# CS5740: Assignment 4

# https://github.com/cornell-cs5740-20sp/assignment-4-nlprime

Kushal Singh ks2377
Willy Lin wl677

8 May 2020

We are using `DyNet`.

## 1 Introduction (5pt)

The goal of this assignment is to implement a sequence to sequence model to map instructions to action in the Alchemy environment, which contains a certain number of beakers, colors, and beaker positions. The system can move colored liquid content from one beaker to another, mix content, or remove content from the set of beakers. The data provided contains:
1) a unique ID (e.g. train-A9164)
2) an initial environment configuration which represents the state of the initial Alchemy beaker setup, prior to executing the sequence of actions (e.g. 1 2 3 4p 5bb 6yyy 7rrrr represents the initial state of the seven beakers such that beakers 1 through 3 are empty, beaker 4 contains one part purple, beaker 5 contains two parts brown, beaker 6 contains three parts yellow, and beaker 7 contains four parts red
3) a list of utterances, where the key is the natural language instruction (e.g. "pour purple beaker into yellow one"), and the value is a set of actions corresponding to that instruction (e.g. "pop 3, push 5 p, stop"), as well as the final environment configuration which represents the state of the Alchemy beaker setup, once the action has been executed.

Primary experiments included testing for optimal DyNet builder and optimizer, making incremental additions to the model (e.g. adding state information, adding attention, adding interaction history) and fine-tuning hyper-parameters (e.g. number of hidden layers, number of training epochs). Cursory results of these experiments included an embedding dimension size of 50 for word-level embeddings and 20 for character-level embeddings, number of hidden layers at 300, and 30 training epochs. The model was evaluated by testing development data prediction against the ground truth, using two key metrics: single-instruction task completion and interaction task completion. Interaction-level accuracy tended to be lower than instruction-level accuracy, since the former requires correct execution of ALL the instructions in the sequence. Final instruction-level accuracy achieved on leaderboard was **0.48552**.

## 2 Model (35pt)

To achieve the objective detailed above, we used a sequence-to-sequence model that comprises of two neural networks: an encoder and a decoder. In tandem, the encoder and decoder help the model learn to perform a sequence of actions corresponding to language instructions and environment states. Specifically, the encoder encodes instructions into a context vector. The decoder, in turn, takes this context vector and based off the environment state predicts the actions to be performed.

The first optimization we made to our model was to incorporate environment state information, so that when the model is provided with the natural language expression *throw out two units of brown one* it knows which beaker has the brown color and in which positions in order to predict the right action. In order to incorporate this behavior we created an embedding matrix for all 7 beakers, where each individual entry contained color by parts information for each beaker. Subsequently, we add attention, thereby allowing the decoder to pick the encoded inputs that are relevant to make the prediction at hand. Finally, the output is concatenated to each input of the decoder.

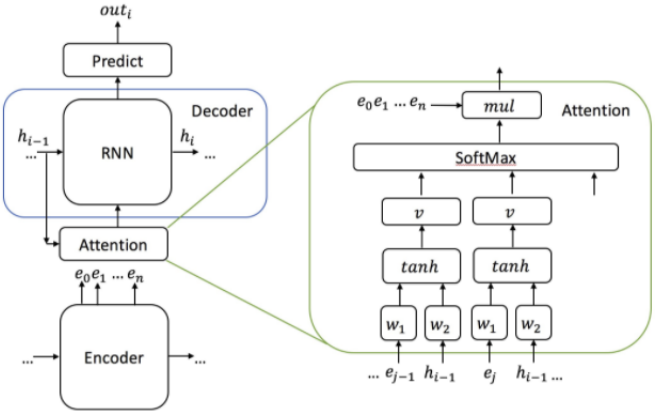
The second optimization we made to our model was adding an attention mechanism, which computes a weighted combination of multiple inputs, where the weights are determined by the current state of the decoder. The attention mechanism attends on both the output of the instruction encoder and the output of the environment encoder, since these were the two aspects that most directly determined an optimal decoded state.

The third optimization we made to our model was adding interaction history, so that when the model is presented with the series of instructions [*pour sixth beaker into last one*, *it turns brown*], it knows that the word "it" is referring to "last one" as its modifier. To achieve this, we concatenate the previous instruction to the current instruction, and pass this as input to the encoder, instead of passing each instruction individually. So, the instruction sequence

above becomes *pour sixth beaker into last one <end> it turns brown*].

We used Finite State Automata (FSA), which was provided as part of the template code, to model the transitions provided by the decoder's output, given the initial environment state and final environment state.

A visualization of the encoder-decoder network and the final model is shown below.



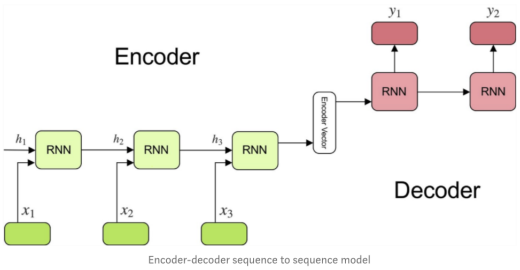

## 3 Learning (15pt)

To maintain consistency between training and testing data, we grouped series of five sentences into a single batch, and trained on this batch. By doing this, our model was robust enough to handle aforementioned discrepancy of training on single sentences versus testing on sequences, which could have lead to errors where antecedents were not correctly referred back to their modifiers.

To calculate instruction and interaction level losses, we made the MLE prediction, took the negative log, and then used the DyNet esum function to calculate loss expressions. During training, we grouped series of five instructions into one batch, took the average loss for each expression tied to an instruction, propagated this loss back to readjust network weights, and then reset batch loss.

The stopping criteria for making an interaction-level prediction was based on whether a prediction was even possible for a given environment configuration and, if so, making a sequence of action predictions until hitting the <end> tag. The stopping criteria for training the model was determined by the number of epochs. The training data was bundled into a list of instructions, actions, environment states, and IDs.

The graph below visualizes the encoder-decoder sequence discussed earlier. Within each of the encoder cells is a recurrent neural network (RNN), which we chose to represent as an LSTM, since instructions and actions can be determined by long term memories.




$$
\begin{aligned}
i_t &= \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \\
f_t &= \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f + 1) \\
o_t &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \\
\tilde{c_t} &= \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \\
c_t &= c_{t-1} \circ f_t + \tilde{c_t} \circ i_t \\
h_t &= \tanh(c_t) \circ o_t
\end{aligned}
$$

## 4 Data (5pt)

Shallow statistics of the data can be found in the table below. The filtered instructions vocab size represents the number of instruction words that appeared more than 7 times throughout the entire training set. Words that appeared less than 7 times were tagged as unknown via the '<UNK>' key. Further analysis revealed that several of these rare words were typos of relevant words. For instance, *thirds* was misspelled as *thrids*, while *beaker* was misspelled as *beeker* and *breaker*. Therefore, the total number of unknown words was calculated as the difference between the length of the corpus after the above filter was applied. In total, there were 610 - 245 = 365 unknown words in training set, 133 - 74 = 59 unknown words in dev set, and 252 - 121 = 131 unknown words in the test set.

Table 1: Data Statistics

| Metric | Value |
|---|---|
| Instruction and Interaction Size (Training) | 18285 |
| Raw Instructions Vocab Size (Training) | 148568 |
| Raw Actions Vocab Size (Training) | 77449 |
| Unique Instructions Vocab Size (Training) | 610 |
| Unique Actions Vocab Size (Training, Dev) | 51 |
| Filtered Instructions Vocab Size (Training) | 245 |
| ID Size (Training) | 3657 |
| Instruction and Interaction Size (Dev) | 1225 |
| Filtered Instructions Vocab Size (Dev) | 74 |
| ID Size (Dev) | 245 |
| Instruction and Interaction Size (Test) | 2245 |
| Filtered Instructions Vocab Size (Test) | 121 |
| ID Size (Test) | 449 |

For data preprocessing, I extracted sentence instructions (e.g. *throw out first beaker*), action sequences (i.e. *pop x, push y o*), environment stage configurations, and ID's (i.e. *train-A9164*) for the training, development, and test data. I also vectorized words by creating a *vocab_dict* dictionary that mapped all words from sentence-level instructions to integers and a *vocab_actions_dict* that mapped all words from action sequences to integers, from the training set. Finally,

I created a *conv_char_to_int* dictionary that vectorized the tagged beaker colors as follows: {_: 0, 'b': 1, 'g': 2, 'o': 3, 'p':4, 'r':5, 'y':6}.

## 5 Implementation Details (2pt)

The final sequence-to-sequence model used simple stochastic gradient descent (SimpleSGD) as the optimization algorithm. It comprised of the following hyper-parameters and their values:

```
1 LSTM_NUM_OF_LAYERS = 1
2 EMBEDDINGS_SIZE_INSTRUCTION = 50
3 EMBEDDINGS_SIZE_ACTION = 50
4 EMBEDDINGS_SIZE_BEAKER_STATE = 50
5 HIDDEN_SIZE_ENCODER_1 = 100
6 HIDDEN_SIZE_ENCODER_2 = 75
7 HIDDEN_SIZE_DECODER = 100
8 ATTENTION_SIZE = 100
9 EPOCHS = 50
```

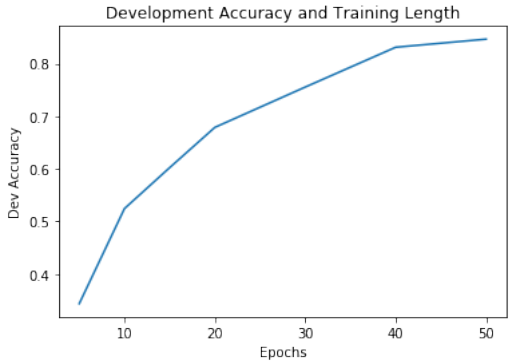
During the results and experimentation phase, the following variables were fine-tuned to optimize performance: embedding size (for instruction, action, beaker state representations), DyNet builder, model optimizer, hidden layer size, and number of epochs.

## 6 Experiments and Results

**Test Results (3pt)** We generated this test score using our procedure for handling unknown words described above. The hyper-paramater values were $d = 50$ for input embeddings and $h = 100$ for attention size.

| Builder | Optimizer | Test Accuracy |
|---|---|---|
| LSTM | SimpleSGD | 0.48552 |

**Ablations (15pt)** Number of epochs had a directly positive correlation with instruction-level accuracy on the development data, as seen from the plot below.




| Method | Optimizer | Instruction Acc |
|---|---|---|
| VanillaLSTM | SimpleSGD | 0.5492 |
| CompactVanillaLSTM | SimpleSGD | 0.4985 |
| CoupledLSTM | SimpleSGD | 0.4456 |

Progressive task completions played a crucial role in increasing instruction-level and interaction-level accuracy, with state information producing the highest jump in accuracy, as seen from the table below.

| | Accuracy | |
|---|---|---|
| **Task** | Instruction | Interaction |
| Seq2Seq Model | 0.1285 | 0.0143 |
| W/ State Info | 0.4458 | 0.2271 |
| W/ Attention | 0.4944 | 0.2619 |
| W/ Interaction History | 0.5492 | 0.2835 |

Different model optimizers yielded noticeable performances in instruction-level and interaction-level accuracy, with SimpleSGD netting the best overall accuracy.

| | Accuracy | |
|---|---|---|
| **Optimizer** | Instruction | Interaction |
| Adam | 0.5286 | 0.2628 |
| SimpleSGD | 0.5492 | 0.2835 |
| CyclicalSGD | 0.5161 | 0.2543 |
| MomentumSGD | 0.4677 | 0.2179 |
| Adagrad | 0.4827 | 0.2394 |

## 7 Error Analysis (7pt)

We identified several classes of error in interpretation. For instance, with transferance between containers, there exist inaccurate key phrases that the model fails to recognize such as commands that indicate multiple repeated actions such as "add remaining" or "empty into". However, the model seems to perform better on commands that specifically dictate the quantity of transferance, e.g. "pour two". This class of "vague" actions seem to trick the model whereas concise instructions with a discrete value are interpreted well. Similarly, actions that dictate relational modifiers perform worse than ones targeting a single beaker by color, e.g. "rightmost" vs "orange chemical" as there is an extra step of comparing the relation of the entire set which introduces complexity.

## 8 Conclusion (3pt)

The largest improvement in sequence-to-sequence model performance stemmed from adding state information (Task 2 in the spec report). Specifically, the instruction-level accuracy jumped from — to — after incorporating the initial state information. Furthermore, adding state information, attending on color states and instructions, and adding history interaction utterance, enriched contextual knowledge gleaned during neural network training phase, and also provided substantial improvements. Fine-tuning hyperparameters (e.g. number of hidden layers, embedding dimensions, attention method, optimizer, number of epochs) proved crucial in making incremental performance progress. Next steps would be to increase the $n$-gram size for interaction history to consider multiple prior instructions ($n$>1) instead of only the previous instruction ($n$=1), thereby providing more contexts for pronouns and their antecedents, as well as exploring other attention mechanisms.