. Whenever the predicted state takes value 1, the corresponding start and end spans will be the final output, otherwise such spans are disregard and the default padding span will be assigned as final output, in our case [-100, -100].

This procedure is applied to every input sequence, thus all input sequences iterate such process, yield a number of outputs for each conversation turn. To be specific, the number is equal to the number of slots available in the service that current dialogue context belongs to.

Algorithm 1 Dialogue State Update for each input sequence

Input:

1. Target slot state value: state;
2. Target slot start and end span: [start, end]

Output:

Target slot value: output

if state == 1 **then**
  return output = [start, end]
else
  return output = [-100, 100]
end if

Chapter 5

Experimental Setup

In this chapter, we first analyze the post-processed, ready-executed data, followed by the implementation details, next we brief the framework and libraries used in the code development, and finally introduce the execution environment.

5.1 Data Analysis

As reported in dataset chapter, there are total 26 services and 214 slots in the training set, so every service has an average of 8 slots. Given the method in approach chapter how we construct the input sequences, every conversation turn will produce about 8 input sequences. However, in each turn, not all the slot values exist in the dialogue context, sometimes none of the slot values present as well. It leads to an imbalanced data, actually, after data pre-processing, we have a total number of 1601356 input data, 1477493 out of 1601356 associated without any slot, only 123863 inputs are associated with slot value. It means for the prediction of state value in the classification module, more than 92% of its ground truth outputs are 0, only less than 8% should produce the value of 1.

To handle the imbalanced problem in classification, one possible solution is to introduce a weighted sampling strategy

1. Oversampling the minority class
2. Undersampling the dominant class

Both have its own drawback: the former may increases the possibility of overfitting whereas the latter potentially discard useful information since it

does not iterate over all the samples.

In our approach, instead of weighted sampling, we introduce a weighted loss function [16].

$$\text{loss}(x, class) = weight[class] \left(-x[class] + \log \left(\sum_j \exp(x[j]) \right) \right)$$

where $class$ is the ground truth, $j \in [0, ..., C]$ for all the possible values, in our case $j \in [0, 1]$.

According to the distribution in the training set, we assign a rescaling weight to each class which is inverse to its size. To penalize more if a prediction to a minority class is wrong, in a sense of introducing the bias to coincide with actual situation in our input data.

5.2 Implementation details

We use pretrained BERT-base-multilingual-cased model, which has 12 layers, hidden size 768, 12 multi-attention heads, total 110M parameters, it is trained on cased text in the top 104 languages with the largest Wikipedias. We use weighted cross-entropy loss function for state prediction in the classification module, and cross-entropy loss function for span prediction. Meanwhile in the training, according to the ground truth value of state, whenever it is equal to 0 the corresponding span loss is set to be 0 as well. The total loss is the sum of state loss and span loss. We apply Adam optimization to update all layers in the model with an initial learning rate of $5e^{-5}$. A dropout rate of 0.1 is applied to encoding module's sentence-level output. Batch size is set to 32, at training time, inputs are randomly chosen from the whole input data, whereas at evaluation, inputs are taken sequentially.

5.3 Framework and Libraries

The code is developed in pytorch [17], the pretrained BERT model is downloaded from PyTorch-Transformers [29], a library of state-of-the-art pretrained models for Natural Language Processing (NLP) maintained by huggingface team.

5.4 Execution Environment

The training is executed on Legion cluster in HPC @ POLITO. The HPC @ POLITO supercomputing initiative, managed by DAUIN (Department of Automation and Computer Science of the Polytechnic of Turin) provides high-performance computing resources and technical support for academic and teaching research. The service is aimed both at internal research groups and at institutions and entities outside the Polytechnic. LEGION was born to be modular and grow over time well beyond the size of a single rack cabinet, this system is currently the most recent among those available in HPC @ POLITO. It is an InfiniBand cluster of over 21 TFLOPS with the characteristics listed in Figure 5.1.

Architecture	Linux Infiniband-EDR MIMD Distributed Shared-Memory Cluster
Node Interconnect	Infiniband EDR 100 Gb / s
Service Network	Gigabit Ethernet 1 Gb / s
CPU Model	2x Intel Xeon Scalable Processors Gold 6130 2.10 GHz 16 cores
GPU Model	8x nVidia Tesla V100 SXM2 - 32 GB - 5120 cuda cores
Sustained performance (Rmax)	**21.094 TFLOPS** (last update: july 2019)
Peak performance (Rpeak)	**27.238 TFLOPS** (last update: july 2019)
Computing Cores	448
Number of Nodes	14
Total RAM Memory	5,376 TB DDR4 REGISTERED ECC
OS	Centos 7 - OpenHPC 1.3
Scheduler	slurm

Figure 5.1: The characteristics of Legion cluster.

Chapter 6

Results and Discussion

We evaluate the proposed model performance with following metrics:

- Average Goal Accuracy: average accuracy of predicting the value of a slot for a turn correctly.
- Joint Goal Accuracy: average accuracy of predicting the value of all the slots for a turn correctly.

6.1 Results

Performance on SGD dataset

Table 6.1 presents the performance of our proposed model compared to the baseline model on the dev set of the entire(combined) dataset. In general our model outperforms the baseline model for both average goal accuracy and joint goal accuracy. And as expected, the performance on seen services are better than unseen services, it is shown in Figure 6.1.

Model	Avg GA	Joint GA
Baseline	0.694	0.383
Our Model	0.826	0.415

Table 6.1: Model performance on dev combined set.

Performance on different domains

Table 6.2 presents the model performance on different domains. Likewise,

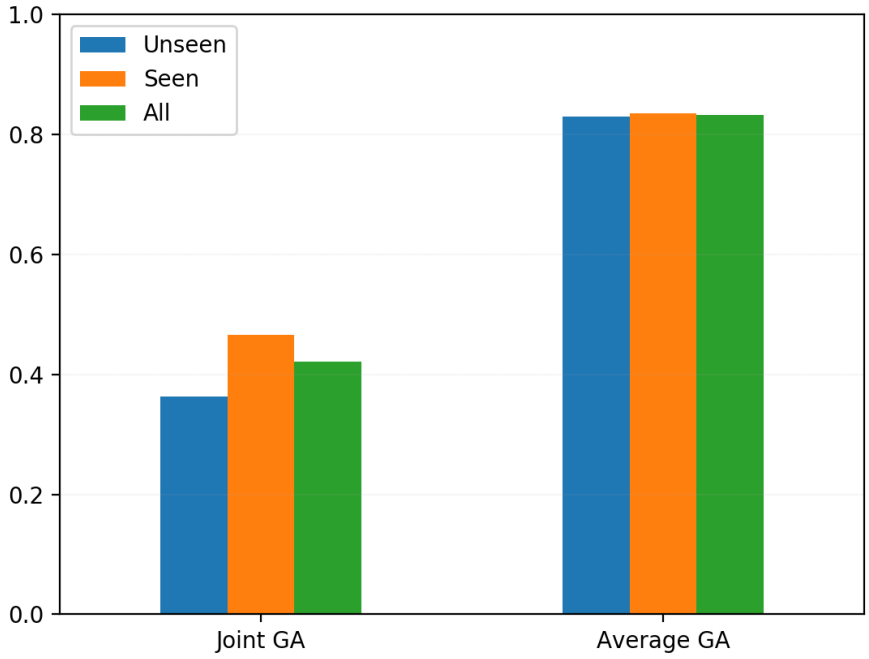

Figure 6.1: Model performance on dev combined set per all services, services seen in the training, and services unseen in the training.

the performance for each domain largely depends on the presence of services in the training set. Among unseen services, high out-of-vocabulary rate for slots in Flights and Movies results to a poor performance.

Domain	Average GA	Joint GA
Flights*	0.793	0.192
Movies*	0.529	0.266
Buses	0.838	0.315
Music	0.647	0.355
Banks*	0.800	0.359
Media*	0.668	0.363
Hotels**	0.838	0.376
Events	0.800	0.383
RentalCars	0.817	0.386
Alarm*	0.633	0.419
Restaurants*	0.902	0.432
Services*	0.863	0.436
Homes	0.908	0.535
RideSharing	0.889	0.649
Weather	0.948	0.759
Travel	0.965	0.804

Table 6.2: Model performance per domain. Domains marked with '*' are those for which the service in the dev set is not present in the training set. Hotel domain marked with '**' has one unseen and one seen service. For other domains, the service in the dev set was also seen in the training set.

6.2 Discussion

To further investigate, we would like to check the performance for individual modules. Recall that at every turn, the output for a given slot is either the estimation from span prediction module, or a default padding value, according to its corresponding state value. i.e. the output of classification module. Therefore by checking the performance of classification module allow us to understand where the bottleneck is.

As discussed in section data analysis earlier, the input data is imbalanced in training set, such imbalanced distribution remains in dev set, therefore accuracy is not a good metrics for evaluation, because the majority class 0 is

92%, even if the classifier simply outputs 0, without any learning process it achieves 92% accuracy, which makes no sense as the model would not predict any slot value at all. Instead, we use precision, recall and f1 score to evaluate the performance of this classifier. We calculate the confusion matrix first, it is shown in Figure 6.2.

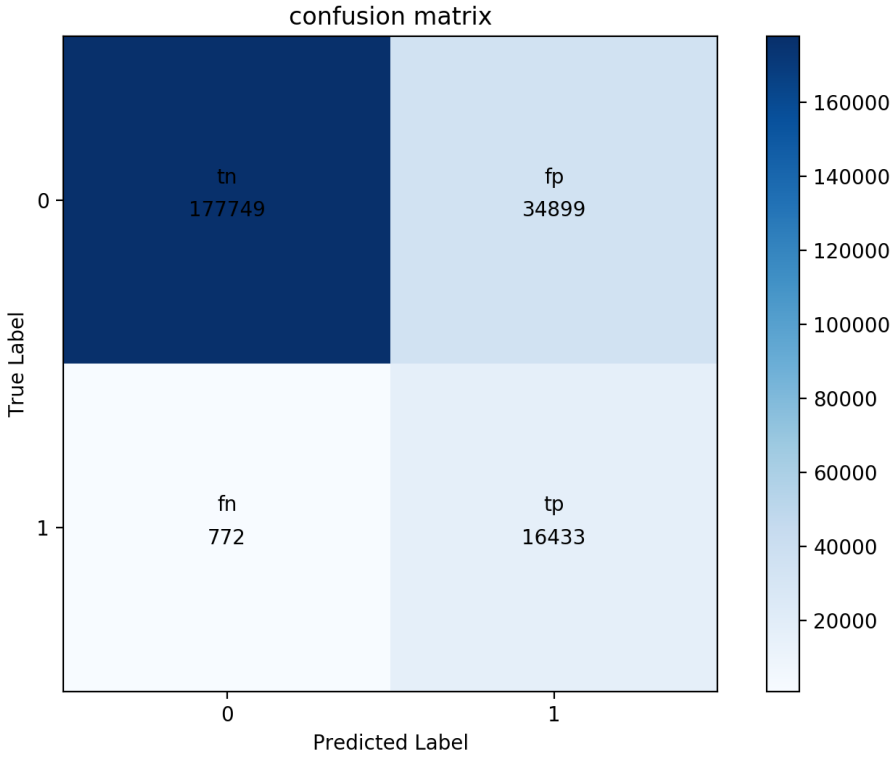

Figure 6.2: Confusion matrix for the classification module.

Then we can calculate the metrics according to following formula

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Table 6.3 presents the result, the classifier achieves a low precision score only at 0.320, a relatively good recall score at 0.955, leads to 0.480 for F1 measure.

Precision	Recall	F1
0.320	0.955	0.480

Table 6.3: Classification module performance on the entire dev sets.

A classifier with high recall but low precision indicates it often wrongly classifies a negative sample as positive, though it is able to identify most of the positive samples. In our case, it reveals the classification module mistakenly treat the slot whose value not present in the dialogue as present. Therefore, in order to improve the joint accuracy for the whole system, we need to further study the classification module, help it better correlate a dialogue and the described slot.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

We present our proposed approach, a Schema-Guided, BERT-based dialogue state tracker, to address the zero-shot dialogue state tracking challenge. Rather than treating the slots as labels, the model extracts slot value from the dialogue context by interpreting the semantics of the slots. Without any domain specific parameters, the model size is consistent no matter how many slots exist. Thanks to pretrained BERT model, the evaluation results prove its contextualized representation outputs allow our approach to be effective. Further, it demonstrates transfer learning with language models has become an integral part of many language understanding problems.

7.2 Future Works

Nevertheless, a lot of work need to be done to further improve the performance. Needless to say, the first is to work on the classification module. Besides, we would like to highlight another three out of many for our future work.

Long-range dependency as mentioned in the task description, our model is applied to a special setting where the target slot value can be found as word segment in the dialogue context. Since the input sequence only consists of dialogue context within the current conversation turn, in the case where slot value mentioned beyond current turn, such setup inevitably miss shots. One possible solution is to encode the whole dialogue history instead of current turn. Another solution we consider is to introduce a so-called system belief to the model, which is essentially a dictionary contains slot-value pairs we

encountered along the conversation. Then we can modify the output layer in the classification module, add one value indicates the target slot presented in the system belief. So that we can perform another classification over all the slots presented in such belief to decide the final output.

Slot transfer this is another area we need to dive into for the performance improvement. In multi domain dialogues, slot value in a service maybe initialized already by another slot in previous service. For example, after user requests the restaurant reservation, he may also asks for the direction, in such case the value of destination slot in navigation service is actually initialized by the value of address slot in restaurant service. Thus our model is unable to capture the value span if the address is beyond current turn. Similar system belief methodology mentioned earlier can be used to tackle this problem.

Integration with intent detection last but not least, we also consider developing a joint model to perform intent detection and dialogue state tracking simultaneously. Such integration also worth the effort.

References

[1] Jay Alammar. *The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)*. 2018. URL: http://jalammar.github.io/illustrated-bert/.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2014).

[3] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5 (2016), pp. 135–146.

[4] Guan-Lin Chao and Ian Lane. "BERT-DST: Scalable End-to-End Dialogue State Tracking with Bidirectional Encoder Representations from Transformer". In: *ArXiv* abs/1907.03040 (2019).

[5] Francois Chaubard et al. *Word Vectors I: Introduction, SVD and Word2Vec*. URL: https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes01-wordvecs1.pdf.

[6] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT*. 2019.

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[8] Matthew Henderson, Blaise Thomson, and Steve J. Young. "Word-Based Dialog State Tracking with Recurrent Neural Networks". In: *SIGDIAL Conference*. 2014.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (1997), pp. 1735–1780.

[10] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *ArXiv* abs/1907.11692 (2019).

[11] Yue Ma et al. “An End-to-End Dialogue State Tracking System with Machine Reading Comprehension and Wide & Deep Classification”. In: *ArXiv* abs/1912.09297 (2019).

[12] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *NIPS*. 2013.

[13] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *CoRR* abs/1301.3781 (2013).

[14] Nikola Mrksic et al. “Neural Belief Tracker: Data-Driven Dialogue State Tracking”. In: *ArXiv* abs/1606.03777 (2016).

[15] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 22 (2010), pp. 1345–1359.

[16] Adam Paszke et al. *CrossEntropyLoss*. URL: https://pytorch.org/docs/stable/nn.html#torch.nn.CrossEntropyLoss.

[17] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[18] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “Glove: Global Vectors for Word Representation”. In: *EMNLP*. 2014.

[19] Matthew E. Peters et al. “Deep contextualized word representations”. In: *ArXiv* abs/1802.05365 (2018).

[20] Abhinav Rastogi et al. “Towards Scalable Multi-domain Conversational Agents: The Schema-Guided Dialogue Dataset”. In: *ArXiv* abs/1909.05855 (2019).

[21] Liliang Ren et al. “Towards Universal Dialogue State Tracking”. In: *EMNLP*. 2018.

[22] Sebastian Ruder et al. “Transfer Learning in Natural Language Processing”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*. 2019, pp. 15–18.

[23] Mark Russinovich. “Transfer Learning: Repurposing ML Algorithms from Different Domains to Cloud Defense”. RSA Conference. 2018. URL: https://www.rsaconference.com/industry-topics/presentation/transfer-learning-repurposing-ml-algorithms-from-different-domains-to-cloud-defen.

[24] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *NIPS*. 2014.

[25] Blaise Thomson and Steve J. Young. “Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems”. In: *Comput. Speech Lang.* 24 (2010), pp. 562–588.

[26] Ashish Vaswani et al. “Attention is All you Need”. In: *NIPS*. 2017.

[27] Zhuoran Wang and Oliver Lemon. “A Simple and Generic Belief Tracking Mechanism for the Dialog State Tracking Challenge: On the believability of observed information”. In: *SIGDIAL Conference*. 2013.

[28] Jason D. Williams and Steve J. Young. “Partially observable Markov decision processes for spoken dialog systems”. In: *Comput. Speech Lang.* 21 (2007), pp. 393–422.

[29] Thomas Wolf et al. “HuggingFace’s Transformers: State-of-the-art Natural Language Processing”. In: *ArXiv* abs/1910.03771 (2019).

[30] Chien-Sheng Wu et al. “Transferable Multi-Domain State Generator for Task-Oriented Dialogue Systems”. In: *ACL*. 2019.

[31] Yonghui Wu et al. “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”. In: *ArXiv* abs/1609.08144 (2016).

[32] Zhilin Yang et al. “XLNet: Generalized Autoregressive Pretraining for Language Understanding”. In: *NeurIPS*. 2019.

[33] Victor Zhong, Caiming Xiong, and Richard Socher. “Global-Locally Self-Attentive Encoder for Dialogue State Tracking”. In: *ACL*. 2018.