CS 224N - Project Report
Compositional Pre-Training for Semantic Parsing with BERT

Arnaud Autef
Stanford Unversity
arnaud15@stanford.edu

Simon Hagege
Stanford Unversity
hagege@stanford.edu

Abstract

Semantic parsing - the conversion of natural language utterances to logical forms - is a typical natural language processing task. Its applications cover a wide variety of topics, such as question answering, machine translation, instruction following or regular expression generation. In our project, we investigate the performance of Transformer-based sequence-to-sequence models for semantic parsing. We compare a simple Transformer Encoder-Decoder model built on the work of [4] and a Encoder-Decoder model where the Encoder is BERT [1], a Transformer with weights pre-trained to learn a bidirectional language model over a large dataset. Our architectures have been trained on semantic parsing data using data recombination techniques described in [2]. We illustrated how Transformer-based Encoder Decoder architectures could be applied to semantic parsing. We evidenced the benefits of BERT as the Encoder of such architectures and pointed to some of the difficulties in fine-tuning it on limited data while the Decoder has to be trained from scratch. We have been mentored by Robin Jia from Stanford University's Computer Science Department.

1 Introduction

During this project, we built several models to solve the specific semantic parsing task defined on the GeoQuery dataset. On GeoQuery, the goal is to transform natural language questions about geography to corresponding logical queries to a geographical database.

This reference dataset in semantic parsing is particularly challenging as it enforces to make sure a neural natural language processing model respect the exact syntax of the logical queries. In addition, the dataset is made of only a thousand training examples, which reduces the complexity of the models we can explore.

In [2], Robin Jia, our supervisor, proposed an innovative technique to overcome the limited amount of data available on GeoQuery: *data recombination*. Data recombination is a specific form a data augmentation which focuses on creating new training examples by recombination of existing examples and the syntax of natural language questions and queries.

In his paper, Robin Jia trains a Recurrent Neural Network (RNN) based Encoder Decoder model. In our work, we investigate the performances of more recent neural architectures for natural language processing: Transformers. RNN-based architectures are good at learning directional representations of natural language forms, while Transformers are also able to learn bidirectional representations. Bidirectional representations look particularly relevant to queries in the GeoQuery dataset and motivate our approach. Moreover, the lack of available data in GeoQuery makes the use of a pre-trained natural language model appealing, and we decided to study the benefits from using BERT [1] in our models.

We trained two models, a simple Transformer Encoder-Decoder architecture and a Transformer Encoder-Decoder with a pre-trained BERT Transformer as the Encoder. We carried out a comparative study of their performances on GeoQuery across different architecture choices, training and fine-tuning procedures with data recombination, and hyperparameters. Eventually, we present a model leading to very positive results on the test set.

2 Related Work

Data recombination pre-processing [2] and the introduction of the Transformer deep learning model in [4] followed by the state-of-the-art BERT technique [1] for sentence encoding offer powerful tools to improve performances on semantic parsing tasks.

Data recombination One challenge of semantic parsing is building logical properties that model conditional independence. In many cases, part of the meaning of a sentence (consider *what states border Texas?*) is independent of some words of the sentence (here *Texas* is replaced by any other state, the type of question remains the same). Data recombination provides a framework to model these invariances and introduces a generative model from the training data to augment the dataset. It leverages synchronous context-free grammars (SCFG), which are sets of production rules $X \rightarrow \langle \alpha, \beta \rangle$ where X is a category (non-terminal) and α and β are sequences of terminal and non-terminal symbols. Using a composition of the three rules presented in [2], Jial et al. achieved a state-of-the-art test accuracy of 89.3 on the GeoQuery dataset.

Transformers The Transformer architecture, introduced in [4] offers a model exclusively relying on a new self-attention mechanism, called multi-head attention, in order to draw dependencies between the input and the output. This allows for significantly more parallelization and achieves state-of-the-art results in translation tasks. Originally proposed as a sequence-to-sequence model, the whole architecture was composed of two Transformer blocks, an Encoder and a Decoder, with auto-regressive in the Decoder, as previously generated symbols become additional input when generating the next. The structure can be seen in the figure below.

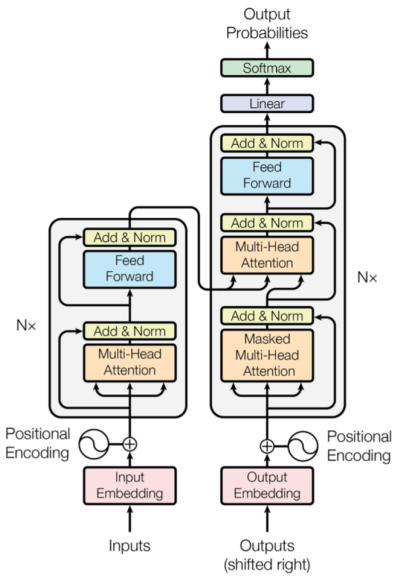


Figure 1: The transformer architecture - figure from [4]

BERT As introduced in [1], BERT is a language representation model that can be applied to a variety of natural language processing tasks. It offers a language model pre-training technique by first training a model on text data and then using the language representations learned by the model to solve a downstream natural language processing task with fine-tuning of hyperparameters. In this setting, the language representations are obtained using a multi-layer, bidirectional Transformer

encoder. The Datasets used are the Books Corpus (800M words) and English Wikipedia (2,500M words). In addition to the usage of transformers, the main contribution of BERT is the tasks defined to learn language representations during pre-training:

- *Masked Language Model*: this task is essential to allow the Transformer encoder to produce bidirectional language representations of tokens in any input sentence. During pre-training, the model is fed with sequences of words from which some are masked. 80% of the time the masked word is replaced by a $[MASK]$ token, 10% of the time by a random word (as $[MASK]$ token won't be observed during the downstream task), 10% of the time it is left unchanged (resulting in representations biased towards the actually observed word).
- *Next sentence prediction*: this task is used to improve the model performance on downstream tasks which require to understand the relationship between two text sentences. During pre-training, the model encoder is fed with pairs of sentences and trained to classify if two sentences are consecutive or not.

3 Approach

In this project, we implement, train and evaluate two Transformer-based encoder decoder architectures on semantic parsing tasks. The first architecture is a classical encoder decoder architecture described in [4]. The second architecture uses BERT as the encoder.

3.1 Semantic parsing inputs

In semantic parsing tasks, the inputs correspond to natural language sentences and target outputs are represented by the logical statements corresponding to their input. To feed input natural language sentences to our architectures, we take the following steps:

- Input sentence is *tokenized* using WordPiece [5]: sentence broken down into its *tokens* which correspond to its different words and some sub-word units ("wordpieces"). This approach limits the likelihood of encountering unknown tokens in semantic parsing datasets.
- Each input is thus a list of WordPiece tokens, and we use d_{model} dimensional trainable look-up embeddings of those tokens to feed them to our Transformer encoders.

After having encoded the dataset $\mathcal{D}$ as an initial grammar with rules ROOT $\rightarrow \langle x, y \rangle$, each grammar induction is defined as a mapping from an input grammar G_{in} to G_{out}. As in [2], we considered the three following grammar rules.

1. Entity: abstracting entities with their types, based on predicates in the logical form (such as `stateid`). For each $X \rightarrow \langle \alpha, \beta \rangle$ in G_{in}, two rules are added: (i) both occurrences are replaced by the type of the entity (e.g. state) and (ii) a new rule maps the type to the entity (e.g. STATEID $\rightarrow \langle$"$texas$", `texas`$\rangle$). The entity rule swaps one instance of a logical form (e.g. `florida` instance of `stateid`) with any other instance of the same form (ex: `maine`).
2. Nesting: abstracting both entities and whole phrases with their types. The first rule added is the same as previously. The second one maps the whole expression β (the logical answer) to a particular type. In this setting, the expression of `what states border A'` is mapped to the type `what is the highest montain in A` in order to produce `what is the highest mountain in states border A'`.
3. Concatenat-k: combining k sentences into one single rule ROOT.

3.2 TSP: Transformer Semantic Parser

We name TSP our simple encoder decoder semantic parser and describe its different components.

Encoder The encoder is composed of N stacked Encoder Transformer layers, as described in [4]. Each Encoder Transformer layer is built as follows:

- Each input sequence in the batch is a tensor whose length is the number of tokens in the input sequence and where each element is a d_{model} dimensional WordPiece token embedding.
- The input is then recursively fed to N transformer sub-layers, a sequence of mutli-head attention and feed-forward layers (whose intermediary dimension is d_{int}, after being mapped back into d_{model}). Between each sublayers, we apply residual connections, layer normalization and dropout.

Input sequences are fed to the first Transformer layer. The outputs of each Transformer layer is the input of the next Transformer layer.

Positional Encoding of inputs The Transformer architecture does not use recurrence or convolutions. Therefore, to make sure that the representation of each token in the input tensor depends on its position in the sequence, a positional encoding is added to the input tensor. Consider an input tensor of length L (number of tokens in the input sequence) and dimension d_{model} (dimension of the token embeddings). The positional encoding added to any such input tensor is a tensor with the same sizes and whose entries are sinusoid functions that will take different values depending on the dimension $1 \leq i \leq d_{model}$ and the position in the sequence $1 \leq pos \leq L$:

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

We add positional encodings to input sentence in the Encoder and target sequence in the Decoder.

Attention mechanisms Attention models link queries Q to key value pairs K, V. The output of this model is a weighted sum of the values, where the weights depend on the *compatibility* of the queries with the keys. The model attends to the values V depending on how relevant their keys K are to the queries Q. Multi-head, self-attention mechanisms with scaled dot-product attention are described in more details in [4].

Decoder The decoder is composed of N stacked Tranformer layers, each adding a Masked Multi-Head Attention sublayer to the Transformer layers we described for the Encoder. Each Decoder Transformer layer is built as follows:

- The positionnally-encoded target sequence is fed to a Masked Multi-Head Attention sublayer. Here, we feed the target output of the Decoder, and the self-attention mechanism should respect the auto-regressive nature of the decoding procedure.
- This first embedding of the target becomes the query Q of a multi-head attention sublayer with keys K and values V equal to the output of the encoder.
- As before, a Feed-Forward layers completes the block. Residual connections, layer normalization and dropout are still applied between sublayers.

Model output The output the of N Transformer Decoder layers is fed to a linear layer followed by a *Softmax* to obtain a probability distribution over the tokens in the WordPiece vocabulary.

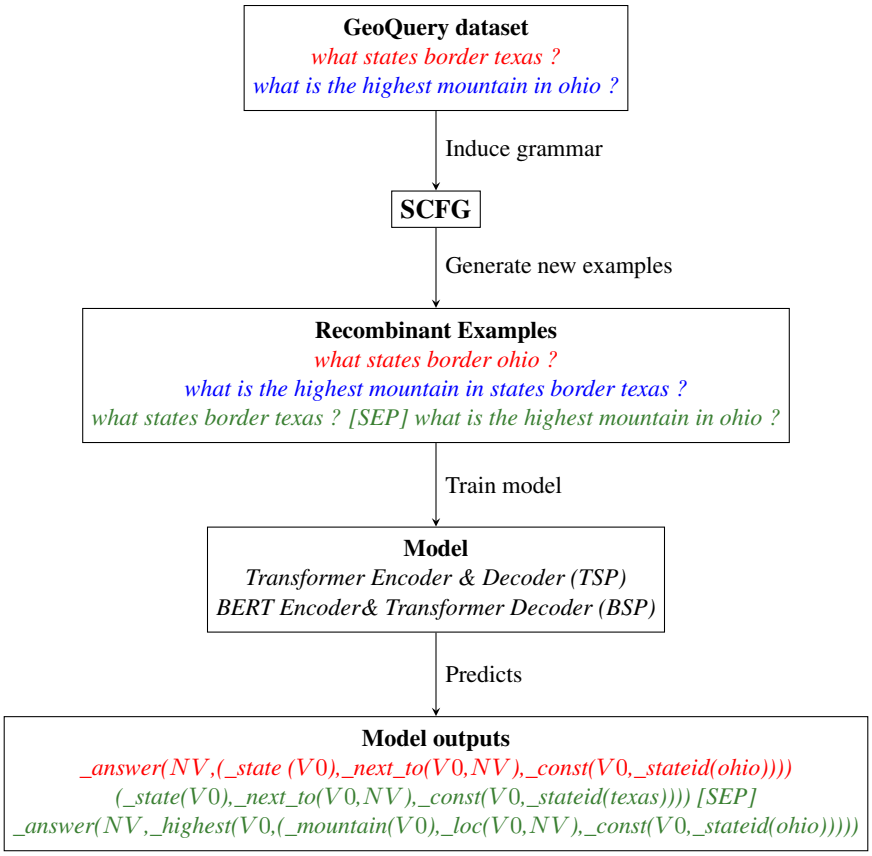
3.3 BSP: BERT Semantic Parser

We name BSP our encoder decoder semantic parser where we replace the simple Transformer Encoder of TSP by BERT, a Transformer sentence encoder, trained to learn a bidirectional language model.

In our project, we analyzed whether BERT Encoder improve TSP's performance. The two main assets of the BERT Encoder seem to be the large amount of data and bidirectional language modelling task from which it learns embeddings of natural language sentences.

3.4 Overall system

Given the GeoQuery dataset, we induce a high-precision synchronous context-free grammar. We then sample from this grammar to generate a new training and development set, which we use to train our TSP or BSP model.



3.5 Implementation with Pytorch

We based our implementation on the following sources:

- HuggingFace's pre-trained BERT GitHub repository [1]: we use their BERT tokenizer to split our inputs into tokens, and use their BERT pre-trained model as the Encoder of our BSP model.
- Harvard NLP's blogpost about transformers [2]: we directly use the code they expose for positional encodings and the masking of the Transformer Decoder inputs. Apart from those two methods, we coded our Tranformer Encoder and Decoder ourselves.
- The structure of the neural machine translation model in homework 5 inspired our implementation of the global Encoder-Decoder models (TSP and BSP), although the code inside each function is entirely different.
- Our code is made public on a GitHub repository[3].

4 Experimental setting

In this section, we describe our experimental approach and the results obtained so far.

4.1 Data

We use the dataset Geoquery [6], which is the standard for semantic parsing tasks. It basically contains at each line a single question (input) (`what is the highest point in florida ?`) and its logical utterance (output) `(_answer(A,_highest(A,(_place(A), _loc(A,B),`

[1] https://github.com/huggingface/pytorch-pretrained-BERTdoc
[2] http://nlp.seas.harvard.edu/2018/04/03/attention.html
[3] https://github.com/simhag/Compositional-Pre-Training-for-Semantic-Parsing-with-BERT

`_const(B,_stateid(florida)))))`. We split this dataset in three different files: one used as training set containing 600 data points, one for development set containing 100 data points and one for testing containing 280 data points. We then apply the three recombination techniques presented in the approach section to the training and development sets to augment and resample the data.

4.2 Training

To perform the training of our model, for each training example (x, y), we maximize the likelihood below given the problem parameters and update the model using stochastic gradient descent.

$$\mathcal{L}_\theta(x, y) = \frac{1}{T} \sum_{t=1}^{T} p_\theta \left(y_t | x, y_1, ..., y_{t-1} \right)$$

4.3 Evaluation metrics

When evaluating the performances of our models, we compare the queries they output to the target query given an input natural language question. To compare output and target queries, we focus on the following two metrics, strict accuracy and knowledge-based evaluation. The latter is employed in [2] and in the majority of the literature on GeoQuery. We also introduced two simpler and softer metrics useful for model selection and development: the Jaccard similarity and the strict Jaccard similarity.

Strict evaluation The share of output queries strings $(\hat{y}_i)_{1 \leq i \leq n}$ that perfectly match the target strings $(y_i)_{1 \leq i \leq n}$:

$$\text{Strict} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(y_i = \hat{y}_i)$$

Knowledge-based evaluation As presented in [3], knowledge-based evaluation consists in interpreting the outputs of our model as SQL queries. The metric computes the share of model outputs that both

- Correspond to a correct query to the GeoQuery geographical database
- Yield to an identical output to the target query y_i

As in [2], we use beam search to produce 5 model outputs per test example, and retain the most likely hypothesis that corresponds to a correct query. We used part of the `scala` and `java` code implemented by Robin Jia in order to parse and execute the queries. Adapting this knowledge-based evaluation code to our models proved challenging. One reason for this could be that our models are initially trained to output WordPiece tokens we eventually detokenize. We found out that some (model output, target query) tuples would not be parsed correctly by the knowledge-based evaluator, resulting in errors. When we report this metric, we leave out the (model output, target query) examples which resulted in an error for the knowledge-based evaluation score. We discuss the resulting bias for this knowledge-based evaluation figure for our final best model.

Jaccard similarity evaluation A much less strict evaluation metric, useful for development. It consists in computing the *Jaccard similarity* between the sets of characters of a model output string and the corresponding target query string:

$$\frac{1}{n} \sum_{i} \text{Jac}(y_i, \hat{y}_i)$$

where

$$\text{Jac}(y_i, \hat{y}_i) = \frac{\#(Y_i \cap \hat{Y}_i)}{\#(Y_i \cup \hat{Y}_i)}$$
$$Y_i = set(y_i) \quad \hat{Y}_i = set(\hat{y}_i)$$

Strict Jaccard evaluation An evaluation metric more severe than the Jaccard similarity and closer to the strict evaluation metric, corresponding to the share of (model output, target query) tuples where the set of their characters perfectly match:

$$\frac{1}{n}\sum_{i} \text{Jac\_strict}(y_i, \hat{y}_i)$$

where

$$\text{Jac\_strict}(y_i, \hat{y}_i) = \mathbb{1}(\text{Jac}(y_i, \hat{y}_i) = 1)$$

4.4 Experimental details

Throughout simulations, we fix the following hyperparameters:

- The batch size is set to 32 natural language questions - logical queries tuples
- The dropout rate between sublayers of our Transformers is set to $p_{drop} = 0.1$
- Decoding method is BEAM search with 5 output hypotheses.

5 Results and analyses

5.1 Baseline: shallow TSP

First, we trained a baseline model, a TSP Transformer Encoder-Decoder model with a reduced number of parameters specified in Table 1. For the project milestone, we compared the sequence of WordPiece

Table 1: Baseline hyperparameters

Hyperparameter	Value
d_{model}	128
d_{int}	128
N	2
learning rate	10^{-3}

tokens of model outputs and target examples in the test set. Results with this approach are reported again for the sake of completeness in the Appendix. Here, we report in Table 2 the results obtained by directly comparing the output strings of the model (once detokenized), as assumed in the evaluation metrics section. As a baseline, we retain the model trained without data recombination. Due to a format mismatch in the outputs when the training data is not recombined, the knowledge-based evaluation of this model becomes too tricky to compute and is hence not reported at this stage.

Table 2: Baseline TSP results

Model	Strict	Jaccard	$\text{Jac}_{\text{strict}}$
Baseline TSP	0.471	0.950	0.579

5.2 Data recombination with fine-tuning

We then carried out a first series of tests with a larger TSP architecture and different data recombination strategies. The goal was to evaluate the benefits of data recombination and large models on GeoQuery. We employed the following training procedure:

1. Training the model on the GeoQuery train and development sets where the number of examples is doubled by the selected data recombination technique. The learning rate is fixed, and the training is limited to 300 epochs. We monitor the error on the development set on `Tensorboard` and retain the latest checkpoint before overfitting.

2. Fine-tuning of the resulting model on the original GeoQuery training sets with a learning rate divided by 10. Fine-tuning is limited to 100 epochs.

The hyperparameters for these larger models are specified in Table 3 and the results are presented in Table 4.

Table 3: Large TSP models hyperparameters

Hyperparameter	Value
d_{model}	512
d_{int}	512
N	6
Training learning rate	10^{-4}

Table 4: Large TSP models with data recombination results

Recombination	Epochs	Strict	Knowledge-based	Jaccard	$\text{Jac}_{\text{strict}}$
No recombination	300 + 100	0.136	0.169	0.891	0.225
Entity	300 + 100	**0.189**	**0.259**	**0.902**	**0.282**
Nesting	200 + 100	**0.189**	0.241	0.900	**0.282**
Concat-2	150 + 100	0.157	0.192	0.892	0.204

Discussion The main result of this series of experiments is the poor performance of large models on GeoQuery, where only a thousand training examples are available. Data recombination may have helped, but limiting the number of recombined examples to doubling the size of the original dataset may have been too restrictive. We observe that the Nesting and Concat-2 data recombination techniques introduce more noise in the training data, models trained on their recombined data were early-stopped respectively after 200 and 150 epochs of training. For this first series of experiments, we employed a fixed learning rate with Adam. We investigate next a more sophisticated learning schedule.

5.3 Large BSP and TSP model

After data recombination experiments, we studied the performance and benefits of using BERT as the Encoder. We compare TSP and BSP with large architectures again, despite the poor performances of large models on GeoQuery. BERT encodings are fixed to an embedding size of 768 consistently with [1]. On the decoder side, constraining the embedding size to 128 would have required us to linear project the BERT output drastically from 768 to 128. This noisy projection would have made difficult to single out the BERT benefits. Lastly, we employ for the first time an adaptive learning rate for our Adam optimizer, as per [4], on Transformers: learning rate increases for the first 4000 optimization steps and decreases afterwards.

Basis hyperparameters have been specified in Table 3. The learning rate is now adaptive for BSP and a new TSP model, and we also expose again TSP with a fixed learning rate as it performed better. Models are trained following the training + fine-tuning procedure described in the previous subsection, with a simple *entity* data recombination operation during training, which did not cause early overfitting in the previous experiments.

Discussion Results are presented in Table 5 and illustrate the significant improvements brought by BERT. We can notice as much as 55% relative improvement over the best TSP model for our strict accuracy metric. BERT is a pre-trained architecture: the limited amount of data in the GeoQuery dataset is less of a problem to fine-tune a proper Encoder with BSP. However, results are still below our shallow baseline, as a much larger Transformer Decoder must be trained from scratch on limited

GeoQuery data. We finally note that the increasing-decreasing learning rate schedule we selected did not benefit to TSP here.

Table 5: Large TSP and BSP comparison

Model	Epochs	Strict	Knowledge-based	Jaccard	$\text{Jac}_{\text{strict}}$
TSP fixed	300 + 100	0.189	0.259	0.902	0.282
TSP adaptive	50 + 30	0.086	0.144	0.859	0.118
BSP adaptive	75 + 75	**0.293**	**0.425**	**0.944**	**0.457**
Relative Impr.	-	*55.4%*	*64.1%*	*4.7%*	*62.1%*
Absolute Impr.	-	*0.104*	*0.166*	*0.042*	*0.175*

5.4 Shallow TSP model with data recombination

Based on the results of the above experiments, we decided to build a more appropriate shallow model. Following the encouraging results of BSP compared to TSP (in the context of large architectures), we trained several BSP models with varying embedding size for the decoder Transformer $d_{model} \in \{64, 128, 256\}$, and a fixed or adaptive learning rate for the optimizer. Unfortunately, none of those models has been able to learn properly and produce satisfying results. As explained above, we suspect a significant loss of information through the linear projection we introduce between the fixed BERT Encoder outputs of size 768 and our Decoder input of size d_{model}.

Therefore, we investigated several shallow architectures with TSP and obtained very satisfying results with the hyperparameters in Table 1 as well as an adaptive learning rate. Following remarks in subsection 5.2, this time we duplicated each training example once per data recombination technique: when 3 data recombination techniques are combined, the number of training examples is thus multiplied by 4. Best results are obtained by combining the three data recombination techniques during training. They are presented in Table 6, where we mention absolute and relative improvements compared to our baseline.

Table 6: Shallow TSP with full data recombination and adaptive learning rate

Model	Epochs	Strict	Knowledge-based	Jaccard	$\text{Jac}_{\text{strict}}$
Shallow TSP	200 + 200	**0.657**	**0.630***	**0.977**	**0.768**
Baseline TSP	200 + 0	0.471	-	0.950	0.579
Relative improv.	-	*33.8%*	-	*2.8%*	*32.6%*
Absolute improv.	-	*0.159*	-	*0.027*	*0.189*

Discussion The model we retain brings significant improvements over our baseline. We regret that the knowledge-based evaluation figures lack relevance. As many as 75% of our (model output, target query) tuples were incorrectly parsed and rejected by the knowledge-based evaluator, due a hard-to-grasp formatting issue. To have an idea of what may have been the true knowledge-based performance of our best model on these failed-to-parse examples, we examined the values of the other evaluation metrics on the (model output, target query) tuples *rejected by the evaluator*. We obtain 0.576 for the strict evaluation, 0.800 for the strict Jaccard evaluation, and 0.981 for the Jaccard evaluation. Thus, we suspect that the formatting issue to be responsible for this drop in strict evaluation on the rejected samples, but the Jaccard figures let us think that rejected samples may be overall at least as good as all the accepted ones.

Overall, results are very satisfying as we obtain about 66% of strict accuracy on the GeoQuery test set. In the conclusion, we discuss several ways to build on them to develop even better Encoder Decoder Transformer models for semantic parsing.

6 Conclusion

With our project, we illustrated how a Transformer Encoder Decoder architecture could be applied to a semantic parsing task such as the GeoQuery dataset. We obtained the best results with a shallow architecture, due to the limited number of training examples in the data even when implementing recombination on the dataset. When it comes to training a larger model on GeoQuery, we demonstrated the significant improvements brought by a BERT Encoder. Unfortunately, we realized that a simple linear projection was not sufficient to properly link BERT to a shallow Transformer Decoder. Immediate next steps to improve the results of our project would include:

- Experiments with a shallow but "wide" Transformer Decoder to link with the BERT encoder in BSP. For instance, restricting the number of Decoder layers to one or two but keeping their dimension close to the fixed 768 size of BERT outputs.
- Push data recombination further by generating even more recombined examples during training. Such an approach would be limited by the number of data recombination techniques, as the number of relevant (not too noisy) examples one technique can generate is limited.
- A more fundamental step forward would be to explore how to pre-train a Decoder before direct training on the semantic parsing data.

Working on this project, we have been able to get hands-on experience with state-of-the-art natural language processing architectures such as Transformers and BERT. We realized how sensible their implementation and training can be, and learned a lot about hyperparameter tuning and model building for Deep Learning architectures. The semantic parsing task we selected has been challenging and extremely interesting to tackle, especially thanks to all the questions raised by data recombination techniques and the limited amount of data.

Acknowledgments

Throughout this project, we have been advised by Robin Jia from Stanford University's Computer Science department, whom we would like to thank for his help and advice throughout this project.

We are also very grateful to the CS224N teaching team. This course has provided us with an amazing exposure to the rich field of Natural Language Processing, from very detailed introductory lectures at the beginning of the quarter to amazing guest lectures, all of that with a very responsive and supportive teaching staff.

References

[1] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[2] Robin Jia and Percy Liang. "Data Recombination for Neural Semantic Parsing". In: *CoRR* abs/1606.03622 (2016). arXiv: 1606.03622. URL: http://arxiv.org/abs/1606.03622.

[3] Percy Liang, Michael I. Jordan, and Dan Klein. "Learning Dependency-Based Compositional Semantics". In: (2011), pp. 590–599. URL: http://www.aclweb.org/anthology/P11-1060.

[4] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[5] Yonghui Wu and al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144. URL: http://arxiv.org/abs/1609.08144.

[6] Luke S. Zettlemoyer and Michael Collins. "Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars". In: (2005), pp. 658–666. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1%5C&smnu=2%5C&article%5C_id=1209%5C&proceeding%5C_id=21.

Appendix

Table 7: Results obtained on the test set

Augmentation method	Decoding method	Jaccard	$\text{Jac}_{\text{strict}}$	Strict
None	Greedy	0.898	0.529	0.471
	BEAM	0.897	0.532	0.475
Entity	Greedy	0.937	0.582	0.675
	BEAM	0.938	0.579	**0.682**
Nesting	Greedy	0.945	0.725	0.654
	BEAM	0.947	**0.732**	0.664
Concat-2	Greedy	0.952	**0.732**	0.657
	BEAM	**0.954**	**0.732**	0.661

The results are higher than what we expected for this simple baseline model, trained on a CPU with only 2 Transformer layers and a vanilla Transformer instead of a BERT Encoder. As expected, BEAM search improves results compared to Greedy decoding. Data recombination leads also seems to lead to higher accuracy, in line with the results from [2].