Notes on the metodology of SCAN seq2seq paper methodology

Domingo Edwards, Ignacio Villanueva, Lucas Gutiérrez

June 29, 2023

1 Dataset

1.1 Input/Output

Input: $command \in C^{m_1}$ Where $m_1 \in \{1, \ldots, m_{1max}\}$ is the length of the *command* and C is the vocabulary of the commands, including

$$\begin{aligned} C = \{&and, after, twice, thrice, opposite, \\ &around, left, right, turn, walk, look, run, jump\} \end{aligned} \tag{1}$$

So the input is a combination of words in the vocabulary of any length. m_{1max} is given by the most long combination that can be produced by the gramar of SCAN generation rules.

Output: $action \in A^{m_2}$

$$A = \{WALK, LOOK, RUN, JUMP, LTURN, RTURN\} \tag{2}$$

Examples: $s \in S \subset C^{m_1} \times A^{m_2}$
Where $|S| \geq 20.000$ and is stored in `tasks.txt`

- s_1 : turn left twice $\rightarrow$ LTURN LTURN
- s_2 : jump opposite left and walk thrice $\rightarrow$ LTURN LTURN JUMP WALK WALK WALK
- ...

2 Tasks

2.1 Task 1: Generalize Holdout Train/Test i.i.d

The first task is to generalize some *i.i.d* data in train and test where the test has not all the combinations, formally

$$\begin{aligned} train \cup test = S \\ |train| = p \cdot |S| \end{aligned} \tag{3}$$

Where $p \in [0, 1]$ represent the quantity of data that is added into the train set.

$$train \cap test = \emptyset \tag{4}$$

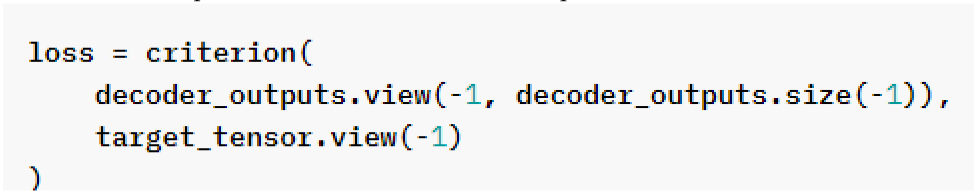
So the test and train share no instances s_i, but the train set is **representative** of S. So the idea is that a model can get the information of the training set and extrapolate that to the test set.

In short terms

1. S using holdout (train/test split) *i.i.d* representative

2. Train model on *train* set
3. ¿Evaluate (COMO EVALUA EL OUTPUT)? model on *test* set

   Based on the implementation where *loss* is computed as



```
loss = criterion(
    decoder_outputs.view(-1, decoder_outputs.size(-1)),
    target_tensor.view(-1)
)
```

Where `decoder_outputs` are the sequence of actions decoded and `target_tensor` is the correct sequence of actions in the dataset.

Accuracy supposition 1 We can **SUPOSE** that the evaluation metric is if the output given by the decoder is the output in the text or not (Binary $\{0,1\}$ metric averaged for all instances)

Accuracy supposition 2 We can **SUPOSE** that the evaluation metric is based in each word predicted by the decoder, vs the word that the decoder should have predicted.

Metric 3 The *loss* generated in the optimization as a metric for all the cases.

Files In SCAN dataset for this task
`SCAN/simple_split` contains the train test data split for this task where:

$p = 0.8$

- `tasks_test_simple.txt` = train
- `tasks_test_simple.txt` = test with 0.2 left of S as test set

More p splits of S For more splits of S with different values of p there is `simple_split/size_variations` folder

$p =$ `X`

- `tasks_train_simple_p<X>` = train
- `tasks_test_simple_p<X>` = test with $(1-p)|S|$ instances

So there are files with multiple train/test splits that gives on the test all the instances of SCAN not included in the training set.

objectives of this task: We could see this task as a proof for generalization because the model generalize instances not seen before from S, but it is **NOT** a proof for **exemplar** vs **rules** based knowledge because the data belongs to the same distribution in train and test (*i.i.d*)

2.2 Task 2: Generalize long actions, Holdout Train/Test o.o.d by *length* criteria

Here the task is that from a training set with commands that only generate small actions, the model can generalize the commands that generate long actions sequence. Formally the split is done like

$$train = \{s_i \in S \mid action\_length(s_i) \leq L\} \tag{5}$$

Where $L = 22$ and *action_length* is a function that gives the length $|y_i$ of the action y_i (ouput) that is generated for $s_i = (x_i, y_i)$

$$test = S \setminus train \tag{6}$$

So still

$$S = train \cup test \tag{7}$$

objectives of this task: We could see this task as a proof for **exemplar** vs **rules** knowledge of the model, because the data that is given in the test have the **same rules** (The grammar used to generate the sequence), but has different **distribution** because the selection based on length breaks the *i.i.d* asumption creating *o.o.d* data.

Notes in the result of the paper

The results of the paper where really bad so the authors provide a ways of "fixing" the given model.

Oracle for given length : They provide an **oracle** that gives the length of the output sequence, forcing the model to generate a longer sequence even if the decoder outputs an `<EOS>` token. This improved the results only from 13.8% to 23.6%, another went up from 20.8% to 60.2%. The interpretation is that the problem is not **only** that the models only stops too early, based on the previous better results in the *i.i.d* task.

There is also a sensibility analisis in the length of the actions L that makes the split of the task dataset.

Files in SCAN dataset for this task

The train and test files are in `SCAN/length_split/**` folder, and contains one file for train and test with $L = 22$ the max number of commands in y_i in train, where each line of the file is an instance (x_i, y_i).

3 Implementation of methodology for Transformer Architecture

Objectives

Rule vs Exemplar based The idea of this project is to implement the same tasks described in the previous sections but for the recent **Transformer** architecture. Given the results given in (Chan et al., 2022) the transformer architecture store **rule based** knowledge in weights, where taht information is obtained during training. So the hipotheisis of this project is that if that is true, the transformer should perform good in *o.o.d* testing because it *should* learn the semantic function that produce the outputs, unlike the models presented in the original paper.

Replication of previous evaluation strategy Another objective is to replicate the current evaluation system given in SCAN article. But with a new transformer model.

Sensibility analysis of transformer architecture The idea is also to give a sensibility analysis for the transformer architecture parameters and compare results in *i.i.d* and *o.o.d* tasks.

Methodology

only transformer: The first idea is just implement transformer architecture for the already given SCAN datasets and measure the stast generated with different parameters.

With Control Model: The second methodology is give a seq2seq LSTM model as control group for the different experiments and see if the results of the Transformer architecture are really better than architectures with memory but no attention as LSTM. (The control models could be expanded)

3.1 Experiment Setup

Preprocessing

The dataset is the same as before but must be treated with some preprocessing for input in the transformer architecture.

Given an instance $s_i = (x_i, y_i) \in S$

We need to *preprocess* x_i with te following

- `x_i = <SOS> + x_i + <EOS>`

Where `<SOS>` and `<EOS>` are the start of sequence and end of sequence tokens.

If after that $|x_i| \leq$ `CONTEXT_LENGTH` we need to apply padding with the `<PAD>` token. Here the context length parameter is the maximum context accepted by the transformer encoder.

example: `jump twice` Assuming CONTEXT_LENGTH as 10 we need to generate

```
(<SOS>, jump, twice, <EOS>, <PAD>, <PAD>, <PAD>, <PAD>, <PAD>, <PAD>)
```

So our new vocabulary of commands is $C^* = C \cup \{$`<SOS>`$,$`<EOS>`$,$`<PAD>`$\}$

For the output we do the same as before, in the same example as before we have

```
JUMP JUMP
```

and the preprocessing should be

```
(<SOS>, JUMP, JUMP, <EOS>, <PAD>, <PAD>, <PAD>, <PAD>, <PAD>, <PAD>)
```

This give us again a new vocabulary $A^* = A \cup \{$`<SOS>`$,$`<EOS>`$,$`<PAD>`$\}$ for the output

We said that $S_{processed} \subset C^* \times A^*$ is the new preprocessed space four our instances s_i. Formally we have

$$Preprocessor : S \rightarrow S_{preprocessed} \tag{8}$$

*A inverse preprocessor can be coded as well to see the sequences of results without the added tokens.

Tokenization

The tokenization simply takes the preprocessed input $(x_i, y_i) \in S_{processed}$ and applies a function $f_{x_i} : C^* \rightarrow \mathbb{N}$ and $f_{y_i} : A^* \rightarrow \mathbb{N}$. In simple words maps from the vocabulary to a different number as token for each word in the vocabulary. Formally our tokenizer maps

$$Tokenizer : S_{processed} \rightarrow S_{tokenized} \tag{9}$$

*A detokenizer can be coded as well to see the sequences of results.

Training

During training the transformer receives an instance $s_i = (x_i, y_i) \in S_{tokenized}$, the x_i is passed to the transformer encoder input, producing an internal representation as output. Formally

$$Encoder(x_i) = encoding \tag{10}$$

After encoding the `<SOS>` is passed to the decoder as first input, the *encoding* generated by the input is passed to. Formally

$$Decoder(encoding, \texttt{<SOS>}) = output_i \tag{11}$$

Is passed first. We then generate until an `<EOS>` is generated. The algorithm is as follwos

Input: $s_i = (x_i, y_i) \in S_{token}$

```
1. final_ouput = <SOS>
2. encoding = Encoding(x_i)
3. output_i = Decoder(encoding, final_output)
4. if output_i is <EOS>, add <PAD> to fill context and return(final_output)
5. final_output += output_i, go to 3
```

Loss Function

The above algorithm gives an output that predicts the sequence of commands and can be tested against y_i vs $y_{pred} = final\_output$, computing a loss function for model optimization.

Loss metric Cross entropy of the ground trouth y_i vs y_{pred} the final output generated by the decoder.

Note*: Here the strategy of "teacher forcing" presented in the original paper is not used, because the model is not recurrent.

Optimization

Use a standard optimizer like ADAM or ADAMW, (the most used in transformer architectures).

Evaluation

The evaluation will be the *accuracy* of the model counting the number of well written sequences of commands like

$$score(y_i, y_{pred}) = \begin{cases} 1 & \text{if } y_i = y_{pred} \\ 0 & \text{if } y_i \neq y_{pred} \end{cases} \tag{12}$$

And the accuracy is computed as the average of all the correct prediction in the test set.

$$\frac{\sum_{i=1}^{n=|S_{test}|} score(y_i, y_{pred})}{|S_{test}|} \tag{13}$$

This metric is used for the sensibility analysis and also for the optional control model.

References

Chan, S. C. Y., Dasgupta, I., Kim, J., Kumaran, D., Lampinen, A. K., & Hill, F. (2022). *Transformers generalize differently from information stored in context vs in weights.*