CS388 Project 2: Semantic Parsing with Encoder-Decoder Models

Kelsey Ball
The University of Texas at Austin
kelseytaylorball@gmail.com

Abstract

In this project, we implement an encoder-decoder model for semantic parsing, evaluating it on the Geoquery dataset. We extend our baseline model by implementing dot-product attention over the input sequence, producing significant improvements in both model performance and training time. Finally, we implement batched processing of inputs during training and report its effect on model performance and training time.

1 Semantic Parsing

Semantic parsing is the task of translating an expression between different semantic forms. In this project, we translate geographical queries written in natural English into Prolog formulas which can be executed against a knowledge base.

1.1 Data

We use the Geoquery dataset (Zelle and Mooney, 1996) to evaluate our model. Here is an example from the dataset of an English query and its Prolog translation:

```
what is the population of
atlanta ga ?

_answer ( A , ( _population (
B , A ) , _const ( B , _cityid
( atlanta , _ ) ) ) )
```

We evaluate our model using three metrics:

1. *Token-level accuracy* computes the percentage of position-by-position token matches across all examples in the evalution set.

2. *Exact-match accuracy* computes the percentage of examples in the evaluation set where the predicted formula exactly matches the true formula.

3. *Denotation accuracy* computes the percentage of examples for which the predicted formula and the true formula, when used to query the given knowledge base, return the same result.

2 Encoder-Decoder Models

We implement an encoder-decoder, or ”sequence-to-sequence”, model, following (Jia and Liang, 2016). Our encoder is an LSTM over the input sequence, such that a sentence encoding is given by the last hidden state (and last cell state) of the LSTM.

The decoder consists of an LSTM initialized with the input sequence encoding and fed the embedding of a special start token. At each step of decoding, we pass the LSTM output vector to a linear layer with outer dimension equal to the size of the output vocabulary. Finally, applying a softmax layer to this result provides a probability distribution over possible output tokens.

During inference, we simply take the argmax of this distribution, emit the corresponding token, and feed in the embedding of this token as input to the next decoding step. However, during training, we implement ”teacher-forcing”: at each decoding step, we feed the correct token into the LSTM as input, regardless of which word the decoder previously emitted.

2.1 Attention

We augment our previous model by implementing attention over the input LSTM states. Specifically, we implement dot-product attention from (Luong et al. 2015), where the attention weight for each input LSTM state is given by its dot product with the decoder output. Then, we take the corresponding weighted sum of the input LSTM states, concatenate it with the decoder output, and feed the result into the final linear and softmax layers.

2.2 Hyperparameters

Both models are trained for 10 epochs using an Adam optimizer with step size $1e-3$. The encoder and decoder have hidden sizes of 100 and use separate vocabularies with randomly initialized, 300-dimensional embeddings.

3 Results

Our attention mechanism dramatically improves the performance of the model. Below, we compare the performance of our encoder-decoder model with and without attention, averaged across 5 runs:

Model	Exact	Token	Denotation
Baseline	10.0 %	72.3%	12.5%
w/ attn	29.3%	73.3%	32.5%

Table 1: Basic encoder-decoder model vs. encoder-decoder model with added attention over input LSTM states.

We note that dot-product attention provides an 20% absolute improvement in average denotation accuracy.

Attention also increases the speed of training. As seen in Table 2, the encoder-decoder model with attention trains $\sim 20\%$ faster than the model without attention.

Model	Avg per epoch	Total train
Baseline	14s	186.86s
w/ attn	16.8s	222.84s

Table 2: Comparison of average training time per epoch over 10 epochs vs. total training time between the baseline encoder-decoder model and the model with added attention.

4 Batching

Our project extension implements batched processing of inputs training. We verify that the total loss decreases per epoch; however, we are unable to reach token-level accuracies above $\sim 25\%$ for larger batch sizes, so it's possible that the implementation has a bug, or that we have not found suitable hyperparameters. Below, we report the token-level accuracy and train times for some different batch sizes.

Batch size	Token-level acc.	Train time
1	72.8	212.37s
10	40.2	73.37s
60	26.4	59.58s

Table 3: Accuracy and training time by batch size.

References

John M. Zelle and Raymond J. Mooney. 1996. *Learning to Parse Database Queries Using Inductive Logic Programming.* In AAAI.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. *Effective Approaches to Attention-based Neural Machine Translation.* In EMNLP.

Robin Jia and Percy Liang. 2016. *Data Recombination for Neural Semantic Parsing.* In ACL.