Task-Oriented Dialogue: Structured Seq2Seq Generation

Michael Isakov **Chang Yu** **Simeon Radev** **Tom Biasi**

Abstract

This report focuses on the problem of task-oriented dialogue from a sequence-to-sequence (Seq2Seq) perspective. We implement from scratch [1] several variants of Seq2Seq models suitable for structured output generation, namely: (1) Seq2Seq with Bahdanau attention, (2) CopyNet (3) CopyNet with retreive-and-edit augmentation, along with their respective combinations. We find that the addition of either the retrieval or attention mechanism greatly improve baseline CopyNet, and achieves near state-of-the-art performance on the SMCalFlow dataset. Interestingly, combining the retrieval, copy, and attention mechanisms does not yield a further improvement, as evidenced by both quantitative evaluation and manual error analysis. Finally, we performed fine-grained structural analysis to show that CopyNet learns modular and potentially interpretable representations of certain syntactic concepts in its hidden states.

1 Introduction

Task-oriented dialog focuses on human-computer dialogues aimed at completing a specific task. In Andreas et al. (2020), a new task-oriented dialogue dataset called SMCalFlow was presented. This contains over forty-thousand user and agent dialogue samples of varying degrees of diversity and complexity. These dialogues are examples of natural, human-like conversations between human and machine, featuring different task-related inquiries/commands roughly related to four major domains: weather, places, people, and calendar (for events).

A core novelty of the SMCalFlow dataset was the introduction of "dataflow" representations of dialogue. These dataflow representations are simple LISPRESS programs that contain functions, constraints, and API calls, as shown in the following example:

User Input:

```
What time is my appointment with Jerri Skinner on Friday ?
```

Dataflow Representation:

```
__StartOfProgram
['Yield',
 ':output',
 [':start',
  ['singleton',
  [':results',
    ['FindEventWrapperWithDefaults',
     ':constraint',
     ['EventOnDate',
      ':date',
      ['NextDOW', ':dow', '#', ['DayOfWeek', '" FRIDAY "']],
      ':event',
      ['Constraint[Event]',
      ':attendees',
      ['AttendeeListHasRecipientConstraint',
        ':recipientConstraint',
        ['RecipientWithNameLike',
         ':constraint',
         ['Constraint[Recipient]'],
         ':name',
         '#',
         ['PersonName', '" Jerri Skinner "]]]]]]]]]]
```

Andreas et al. (2020) show that using this intermediary dataflow representation boosts the performance of task-oriented dialogue systems. In addition, the dataflow representation is meant to be code-like such that the agent can effectively compute the user request. This also allows for effective cross-turn dialogue state tracking, for the agent can utilize and modify previous dataflow from earlier in the conversation to effectively respond to the user at the present moment. In this paper, we focus on the dataflow generation task from past dialogue history. Finally, we note that the difference between the output and natural language is more than simply the vocabulary: there is considerable syntactic structure, most clearly evidenced by the prevalence of parentheses (which make up over 45% of the target tokens). This has implications for our modeling approach and resulting interpretations.

2 Related Work

The conceptual goal of predicting output in the SMCalFlow dataset is closely related to that of broader

[1] Code available at https://github.com/tbiasi/6.806FinalProject

task-oriented dialog: automatic systems aimed at helping a user achieve a specific goal. This is in contrast to "chatbots", whose purpose is to maintain human-like conversation (Roller et al., 2020). The classical approach to task-oriented dialog is a pipeline which breaks up the task into several parts such as (1) natural language understanding, (2) dialog-state tracking, (3) natural language generation (Zhang et al., 2020). Often, the natural language generation portion uses a database to query relevant information.

A hallmark of these models is that each component is trained separately: this makes them more interpretable but necessitates more detailed annotations to learn each facet of the problem. Sophisticated pipelined models incorporating pre-training and neural components have been developed; for example, Gao et. al cast dialog state tracking as a reading comprehension problem, allowing for free-form slot tailored to keep track of information relevant to the task at hand (Gao et al., 2019). An alternative approach to training such pipeline models is provided by reinforcement learning, where generated dialog is viewed as an action toward the ultimate goal (Peng et al., 2018).

In contrast to the modular methods described above, some recent work has focused on end-to-end approaches to task-oriented dialog. For example, Lei et al. introduce a two-stage Seq2Seq framework to both generate text and keep track of dialog state in a concise way (Lei et al., 2018). Neither of the pipeline versus end-to-end approaches dominate state-of-the-art performance for task oriented dialog, and both techniques remain popular today (Zhang et al., 2020).

The crucial difference between this problem and a standard task-oriented dialog is that the desired output is not natural language. Furthermore, auxilliary tasks such as database querying are not required for the simplified setting considered in the SMCalFlow dataset. As such, the technical task of taking natural language and converting it to (structured) LISPRESS output is intimately connected to the problem of code generation. For example, the problem of converting natural language to SQL queries has been extensively studied (Sun et al., 2018), with the (somewhat) human readable SQL syntax being quite similar to the task at hand.

A popular approach to code generation is via semantic parsing-type techniques (Kamath and Das, 2018). Motivated by the fact that code (and many other structured inputs) are naturally represented as trees as opposed to sequences (see Example 1), such models hinge on the concept of "sequence to tree" translation. For instance, Yin and Nuebig consider the problem of generating Python code from natural language (Yin and Neubig, 2017). Viewing the target output as a treebank, they learn a grammar for the target output and use a tree-shaped decoder for the prediction task. Crucially, the use of a tree-based decoder together with a fixed grammar guarantees syntactically well-formed inputs, which is a major challenge for structured output generation with non-grammar based models. Similar models have been proposed for other code generation tasks, which suggests the general applicability of these methods (Rabinovich et al., 2017).

An alternative methodology to the above is to ignore the implicit syntactic structure of the target and learn it in an unrestricted way with a Seq2Seq model (Andreas et al., 2020). The important benefits to this approach lie in its simplicity, and the ability to leverage existing model architectures such as copy and attention mechanisms (Bahdanau et al., 2014; Gu et al., 2016). Furthermore, in light of not needing a problem-specific grammar these models can be used for a potentially wider range of tasks than tree-based approaches.

3 Methods

Because of the broad generalizability of sequence-to-sequence models, we chose to approach this task-oriented dialogue dataset as a Seq2Seq task (similar to that in Andreas et al. (2020)).

3.1 Data Processing

We used the pre-tokenized source and target datasets provided by Andreas et al. (2020). Because dialogues consist of multiple turns, the source sentences here are the concatenation of the two previous turns (user and dataflow representations) and the current user input to allow the model to adequately understand the context of the current input.

Using the train set as our source and target vocabularies, we mapped source and target tokens to unique identifies for our model. To assist with decoding, we appended special start and end tokens to the target sentences. We also included a special token for out-of-vocabulary words. Initially, we attempted to train on the entire dataset, but because of computational constraints, we dropped source sen-

tences that were larger than 200 tokens and target sentences that were larger than 100 tokens. This left us with 115,505 train set source/target pairs and 12,846 test set source/target pairs, which was approximately 97% of the orginal dataset.

3.2 Modeling

3.2.1 Baseline Seq2Seq with Attention

For a baseline model, we implemented in PyTorch a Seq2Seq model with Bahdanau attention (Bahdanau et al., 2014). The encoder was a bidirectional GRU that mapped input token embeddings into hidden state representations (which are the concatenated forward and backward encodings). The decoder was a single layer GRU which mapped target token embeddings to the next target token output; its hidden state was initialized with the final encoder hidden state. Bahdanau attention from Bahdanau et al. (2014) was used to produce context vectors as additional inputs to the decoder. The context vector at step t was the weighted average of the encoder hidden states. These α_j weights for step t were computed by taking the dot products of the decoder hidden state at step $t-1$ and $W_A h_j$, a linear mapping of the encoder hidden states to the decoder's hidden state size. The context vector for step t was concatenated to the token embedding at step t. We chose to use this type of attention because it best reflected the attentive read used in CopyNet, the next model we investigated.

3.2.2 CopyNet

In light of the large overlap between the source vocabulary and target vocabulary (14,812 tokens), and because 571 sentences in our validation set contained "out-of-vocabulary" words that appeared in source and target sentence pairs, we believed that incorporating a copy mechanism into our model would boost its performance. Not only would this encourage it to copy from the source input to target output, but it would allow the model to predict out-of-vocabulary words which appeared in the input (such as proper nouns).

In Andreas et al. (2020), a Pointer-Generator network Seq2Seq from See et al. (2017) was used for this dataset because of its copy mechanism. To differentiate our work from theirs and to reflect the modeling done with the Retrieve-and-Edit framework that we experimented with (described later), we implemented CopyNet from Gu et al. (2016), another popular Seq2Seq model with a copy mechanism. For this Seq2Seq, the same encoder and decoder from our baseline Seq2Seq was used. Nevertheless, instead of Bahdanau attention, a learned copy mechanism outputs probabilities of copying each of the input source tokens at a single decoding timestep. Specifically, these copy probabilities are generated via a dot product between the current decoder hidden state and $W_c h_j$, a linear transformation of each of the encoded source hidden states. The output distribution of the model over the target vocabulary is a sum of the copy distribution and the original decoder output generation distribution. In addition, these copy probabilities are used again as the attention weights for producing the context vector for the next decoding time step; as in the original implementation from Gu et al. (2016), the decoder only attends to tokens in the source sequence that match the current token to encourage the model to copy longer sequences.

3.2.3 CopyNet + Attention

Because of the structural differences between our natural language source sentences and the dataflow target sentences, we believed that CopyNet's use of attention was unnecessarily restrictive for our problem. Because of this, we experimented with removing CopyNet's attention mechanism and reincorporating Bahdanau attention from our Seq2Seq baseline. In the following analysis, this model is referred to as CopyNet + Attention.

3.2.4 CopyNet + Retrieval/Edit

To further improve CopyNet's performance on our dataset, we incorporated a Retrieval and Edit component. As shown in Hashimoto et al. (2018), when dealing with complex structured outputs such as code, off-the-shelf Seq2Seq models can benefit from editing a pre-existing output instead of mapping only from the source sentence. The pre-existing output is chosen via a retrieval mechanism from the training set: the most similar source sentence to the current input is found, and its relevant output is used as the pre-existing output to the editor.

Inspired by this, we developed our own retrieval-and-edit mechanism for our model. For the retrieval, we encoded each of our source sentences with a SentenceBert model pretrained on various English language datasets [2]. These 768 dimensional embeddings capture semantic and syntactic meanings of sentences and can be effectively compared using cosine-similarity (Reimers and

[2] https://github.com/UKPLab/sentence-transformers

Gurevych, 2019). For any source sentence input to our model, we encode it using SentenceBert and then find the most similar source sentence in the train set using cosine similarity. The relevant target sentence is then retrieved and appended to the source input to our model. A special retrieval token is inserted between the original source and retrieved target to explicitly indicate the difference. We then fed this augmented data directly into CopyNet, hypothesizing that it would be easier for the model to correctly predict structured output given a "template".

3.3 Attention + Retrieval

Combining all the approaches above into a single omnibus model, we also experimented with a model that had a copy mechanism, attention, and retrieval-augmented data. Henceforth, we will refer to this as the "attention + retrieval" model.

3.3.1 Transformer Encoder

Finally, we experimented with replacing our bidirectional GRU encoder with a transformer encoder in CopyNet (this transformer encoder and RNN decoder structure reflects (Wang et al., 2019)). Although we thought this may allow the model to produce better encoder representations of the input, the performance of this model was extremely poor. Despite experimenting with different methods for integrating the transformer with the decoder (including different decoder initializations and attention implementations), we were unable to improve this model, so we exclude it from the rest of our analysis. We hypothesize that this may be because of the short input sequences we are training on, but further investigation is needed to definitively establish the cause of this effect.

3.4 Training

For input to our models, we used cased Glove-300d embeddings (Pennington et al., 2014). For compound words (such as the functions described earlier), vector representations were initialized as the average of the GloVe vectors for each component word. These embeddings were frozen for all of training except the last epoch, during which they were tuned.

All of our models used a hidden state size of 256 for the encoder and decoder (because the encoder is bidirectional, its hidden state had a size of 512 in total).

All models were trained with the same hyperparameters. After manual experimentation, we found a learning rate of 1e-3 with the Adam optimizer (Kingma and Ba, 2017) performed best. Training was performed for a total of 5 epochs with the negative log-likelihood loss and a batch-size of 64. To deal with exploding gradients in CopyNet (likely stemming from the length of our input sequences), we clip all gradients at a max of 1.0. During training, teacher forcing was used 90% of the time, in light of recent literature finding probabilistic teacher forcing to help with decoding long sequences (Lamb et al., 2016).

We implemented a beam search for decoding the encoder output at test time. Throughout, we present results with a beam size of 5.

3.5 Evaluation

The primary quantitative metrics we used to evaluate the models were the percent of exact match, percent of well-formed outputs, and BLEU score.

As the desired output should be executable programs, it is imperative that the output does not contain syntactical errors (usually errors with nesting parentheses), and corresponds to correct function calls. Unlike in normal natural-language problems, where slightly incorrect output can be still interpretable by the human eye, slight mistakes such as missing a function parameter could cause the desired output action to fail completely.

We define "well-formed" output as having (1) the number of left parentheses be greater than or equal to the number of right parentheses at any point, (2) an equal number of left and right parentheses at the end of decoding. This ensures that parentheses are plausibly paired, but does not, for instance, verify if function calls are performed properly. Alternative metrics for assessing syntactic correctness remains an avenue for future exploration.

Exact match is defined in the usual way: the proportion of predictions of each model that exactly match the target outputs. Finally, we looked at BLEU score (defined in the standard way) as a softer metric to compare how similar our predictions are to the target output.

4 Results

4.1 Main Summary of Metrics

In Table 1, we present quantitative evaluations for the five main models described in the previous sec-

tion.

	Exact Match (%)	BLEU	Percent Well-Formed
Seq2Seq	62.10	87.66	88.84
CopyNet	43.0	59.43	72.25
CopyNet with Attention	**69.4**	89.09	94.48
CopyNet with Retrieval	67.5	88.78	**95.79**
CopyNet Attention and Retrieval	68.6	**89.25**	95.46

Table 1: Percent of Common Errors. The denominator is the number of errors in the top model.

Interestingly, adding either the attention or retrieval mechanism gives similar (very substantial) improvement over the baseline CopyNet. Moreover, combining the two mechanisms did not achieve better performance. Interestingly enough, CopyNet was strictly worse than the other baseline: a Seq2Seq model with attention but no copy mechanism. However, with the addition of either attention or a retrieval mechanism, CopyNet outperformed the non-copy attention model.

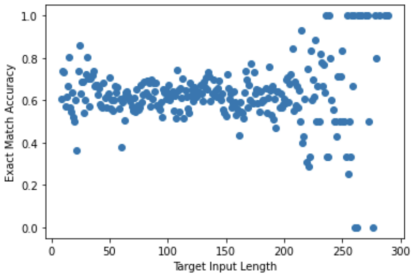

Figure 1: Input length versus Exact Match Accuracy for Baseline Seq2Seq with Attention

From Figure 2, we see that as the input length increases, our CopyNet does better in terms of exact match accuracy. However, for all the other models, the accuracy remains relatively stable on average over input length. Note for input lengths of over approximately 175, the number of inputs decreases drastically, so there is an exceptional increase in variance of the accuracy.

This difference in naive CopyNet's behavior relative to other models could be due to CopyNet failing to correctly attend to input information, as we will examine in the next section.

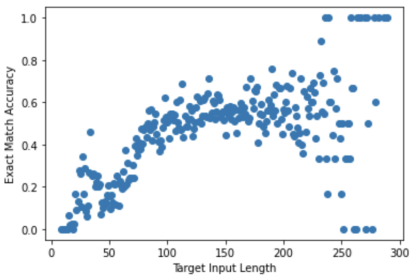

Figure 2: Input length versus Exact Match Accuracy for Naive CopyNet

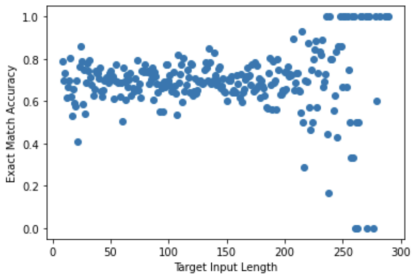

Figure 3: Input length versus Exact Match Accuracy for CopyNet with Attention

4.2 Summary of Error Relationships

To start off, we note that the copy mechanism from CopyNet helps our models by allowing us to predict out-of-vocabulary words. In particular, our CopyNet with Attention model successfully copied over out of vocabulary words on 246 different sentences in our validation set, of which it got 23% exactly correct. Similarly, our CopyNet with Retrieval copied over out of vocabulary words on 322 different sentences in our validation set, of which it got 29% exactly correct. Of course, the baseline Seq2Seq would automatically get these outputs wrong.

Because of Seq2Seq's inability to handle out-of-vocabulary inputs/outputs, and because the CopyNet variants performed best, we will focus most of our analysis on CopyNet and more specifically how attention and retrieval alter the construction of output.

We performed analysis to understand how the attention and retrieval mechanisms differed, focusing on outputs that were not exact matches and/or well-formed. Rather than just focus on selected examples and subject ourselves to small sample

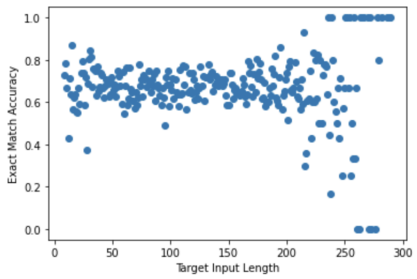

Figure 4: Input length versus Exact Match Accuracy for CopyNet with Retrieval

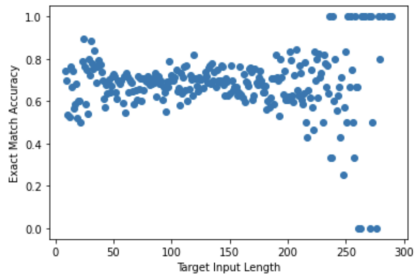

Figure 5: Input length versus Exact Match Accuracy for CopyNet with both Attention and Retrieval

bias, we wish to first get a better picture of how the approaches differ in general. A simple approach to determine whether the models are making similar errors is to examine the proportion of overlap in examples that they predict incorrectly.

Model	Attention	Retrieval	Atten+Retr
Attention	—	74.59	78.12
Retrieval	79.11	—	77.77
Atten+Retr	80.12	75.22	—

Table 2: Percent of exact match errors which overlap between models. The denominator is the number of errors in the top model.

From Table 2, we see that the inputs on which each model made exact match errors were mostly the same. However, for attention and retrieval, the percent of outputs that matched exactly among those common errors was 28.47%, which signifies that attention and retrieval focus on different aspects in forming the output. Additionally, the similar percentage of common errors between attention or retrieval with the combined model illustrates that the combined model retains aspects of both mechanisms without one mechanism completely dominating the other. Therefore, we examine the metric that examines the syntax of the output, our well-formed metric, to understand more deeply where the outputs differ.

Model	Attention	Retrieval	Atten+Retr
Attention	—	23.48	28.82
Retrieval	17.91	—	23.84
Atten+Retr	23.70	25.69	—

Table 3: Percent of well-formed errors which overlap between models. The denominator is the number of errors in the top model.

Interestingly enough, from Table 3, the lack of significantly common errors in well-formed results among the three methods shows us that indeed attention and retrieval are learning different syntax, which is what we suspected from the low output similarity between attention and retrieval.

5 Comparative Error Analysis

5.1 Parsing the Weather in Live Oak

For illustrative purposes, we focus now on a single example, which has a source input requesting for the weather in Live Oak, FL. The output from base CopyNet is:

```
['__START_PREDICT', '(', 'Yield', ':output', '(',
'CreateCommitEventWrapper', ':event', '(',
'CreatePreflightEventWrapper', ':constraint', '(',
'Constraint[Event]', ':start', '(', '?=', '(',
'DateAtTimeWithDefaults', ':date', '(', 'NextDOW', ':dow',
'#','(', 'DayOfWeek', '"', 'FRIDAY', '"',
')', ')', 'FL', '#', '(', 'FL', '(',
'Constraint[Event]', ')', ')', ')', ')', '(', 'refer',
'(', ')', ')', ')', ')', ')', ')',
')', ')', ')','ResponseStatusType', 'x0', ')', ')',
')', ')', ')', ')', ')', ')',')', ')',
')', ')', ')', ')', ')', ')', ')', ')',
')', ')', ')',')', ')', ')', ')', ')',
')', ')', ')', ')', ')', ')', ')',
')',')', ')', ')', ')', ')', ')', ')',
')', ')', ')', ')', ')', ')']
```

Note that there are multiple errors occuring here: (1) the extreme number of right parentheses at the end represents a failure to understand syntax well, (2) the lack of string inputs concerning requests for weather and location information about Live Oak suggests problems with meaning representation. Overall, it is clear that the baseline CopyNet did not fully extract relevant request information nor completely understood how the output syntax operates for this example.

By contrast, the attention mechanism output is

```
['__START_PREDICT', '(', 'Yield', ':output', '(',
'WeatherQueryApi', ':place', '(', 'AtPlace',
':place', '(', 'FindPlace', ':keyphrase', '#',
'(', 'LocationKeyphrase', '"', 'Live', 'Oak',
'FL', '"', ')', ')', ')', ':time', '(', '?=',
'(', 'Now', ')', ')', ')', ')', '__EndOfProgram']
```

Unlike CopyNet, introducing attention allowed the model to retain input information about needing to request for weather and the location being Live

Oak, FL. However, although not immediately clear from the output, the attention mechanism is still missing some function parameter information; for instance, the ":place" parameter is nowhere to be found.

Comparatively, the output from the retrieval model is:

```
['__START_PREDICT', '(', 'Yield', ':output', '(',
'WeatherQueryApi', ':place', '(', 'AtPlace',
':place', '(', 'FindPlace', ':keyphrase', '#',
'(', 'LocationKeyphrase', '"', 'live', ':place',
'FL', '"', ')', ')', ')', ':time', '(', '?=',
'(', 'Now', ')', ')', ')', ')', '__EndOfProgram']
```

The retrieval mechanism was much better in creating syntactically valid functions. However, the retrieval mechanism dropped the input information about Live Oak, despite retaining more semantic information than the baseline CopyNet (such as the request for weather).

Finally, combining both mechanisms, we obtain the following output:

```
['__START_PREDICT', '(', 'Yield', ':output', '(',
'WeatherQueryApi', ':place', '(', 'AtPlace',
':place', '(', 'FindPlace', ':keyphrase', '#',
'(', 'LocationKeyphrase', '"', 'Live', 'City,',
'FL', '"', ')', ')', ')', ':time', '(', '?=',
'(', 'Now', ')', ')', ')', ')', '__EndOfProgram']
```

Unfortunately, combining the mechanisms did not grant us the best of both worlds, as the output is both missing the "Oak" location aspect of Live Oak, FL and missing a ":place" parameter. However, the mechanism still resulted in an output that retained more input information than the naive CopyNet and had less syntactical errors.

For the sake of completeness, we examine the baseline Seq2Seq model output:

```
['__START_PREDICT', '(', 'Yield', ':output', '(',
'WeatherQueryApi', ':place', '(', 'AtPlace',
':place', '(', 'FindPlace', ':keyphrase', '#',
'(', 'LocationKeyphrase', '"', 'Concord,', '#',
'fl', '"', ')', ')', ')', ':time', '(', '?=',
'(', 'Now', ')', ')', ')', ')', '__EndOfProgram']
```

Notice that although Seq2Seq retains important input information such as a request for weather at a particular location and has fewer syntactical errors than naive CopyNet, it has difficulty with proper nouns in the output, inserting "Concord" instead of "Live Oaks" as the location name.

Overall, this comparison solidifies some basic intuitions about the differences in these models and motivates a more fine-grained analysis of the inner workings of the models.

5.2 Analyzing Copy Weights

We continue with the example above to further understand how the augmented data in the retrieval model improves performance.

Fig. 6 shows the copy weights relative to the retrieved sentence at each step of the decoder. There are several interesting and potentially surprising insights to be gleaned here. The retrieved sentence in this case is not particularly close to the target, so one might at first glance expect little gain from the retrieved output in this case. However, one sees from the heatmap a substantial amount of copying at the parentheses of the output. This behavior was observed for other inputs as well, which suggests that a substantial portion of the retrieval mechanism is dedicated to learning syntactic features like parentheses. This hypothesis is supported by the large improvement in percentage of well-formed outputs after the addition of the retrieval mechanism. Interestingly, the copy mechanism mostly ignores semantic information in copying, such as the token "Tomorrow" which is present in both the target and the retrieved sequence.

Curiously, we noticed that, across various examples, the copy weights attend to only a single parenthesis instance in the retrieved target (usually the last parenthesis in a group). This is potentially at odds with the idea that CopyNet sequentially copies inputs; however, we posit that this is an artifact of the bidirectional encoder capturing local information about number of parentheses well (that is, it suffices to attend to only a single parenthesis to successfully copy the entire group). Indeed, we observed the sequential copying behavior in cases where the retrieved code was extremely similar to the target output; in those cases, the copy weights were heavily concentrated on the diagonal as hypothesized in the original CopyNet paper.

6 Interpretability

Given the structured nature of the target sequence, and the fact that the models described above do well at creating well-formed outputs, it is interesting to explore how this syntactic structure (as well as other information) is captured by the model parameters. For simplicity, we focused on the vanilla CopyNet implementation, as the other variants introduce complexities that require a more detailed analysis in the future.

6.1 Clustering Hidden States

First, we looked at the hidden state representations, which form the backbone of the sequential decoding procedure described above. Our setup was as follows: (1) randomly sample 500 examples from

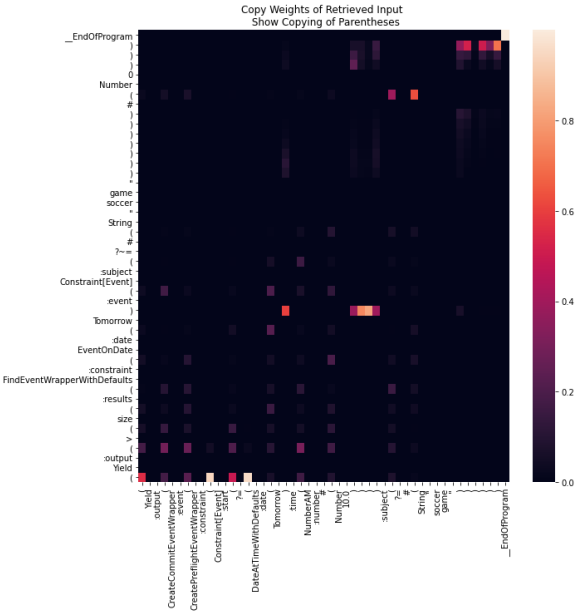

Figure 6: Heatmap of copy weights with respect to retrieved target. The x-axis denotes the true target sequence, while the y-axis represents elements of the retrieved sequence. The value in each cell if the probabiliy of copying over a given element of the retrieved sequence at various points in the decoding.

the validation set, (2) run the sentences through the CopyNet model with teacher forcing, recording the hidden state together with the intended output. Considering each token in isolation, this gives a 256-dimensional datum for each of around 20 thousand randomly sampled tokens. While a dataset generated in this way does not truly give independent samples due to the recurrent nature of the decoding, this method has been applied to interpreting hidden states of RNNs in the past (Belinkov and Glass, 2019).

Applying principal component analysis (PCA) to reduce the dimension of the hidden states to a more manageable size (50 was chosen based on an elbow in the plot of explained variance), we performed K-means clustering with $K = 10$ on the vector representations. In Fig. 7, we report the unique representatives of 10 randomly sampled tokens from each cluster. In this figure, we see several interesting trends: first, parentheses (left or right) are the only elements of four out of the ten clusters. This shows that the hidden states encode significant syntactic information (as opposed to, say, the embedding or the copy weights). While perhaps unsurprising given the large number of parentheses in the structured outputs, but is nonetheless reassuring.

Other observations are more interesting: for example, group 1 appears to contain only *named* arguments to functions. This is another syntactic feature of the output, and the model is able to recognize when a certain configuration necessitates a named as opposed to an unnamed argument. The relationships encoded in the hidden states are not purely syntactic, however. For instance, group 8 features many functions pertaining to events. Semantically similar function calls being related in the hidden state representations suggests that crucial context is being carried over from the encoder.

To further probe the hidden state representations learned by the vanilla CopyNet, we removed the tokens for end of string, parentheses and the pound symbol from consideration, as these syntactic features mostly dominated the K-Means clustering described above. Fig. 8 shows the first two principal components (PCs) of the hidden state representations. Applying a density-based scan, DBSCAN (Ester et al., 1996), we can see three main clusters in the plot. The one labelled "Yield" contains only "Yield" tokens, which accounts for more than 99% of yield tokens found in the dataset. The one labelled ":output" has the token for :output as more than 80% of its constituents. Overall, this suggests that there is clear structure in the CopyNet hidden states representations, and that this even low-dimensional visualizations can help interpret the inner workings of the model.

We remark in passing that there appears to be additional structure the PCA plot, so one may wonder whether these clusters (as well as the previous results) are merely noise due to either the randomness in the inputs or the visualization technique. To assess the first question, alternative visualization methods such as PCA in three dimensions and t-distributed stochastic neighbhor embeddings (Van der Maaten and Hinton, 2008) were also explored. To mitigate the second concern, we reran several different initializations for the target sentences. All of these analyses achieved qualitatively similar results, so we can be relatively confident that these observations are not merely due to chance.

6.2 Role of individual neurons

In light of the interpretable hidden state structure observed in the previous section, it is reasonable to ask whether we can interpret individual hidden states in the vanilla CopyNet. It has been shown

```
Random tokens from group 0 are:  ", Constraint[Date], MONDAY, RecipientWithNameLike
Random tokens from group 1 are:  :constraint, :date, :holiday, :id, :period
Random tokens from group 2 are:  )
Random tokens from group 3 are:  (, __START_PREDICT
Random tokens from group 4 are:  :output, Yield
Random tokens from group 5 are:  )
Random tokens from group 6 are:  __EndOfProgram
Random tokens from group 7 are:  (
Random tokens from group 8 are:  :start, ConfirmAndReturnAction, Constraint[Event], EventOnDate, FindNumNextEvent
Random tokens from group 9 are:  (
```

Figure 7: Unique random tokens drawn from each cluster of target words. Clusters were determined by performing K-means clustering on the top 50 principal components of CopyNet's hidden state representations.

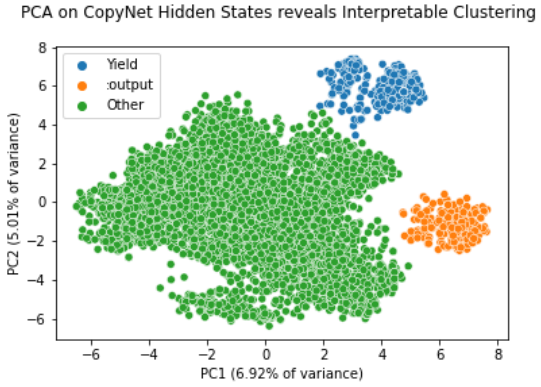

Figure 8: PCA reveal visual clustering of hidden states. The "Yield" and ":output" clusters are manually labelled due their major constituents.

that such interpretations are occasionally possible, and given additional insight into the structural behavior of the model (Dalvi et al., 2019).

First, we explored whether the main clustering behavior of the hidden representations could be explained by only a handful of hidden state neurons. To test this, we used the randomly sampled hidden states from the previous experiment as input into an L^1-regularized logistic regression on whether a given token was a left parenthesis (choosing other representatives of common clusters gave qualitatively similar results). Taking the coefficients with highest absolute value and manually examining actions at various points in the sentence, there did not appear to be a single neuron which was highly correlated with parentheses alone. This suggests that some syntactic information is distributed across the hidden states.

With this negative result, a natural followup is to ask whether any neurons in CopyNet have specialized functions. Indeed, Fig. 9 shows a single hidden state which is highly negative only on the ":intension-ActionIntensionConstraint" (henceforth, intension) construct. This suggests that the hidden state representations are more modular than it seems at first glance: the activation of a single neuron essentially determines the existence of an intension construct at a given point.

The differential representations of syntactic constructs such as parentheses and specific code chunks is intriguing. We hypothesize several possibilities: (1) parentheses are far more prevalent the training and testing set than the intension construct, which occurs only in 0.007% of the training target sentences. Hence, the model may be forced to have a more robust representation of the former, as systematic mistakes on parentheses would be more costly in terms of average training loss. In addition, it is possible that: (2) "general" syntactic structures like parentheses, which require substantial information about many points in the past to get correct, may be harder to represent in just a single neuron that a more specific feature like the intension construct.

Finally, we note that the above analysis is only associative. To more fully assess the effect of these neurons on the output, one would need to perform an ablation study. We leave such analysis to future work.

Although interpretability of non-natural language outputs is necessarily challenging, structural analysis like the above has important ramifications for model distillation and architecture improvements (Belinkov and Glass, 2019). Additional ex-

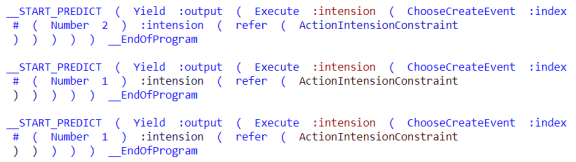

Figure 9: Activation of a single neuron in CopyNet hidden states on three randomly sampled sentences from the validation set having the intension construct. Blue and red represent positive and negative activation, respectively, with intensity being indicative of intensity.

ploration to this end, especially in the context of other code generation tasks, is thus another important aspect for future work.

7 Conclusion and Future Work

In this report, we consider several modifications to the sequence to sequence framework for structured output generation, specifically in the context of task-oriented dialog. We found that a naive Seq2Seq model with attention suffers greatly from out-of-vocabulary errors and is thus not suitable for the task-oriented dialog problem without modification. An important modification considered here is CopyNet; however, despite ameliorating the out of vocabulary challenge, we found that at first attempt this performs worse than the naive Seq2Seq. To remedy this, we considered the addition of two mechanisms: global attention and retrieval/edit.

Although both attention and retrieval have increased CopyNet's performance on exact match and well-formed metrics, the two mechanisms greatly differed in their focus for improving output generation. Specifically, the attention mechanism retained more information from the input, while the retrieval mechanism made fewer syntactical errors in the output.

Surprisingly, combining the two mechanisms does not yield a model that is clearly superior to either mechanism alone. Through manual analysis, we found that incorrect programs are neither fully syntactically correct nor completely retain the important input information, a middle ground that yields no benefit. Future work on a more fruitful combination of these two approaches is needed, and will likely result in substantial performance improvement.

Our analysis of the hidden state representations of CopyNet also shows substantial structure in the way syntactic and semantic information is encoded in the network. More work on interpretability can give additional insights into the differences between the architectures explored, and suggests further modifications to improve performance. Moreover, such research would allow for greater accountability if such task-oriented dialog systems are put into practice.

References

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Andrew Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *CoRR*, abs/2009.11423.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Yonatan Belinkov and James Glass. 2019. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, Anthony Bau, and James Glass. 2019. What is one grain of sand in the desert? analyzing individual neurons in deep nlp models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6309–6317.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

Shuyang Gao, Abhishek Sethi, Sanchit Agarwal, Tagyoung Chung, and Dilek Hakkani-Tur. 2019. Dialog state tracking: A neural reading comprehension approach. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 264–273, Stockholm, Sweden. Association for Computational Linguistics.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.

Tatsunori B. Hashimoto, Kelvin Guu, Yonatan Oren, and Percy Liang. 2018. A retrieve-and-edit framework for predicting structured outputs.

Aishwarya Kamath and Rajarshi Das. 2018. A survey on semantic parsing. *arXiv preprint arXiv:1812.00978*.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. 2016. Professor forcing: A new algorithm for training recurrent networks. *arXiv preprint arXiv:1610.09038*.

Wenqiang Lei, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He, and Dawei Yin. 2018. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1437–1447.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, Kam-Fai Wong, and Shang-Yu Su. 2018. Deep dyna-q: Integrating planning for task-completion dialogue policy learning. *arXiv preprint arXiv:1801.06176*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. *arXiv preprint arXiv:1704.07535*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

Stephen Roller, Y-Lan Boureau, Jason Weston, Antoine Bordes, Emily Dinan, Angela Fan, David Gunning, Da Ju, Margaret Li, Spencer Poff, et al. 2020. Open-domain conversational agents: Current progress, open problems, and future directions. *arXiv preprint arXiv:2006.12442*.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax- and table-aware sql generation. *arXiv preprint arXiv:1804.08338*.

Chengyi Wang, Shuangzhi Wu, and Shujie Liu. 2019. Accelerating transformer decoding via a hybrid of self-attention and recurrent neural network. *CoRR*, abs/1909.02279.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*.

Zheng Zhang, Ryuichi Takanobu, Qi Zhu, Minlie Huang, and Xiaoyan Zhu. 2020. Recent advances and challenges in task-oriented dialog systems. *Science China Technological Sciences*, pages 1–17.