CS5740: Assignment 4
Github Repository Link (a4–group-11-1)

Hangyu Lin hl2357

Rashmi Sinha rs2584

Tianyun Zou tz392

1 Introduction (5pt)

We are required to implement a sequence to sequence (referred as seq2seq hereafter) model to map instructions to action in the Alchemy environment. Each example has a sequence of instruction which are mapped to action *push* and *pop* in an environment containing 7 beakers and 6 colors. Each beaker is treated as a stack.
We experimented various versions of seq2seq models in terms of architecture by adding state information, attention and instruction history. We also used GRU and LSTM and experimented with various hyperparameters. The best score we get on dev set is 0.772 on instruction and 0.232 on interaction. And on test set is 0.44766 on interaction, using LSTM, with attention, world state, previous interaction history.

2 Model (35pt)

We first implemented a simple seq2seq model with encoder-decoder architecture. In each step, the instruction encoder ${R_E}^I$ takes the output of the embedding function E for the word x_i in the instruction X and the hidden data $\mathbf{s}_{i-1}^E$ from the previous step as input. The hidden state of the final step of the instruction encoder s_n^E is used as context to input to the decoder. For the decoder, in each step, we concatenate the output of the last predicted item $\hat{t}_{j-i}$ with the context c from the encoder as the current information, and then we send it and hidden data $\mathbf{s}_j^D$ from last step to our decoder.

$$\mathbf{X} = x_1, ..., x_n$$

$$\mathbf{s}_i^E = {R_E}^I(\mathbf{s}_{i-1}^E, \mathbf{E}_{[x_i]}), i = 1, ..., n$$

$$\mathbf{c} = O_E(\mathbf{s}_n^E)$$

$$\mathbf{s}_j^D = R_D(\mathbf{s}_{j-1}^D, [\mathbf{E}_{[\hat{t}_{j-1}]}; \mathbf{c}])$$

$$O_D(\mathbf{s}_j^D) = LogSoftmax(\mathbf{s}_j^D \mathbf{W} + \mathbf{b})$$

$$\hat{t}_j = argmax O_D(\mathbf{s}_j^D)$$

We go on to further add the current world state of beakers to allow model to learn colors in a beaker. Hence we encode the world states using the world state encoder ${R_E}^S$. This would operate similar to the instruction encoder, the only difference is that it takes the current world state w_i for each step instead of the instruction.

$$\mathbf{s}_{S,i}^E = {R_E}^S(\mathbf{s}_{S,i-1}^E, \mathbf{E}_{[w_i]}), i = 1, ..., n$$

We also have two encoders ${R_E}^P 1$ and ${R_E}^P 2$ which are very similar to ${R_E}^I$. The only difference is that they take the previous instruction $x_{P1,i}$ and the second previous instruction $x_{P2,i}$ as input.

$$\mathbf{s}_{P1,i}^E = {R_E}^I(\mathbf{s}_{P1,i-1}^E, \mathbf{E}_{[x_{P1,i}]}), i = 1, ..., n$$

$$\mathbf{s}_{P2,i}^E = {R_E}^I(\mathbf{s}_{P2,i-1}^E, \mathbf{E}_{[x_{P2,i}]}), i = 1, ..., n$$

Next, we go on to add attention in our decoder to produce a weighted vector that represent each step of the encoders that the decoder should consider. The attention model would learn about what information to keep from the encoders' hidden state outputs, and we use the operations in below equations to apply it to the concatenated vector of all four hidden state outputs from the encoder and the previous output in the decoder model.

$$\overline{\alpha}_i^j = \mathbf{s}_{j-1}^D \overline{\mathbf{c}}_i$$

$$\alpha^j = softmax(\overline{\alpha}_1^j, \ldots, \overline{\alpha}_n^j)$$

$$\mathbf{c}_j = \sum_{i=1}^n \alpha_i^j \overline{\mathbf{c}}_i$$

$$\mathbf{s}_j^D = R_D(\mathbf{s}_{j-1}^D, [\mathbf{E}_{[\hat{t}_{j-1}]}; \mathbf{c}_1; \mathbf{c}_2; \mathbf{c}_3; \mathbf{c}_4])$$

Hence our final architecture involves separate encoders for beaker state, current instruction,

Dataset	original vocab size	vocab size after preprocess	number of examples	avg instruction length	instruction counts
Train Set	694	615	3657	8.12	18285
Dev Set	138	137	245	6.37	1225
Test Set	268	256	449	7.83	2245

Table 1: Data Overview

last instruction and second last instruction. The weighted contexts obtained from these encoders using attention is concatenated with the current encoding to feed into the decoder. Teacher forcing is applied to help the learning of the weights which is then used for predicting the outputs.

The model architecture is shown in Figure 1.

3 Learning (8pt)

For LSTM model, CrossEntropyLoss function was used along with SGD optimizer for both the encoder and decoder, and LogSoftmax as activation function. For GRU model, everything was kept same as LSTM except using NLLLoss as loss function. The stopping criteria was minimum number of epochs to get instruction level accuracy improve of more than 0.01. We observed that we had to run our final model on 25 epochs. For training data, for instructions we create a hashmap from indices of all the words present in the corpus. As actions are limited in scope, we create a dictionary of 16 (7 beakers, 6 colors, 2 actions and no color) irrespective of presence in training corpus. We treat test data analogous to training data apart from using the training corpus to index the elements in the test data.

4 Data (4pt)

We use the entire training data available for training. We tokenize the utterance into words delimited by space. Each word is lower cased. We remove all punctuation present in the words as they add noise. Refer table 1 for all the statistics before and after processing. For unknown words, we use replace all the words in the vocabulary less than 10 with *unk* token and use this token for all unknown words encountered in dev and test data.

5 Implementation Details (2pt)

The model has the hyperparameters involving unknown word threshold, learning rate and 2 hidden sizes (1 for world state encoder, other for remaining encoders), batch size, number of epochs, RNN units (LSTM or GRU) and loss functions. We used batching across various examples to speed up the processing. We made use of padding tokens to make lengths of tensors across examples in batches same. Masking was used to cover the padding positions when using attention, where we used a mask of 0 for actual instruction and negative infinity for padding positions. Dropout was also used in the LSTM and before the linear layer of the decoder to prevent overfitting.

6 Experiments and Results

Test Results (3pt) The best score of our model is 0.44766 on test set, using 1 encoder world state with hidden size 32 and 3 encoders for current, last and second last instruction with hidden size 128, learning rate = 0.01, 25 epochs, batch size = 1 and LSTM RNN unit.

Ablations (13pt) The results of ablations experiments are shown in Table 2. We start ablating initially with the various model architectures described in the model section as we believe that model architecture will cause the maximum impact on performance. We keep the hyperparameters fixed and decide to tune on them once we finalise the model as shown in the table. We also first used a 2D matrix (number of beakers * number of colors) to quickly encode world state (2D WS). We observed that adding the 2D WS, attention (Attn), last instruction (Prev1) and second last instruction (Prev2) each helped increase the performance incrementally. Also we observed LSTM as expected to be more powerful and hence discarded using GRU. Finally we added encoding of current state using encoder RNN which also marginally helped improve performance. The maximum impact was observed on adding the world state information as was expected because the model is almost blind without this information and does not understand which beaker is being talked about. Also to our surprise encoding the beaker state caused minimal improvement in the performance even though color ordering in beaker was now being accounted for. Finally we start ablating with the model hyperparameters. We observed a learning rate of 0.01 to be appropriate, changing it beyond that almost caused slight decrease and slowed down the training process. Significant jump was observed on increasing the hidden size of the encoders handling english utterances and number of epochs which is expected as the expressivity of the model improves. Another

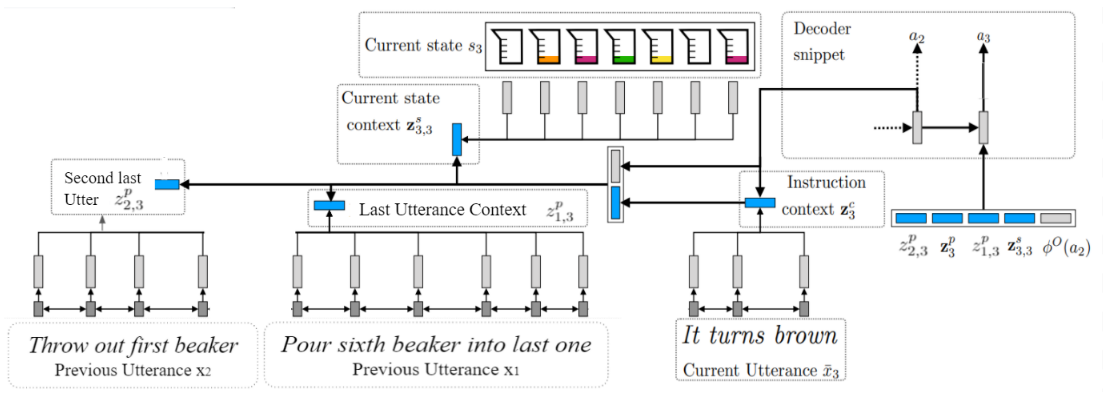

Current state s_3
Decoder snippet
a_2
a_3
Current state context $\mathbf{z}^s_{3,3}$
Second last Utter $z^p_{2,3}$
Last Utterance Context $z^p_{1,3}$
Instruction context $\mathbf{z}^c_3$
$z^p_{2,3}$ $\mathbf{z}^p_3$ $z^p_{1,3}$ $\mathbf{z}^s_{3,3}$ $\phi^O(a_2)$
Throw out first beaker
Previous Utterance x2
Pour sixth beaker into last one
Previous Utterance x1
It turns brown
Current Utterance $\bar{x}_3$

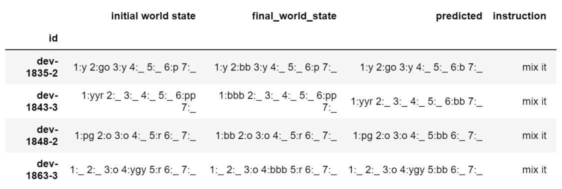
Figure 1: Architecture of Seq2Seq model.

thing we observed was the continuous fluctuation in results which is expected due to randomization present in the models. Also batching helped increase the speed of the processing, but resulted in lower accuracy relatively.

System	Ins Sc	Int Sc
Development Results		
n=5, LR=0.1, HD1=64, HD2=32, unk=10, b= 1		
Vanilla Seq2seq (GRU)	0.073	0.001
Seq2seq+2D world state(2D WS) (GRU)	0.324	0.072
Seq2seq+2D WS (LSTM)	0.451	0.132
Seq2seq+Attn+2D WS (LSTM)	0.503	0.125
Seq2seq+Attn+2D WS+Prev1 (LSTM)	0.532	0.155
Seq2seq+Attn+2D WS+Prev1+Prev2 (LSTM)	0.577	0.161
Seq2seq+Attn+EncWS+Prev1+Prev2 (LSTM)	0.586	0.165
Ablating params with above (last) model and b = 1		
n= 5, LR= 0.01, HD1= 64, HD2= 32,unk= 10	0.602	0.167
n= 5, LR= 0.001, HD1= 64, HD2= 32,unk= 10	0.593	0.165
n= 5, LR= 0.01, HD1= 128, HD2= 32,unk= 10	0.652	0.173
n= 10, LR= 0.01, HD1= 128, HD2= 32 ,unk= 10	0.732	0.199
n= 25, LR= 0.01, HD1= 128, HD2= 32,unk= 10	0.772	0.232
Ablating params with above (last) model and b = 5		
n= 25, LR= 0.01, HD1= 128, HD2= 32,unk= 10	0.702	0.182
Test Results		
n=5, LR=0.1, HD1=64, HD2=32 ,unk=10		
Vanilla Seq2seq (GRU)	NA	0.0023
Seq2seq+2D world state(2D WS) (GRU)	NA	0.155
Seq2seq+2D WS (Incorrect Attn+Prev1) (LSTM)	NA	0.175
Ablating params with above (last) model		
Seq2seq+Attn+2D WS+Prev1+Prev2 (LSTM)	NA	0.447

Table 2: Correlation Scores for various models Ins Sc: Instruction level accuracy; Int Sc: Interaction level accuracy; n: epochs; LR: Learning rate; HD2: hidden layers in world state encoder; HD1: hidden layers in other encoders; b: batch size; unk: unknown word threshold

7 Error Analysis (7pt)



id	initial world state	final_world_state	predicted	instruction
dev-1835-2	1:y 2:go 3:y 4:_ 5:_ 6:p 7:_	1:y 2:bb 3:y 4:_ 5:_ 6:p 7:_	1:y 2:go 3:y 4:_ 5:_ 6:b 7:_	mix it
dev-1843-3	1:yyr 2:_ 3:_ 4:_ 5:_ 6:pp 7:_	1:bbb 2:_ 3:_ 4:_ 5:_ 6:pp 7:_	1:yyr 2:_ 3:_ 4:_ 5:_ 6:bb 7:_	mix it
dev-1848-2	1:pg 2:o 3:o 4:_ 5:r 6:_ 7:_	1:bb 2:o 3:o 4:_ 5:r 6:_ 7:_	1:pg 2:o 3:o 4:_ 5:bb 6:_ 7:_	mix it
dev-1863-3	1:_ 2:_ 3:o 4:ygy 5:r 6:_ 7:_	1:_ 2:_ 3:o 4:bbb 5:r 6:_ 7:_	1:_ 2:_ 3:o 4:ygy 5:bb 6:_ 7:_	mix it

We compared the predictions using the actual previous final world state with the actual labels on the development set and was able to identify a few classes of errors made by our model. One example that our model did horrible on is the instruction ”mix it”. This instruction appeared 19 times in the development set, we got them all wrong. Our model would just change the colors in some beakers to another color with no real pattern. To be fair, this is a complicated instruction for machines to learn. The model does not understand to mix the colors to create new colors in each beaker, it is fed in with a bunch of actions such as ”pop 1 pop 1 pop 1 push 1 b push 1 b push 1 b” for training for the instruction of ”mix it”. Which confuses the model as it is a lot of different actions that is produced by the a short instruction.

8 Conclusion (3pt)

Our experiments showed that the most important part of the model is to include the initial world state. The model would be guessing blindly about the actions if the instruction only mentioned the beaker position and not the colors without the world state. We found that by including attention and previous instruction history also improved the accuracy, especially when predicting the entire interactions. Also it was interesting how our performance on the test set, i.e 0.447 was way better than the performance on the dev set i.e 0.232. Moreover, we found that model does not perform well with short instructions that has complex actions. Due to time constraint, we were not able to implement all of our ideas.