CS5740: Assignment 4

https://github.com/cornell-cs5740-21sp/a4--a4-group-16

Moshe Borouchov mb2686 | Yiling Sun ys994 | Xiyue Shi xs356

1 Introduction (5pt)

In the assignment, we built a sequence to sequence model with attention to map instructions to actions in the ALCHEMY environment. The ALCHEMY environment has seven beakers, and the color liquid from one beaker could be moved to another beaker, mix with others, or remove from the environment. Each beaker is treated as a stack with two types of actions: pop and push. The data is in JSON format, where each example is a dictionary with keys for the initial environment and the utterance list.

Using the provided training and development data, we experimented with different network architectures, embedding approaches, with or without state information, with or without attention, and parameters. As in Section 6, with LSTM world state and bi-directional LSTM encoder, our model attained 56.3% accuracy on the instruction-level development data, 30.1% and 55.01% accuracy on the interaction-level development and test data.

2 Model (35pt)

Our sequence to sequence model is based on the recurrent neural network (RNN) encoder-decoder architecture with attention. For each example i, The model maps sequences of instructions $\bar{x}_i = \langle x_{i,1}, ..., x_{i,n} \rangle$ to actions $a_1, ..., a_k$. At each step k of executing the i-th instruction, the input to the model is the current instruction x_i, the world state s_i at the beginning of executing x_i and the previous action a_{k-1} if we would include interaction history. The model predicts the next action a_k to execute.

For each token $x_{i,j}$, the instruction encoder computes a hidden state $h_j^E = [h_j^{\overrightarrow{E}}; h^{\overleftarrow{E}_j}]$ using a bi-directional LSTM RNN. The forward LSTM RNN computes its hidden state as:

$$h_j^{\overrightarrow{E}} = LSTM^{\overrightarrow{E}}(\phi^x(x_{i,j}); h_{j-1}^{\overrightarrow{E}}), \tag{1}$$

where ϕ^x is a learned word embedding function for input instructions and $LSTM^{\overrightarrow{E}}$ is the long-short term memory recurrence function. The backward hidden states is computed with a similar equation.

Similarly, a vector is generated for each beaker using an bi-directional LSTM. The world state vector is represented as the concatenation of the seven beakers encoding with the beaker index encoding.

For model with attention, the decoder at step k's state is computed with the equation:

$$h_k^D = LSTM^D(\tanh[\phi^y(a_{i,k}, c_k)]W^d + b^d; h_{k-1}^{\overrightarrow{E}}), \tag{2}$$

where ϕ^y is a learned word embedding function for output actions and $LSTM^D$ is the long-short term memory recurrence function, and c_k is an attention vector computed from the encoder states.

The initial hidden state and cell memory of the decoder are initialized with zeros. The attention vector c_k is the concatenation of all computed context vectors.For the model, we compute a context vector c using the Bahdanau-style additive attention. For example, the context vector of h_j^E for the k-th action would be:

$$c_{k,h_j^E} = \sum_{i=1}^{N=1} \alpha(h_k^D, h_j^E) h_j^E, \tag{3}$$

where there are N tokens in the input instruction. The query is the decoder hidden state h_k^D, and both key and value are the encoder hidden states h_j^E. Given a query q and key k, the attention weight α is computed using the additive attention scoring function:

$$\alpha_{q,k} = W_v^T \tanh(W_q q + W_k k), \tag{4}$$

The context vector for the world state is computed with the same equation. $[h_k^D; c_{k,h_j^E}]$, the concatenation of the decoder hidden state h_k^D and the context vector of h_j^E, is the query. and both key and value are the world state encoder hidden states. The high-level architecture of the model is shown in the Fig 1.

However, our attention model does not work. We spent time to debug and tried different variations, but it is still inaccurate. So we test models without

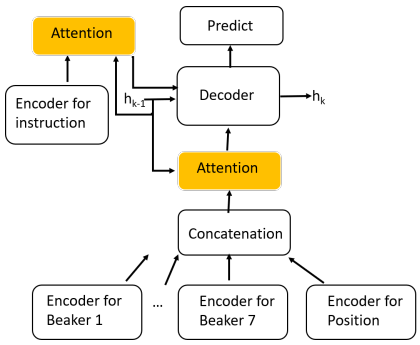

Figure 1: The architecture of the encoder-decoder network with attention.

the attention that the initial hidden state of the decoder is initialized with the concatenation of the final hidden state of the instruction encoder and beaker encoders and the same for the cell memory. We experiment with this simple model by adding interaction history by computing another vector for the previous action with its encoder. The decoder's hidden state is the concatenation of all encoders' hidden states.

3 Learning (8pt)

The training data contains the instructions as the key input, actions as the key output, with the world state information of beakers at each step. While in the test data, we only have the world state information of the initial states. When training, we have ground truth data from previous like previous world states and previous actions, while those information is lost in testing that we make prediction based on the last prediction generated by the model. So, as farther the prediction on testing goes from the initial, our model become less robust when measuring the performance of the last states. We predict the output actions with an intermediate hidden layers generated by the encoder with selected informative input. We don't measure the performance of encoder directly. We optimize the model by calculating the loss of output actions with **CrossEntropyLoss**. The formal definition of the loss from PyTorch is

$$loss = \frac{\sum_{i=1}^{N} loss(i, class[i])}{\sum_{i=1}^{N} weight[class[i]]}, \quad (5)$$

where N is the number of class, and weight is assigned to each class. The loss of each class is calculated by

$$loss(x, class) = -log(\frac{exp(x[class]}{\sum_j exp(x[j]}), \quad (6)$$

where x is the probability output across all possible outputs and class is a specific class of output. Then,total loss is calculated by We assign equal weights to all class of output. This loss is particularly useful for our work because the output length could be varied for seq2seq model. We stop learning when the loss is not significance that lower than 1% decreases, usually is around 15 epochs.

4 Data (4pt)

Data are consists of sets of interaction. Each interaction includes a key identifier ('train-A9164') and an 'utterances' with contains five instruction level steps. In the utterance, each utterance contains an **after env** ("1:g 2:_ 3:_ 4:_ 5:_ 6:ooo 7:gggg"), world state after action, **instruction** in words ("pour sixth beaker into orange one") and **action** correspond to the instructions ("push 5 g", "pop 6"). There are seven beakers, each beakers has four units of capacity and there are 6 colors: yellow, orange, red, green, blue and purple. As in section 5 , we encode it with two different methods. Actions are treated as a three tokens value: (POP, N, None), (PUSH, N, color), where 1<=N<=7, color belongs to one of the color spaces. Table 4 shows the shallow statistics of our data. Notice the numbers are before padding and we also added **'SOS'** and **'EOS'** on instructions and actions to represent the start and end of sentences. In the data, all instructions are lower case. Since we have limited number of unknown words in dev and test, we treated words with occurrences less than 2 as **UNK**. We tokenized the instructions and actions and train the embedding with embedding size 50 for both. We removed all the punctuation before tokenization and keep words in their original shape.

Data Size	Train	Dev	Test
Num. of Instructions	18285	1225	2245
Num. of Interaction	3657	245	248
Instruction Vocab. size	569	135	2245
Ave. Instruction Length	53.3	43.2	51.5
Instruction Unknown words	-	1	16
Actions Vocab. size	49	49	49

Table 1: Shallow statistics of our three datasets.

5 Implementation Details (2pt)

To train the model, each time we read in one instructions and its current world state, with **previous instructions**, **previous actions** during experiments. For the world state information, we experiment with one-hot-encoding with dimension 168 and LSTM encoding of 7 beakers and concatenate with embedded position vectors. The encoder/decoder parameters used for experiments are shown in Table 2. For all the encoder/decoder, we used LSTM with 1 hidden layer and drop-out rate 0.05. We initialized the hidden state and cell states of LSTM as zeors. After concatenating the hidden states of encoder output, we then concatenate with the embedded output actions as the input of decoder. For training purpose, at each time step, we applied **tearcher forcing** with

0.5 rate that 50% probability that the model takes the ground truth as the next input of decoder, otherwise take the best guess from the model as next input. While when testing, we always takes the best guess.

We trained with Cross Entropy Loss and Adam Optimizer with learning rate 0.01. To optimize the speed and use of memory, we batch with batch size 50 during training. We sorted the data based on the length of instructions. So, in every batch, the length of instructions are similar that we don't need lots of extra padding. Furthermore, as we input data into LSTM encoder and generate hidden state for use, when experimenting without attention, instead of taking the last layer of hidden state, we output the hidden state of last not padding tokens.

Encoder/Decoder	Embedding	Hidden layer
Instruction	100	100
Action	50	100
World State	100	100
Position	50	7

Table 2: Parametes used of models.

6 Experiments and Results

Test Results (3pt) The best rest result from the leaderboard has an interaction accuracy of 0.03341 on the test data. The model has one instruction encoder and seven encoders for the seven beakers that each represents a beaker. The model does not use attention. Instead, hidden states of all encoders are concatenated to a vector and the same for cell memory. They are feed into the decoder to initialize the decoder's input. All RNN embedding and hidden layer parameters are as described in Section 5.

Ablations (13pt) We have experimented with the variations of model as shown in Table 3

Interaction accuracy is not reported in the table, because as the instruction prediction is not accurate, the interaction prediction is not reliable that each time we test with the same model, it would produce a different result, within the range from 0.03 to 0.065.

The ablation shows that while keeping the rest of the model the same, accessing state information and previous action could increase the prediction accuracy. However, attention does not work for our model.

7 Error Analysis (7pt)

Table 5 shows the prediction accuracy by the length of instruction. The accuracy was especially low when the instruction is too low or too long. For the short instruction, the main error we noticed appears when the encoded instruction refers to a beaker manipulated in the previous utterance. For example, when

Model	Model Description
Model 1	Baseline, which has no access to the state information and the encoder-decoder network is trained with instructions alone.
Model 2	Create one-hot-encoding for beaker states. Without attention. Use uni-directional LSTM for encoder.
Model 3	Create LSTM encoding of 7 beaker states. Without attention. Use uni-directional LSTM for encoders.
Model 4	Create LSTM encoding of 7 beaker states. Withoutate LSTM encoding of 7 beaker states, without attention. Use bidirectional LSTM for encoders.
Model 5	(as described in the model section) Create LSTM encoding of 7 beaker states. With attention. Use bidirectional LSTM for encoders.
Model 6	Create LSTM encoding for previous action. Create LSTM encoding of 7 beaker states. Without attention. Use uni-directional LSTM for encoders.
Model 7	Create LSTM encoding for previous action. Create LSTM encoding of 7 beaker states. Without attention. Use bi-directional LSTM for encoders.

Table 3: Development test results of our models.

Model	Dev Instruction Acc.	Interaction Acc.
Model 1	0.148	0.005
Model 2	0.401	0.113
Model 3	0.409	0.121
Model 4	0.563	0.301
Model 5	0.216	0.081
Model 6	0.492	0.122
Model 7	0.451	0.117

Table 4: Development results of our models.

an instruction is only "mix it," referring to the instruction of the prior utterance - "then add the leftmost beaker of green chemical to the yellow chemical," our model doesn't manage to deduce which beaker it should manipulate. While our model uses previous instructions and world states, it seems our architecture fails to deduce which beaker should be manipulated when it is referred to in third person pronoun.

Length	2∼3	4∼8	9∼13	14∼21
Num. of Instr.	290	800	194	29
Num. of correct	2	297	26	3

Table 5: Accuracy by length of instructions

8 Conclusion (3pt)

Our best performing model uses bi-directional LSTM as the encoder and unidirectional LSTM as the decoder and embeds the current instruction and the world state without attention. It achieves an instruction accuracy of 0.563 on the development data.

Surprisingly, attention does not improve our model performance. But from the simple model with concatenating of hidden layers of encoders for instruction and world states, it show that accessing state information could improve the performance.