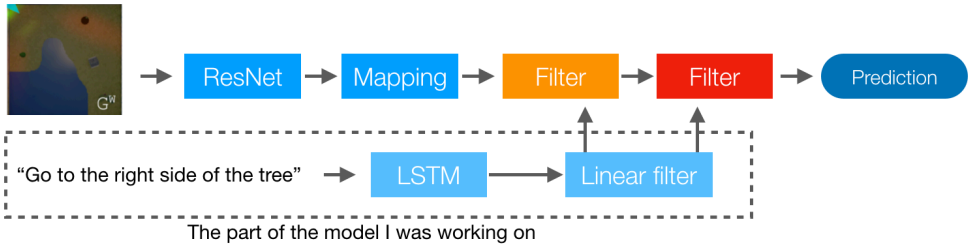
Harrison Zhao

**Introduction**

For this independent study, I was mainly working on instruction embedding for this project. I implemented several RNNs with attention mechanism to encode instructions. In the following sections, I will carefully go over each model or techniques I implemented.




**BILSTM with self-attention**

The first model I implemented is Bidirectional LSTM with self-attention mechanism (*Zhouhan Lin, 2017*). It used to take the last hidden layer of the LSTM as embedding but it may not be able to capture information at the beginning of the sentence. BiLSTM encodes the sentences both forward and backward that can extract more information from the sentence. Then, I pass those hidden layers to attention. The attention has multiple attention heads (each row of the attention matrix) that could learn different aspect of the instruction, such as some are respect to direction and some are respect to objects.

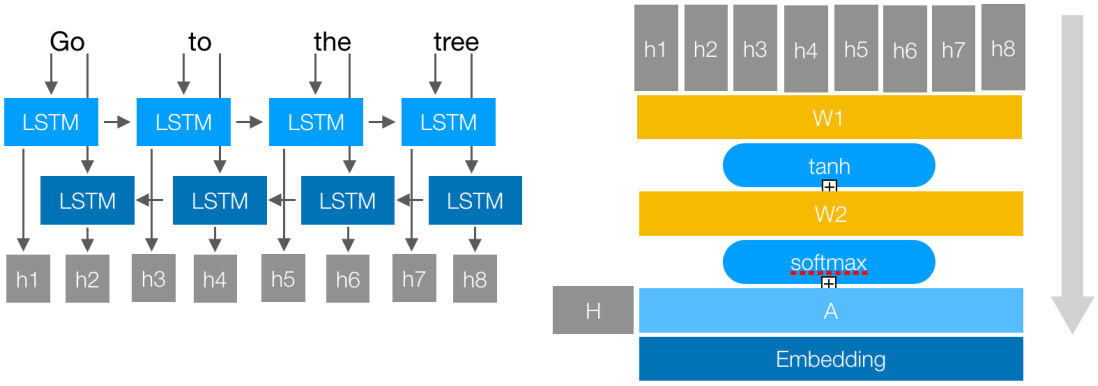
Basically, for attention, the input is hidden state H and the output is matrix A such as:

$$A = softmax\left(W_{s2} tanh\left(W_{s1} H^T\right)\right)$$

Then, I multiple the hidden state H by the attention matrix A to get instruction embedding M:

$$M = AH$$

The below diagram shows the workflow:




**Regularization**

Without regularization, those attention heads all learned the same thing so all the attention heads have the same weight (figure 1). However, we want each attention heads to focus on the different aspects of the instruction so I added the regularization term to loss.

The first regularization I tried was mentioned in the self-attention paper. It is L2 norm of the attention matrix A and subtract by an identity matrix:

$$P = \left\| \left(AA^T - I\right) \right\|_F^2$$

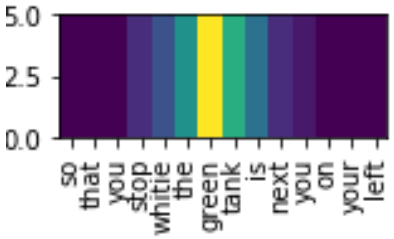



*Figure 1*
*(Note: the y axis is different attention heads)*

If two rows of attention matrix are focused on different aspects, the multiplication of those two rows will be small, which leads to a lower loss. Be subtracting the identity matrix, it will not penalize each attention head itself.
With this regularization term, the attention heads start to focus on different aspects but the attention matrix become very sparse (figure 2). By subtracting the identity matrix, it also limits the variance across each attention head. However, in our case, we want each attention heads to take in account of multiple hidden states.

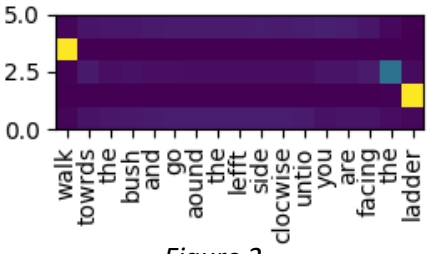


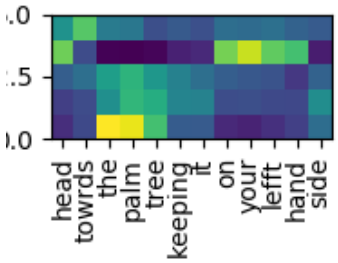

*Figure 2*

To solve this problem, I zero outed the diagonal of $AA^T$ matrix:

$$\mathrm{P} = \|(AA^T)^*\|^2 \quad * where\ (AA^T)_{ii} = 0$$

So, the attention can focus on different aspects of the instruction meanwhile not limiting the variance of each attention head. As a result, some attention heads learned to focus on objects and some learned to focus on actions (figure 3&4)

*Figure 3*



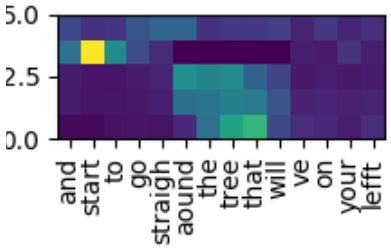
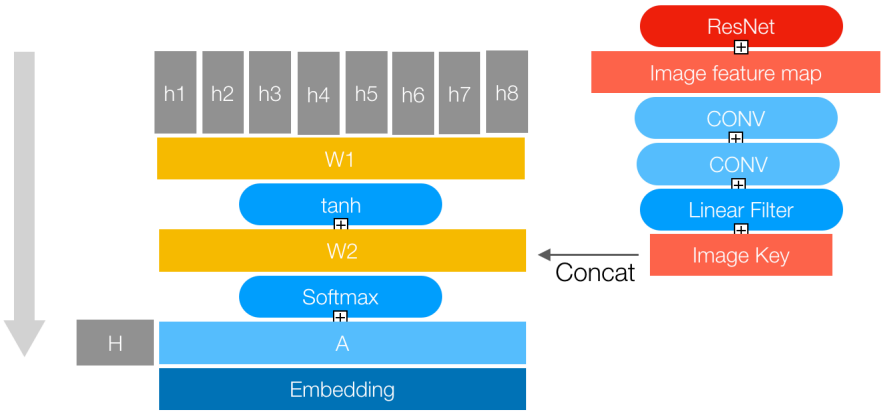




*Figure 4*

For an example, in figure 3 & 4, we can roughly see that the first-row focus on action “head towards”. The second row learned the which side of the object should it goes to. The last row learned the which objects was mentioned in the instruction.

**Conditional Self-Attention**

In the self-attention architecture, instruction embedding and image feature are two independent pipelines. In other words, the attention matrix doesn’t not take in account of the image. However, we think, by adding an image key to the attention matrix, it can learn to weight the hidden states dynamically based on the current image. The architecture is following:




Mathematically:

$$\mathrm{A} = \mathrm{softmax}([\mathrm{W}_{s2}\tanh(W_{s1}H^T), Key]_{cat})$$

**Performance**

Then, we integrated this self-attention embedding into the whole model and ran some experiments.

| | LSTM Layer | LSTM Size | Embedding Size | Attention Heads | Conditional Self-attention | Goal Region Score* |
|---|---|---|---|---|---|---|
| Model 1 | 1 | 40 | 300 | 5 | NO | 0.5888 |
| Model 2 | 2 | 40 | 300 | 5 | NO | 0.5420 |
| Model 3 | 1 | 40 | 300 | 5 | YES | 0.5523 |

*Goal Region Score: we used goal region score to evaluate the performance. It will compare the end point of the predicted trajectory with the goal region.

I also ran some interactive predictions with self-attention embedding and here are some examples:

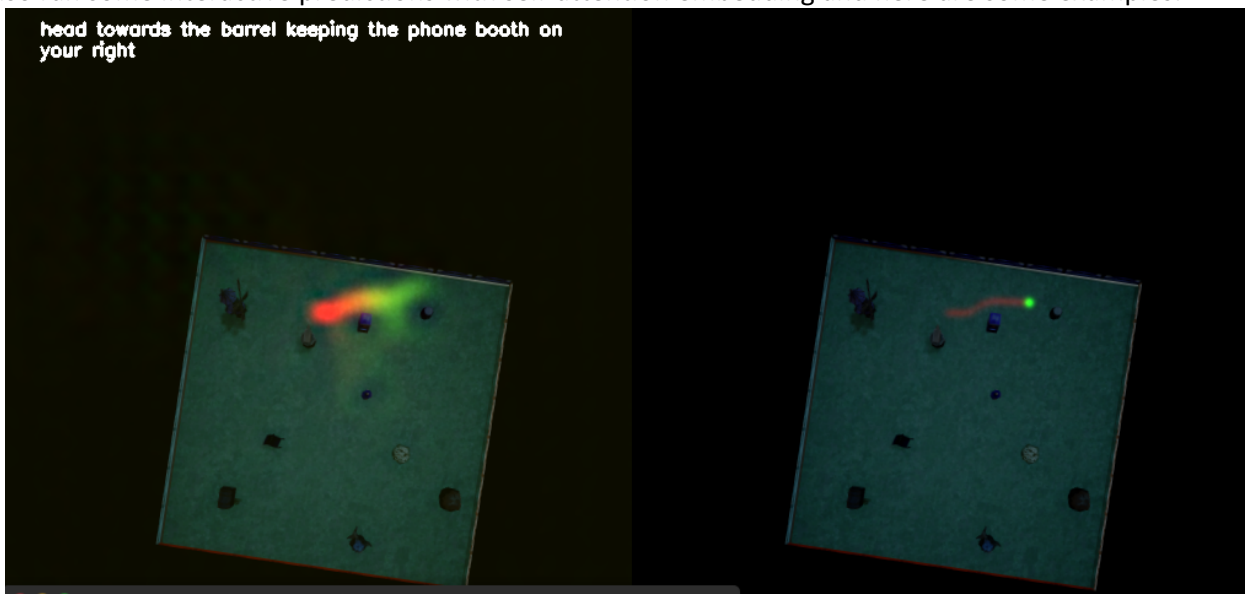



*figure 5*

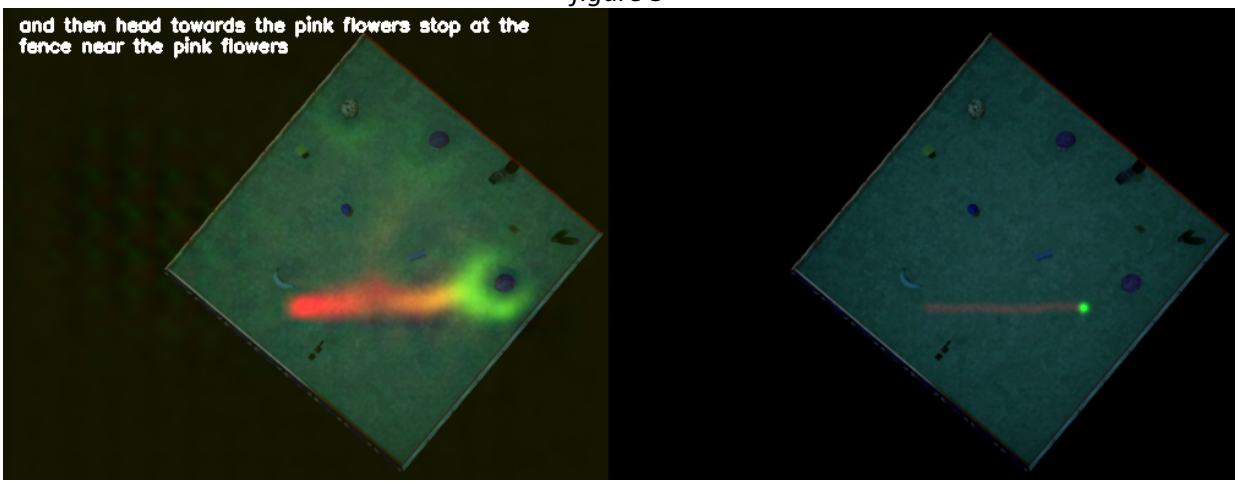



*figure 6*

From figure 5 and 6, we can see that it can predict the correct end point. From figure 5, we can also see it can predict the correct trajectory. It can understand “keeping the phone booth on your right.”

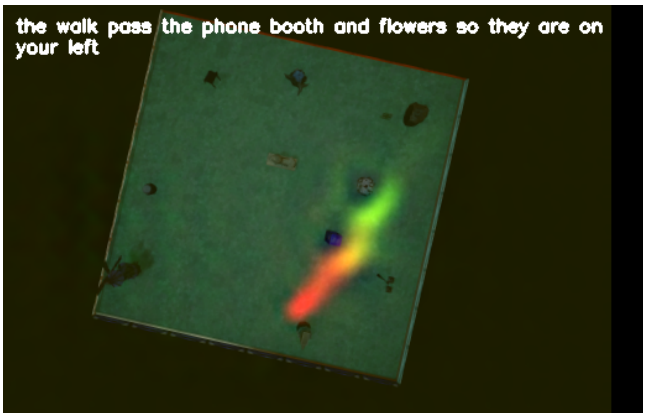



*figure 7*

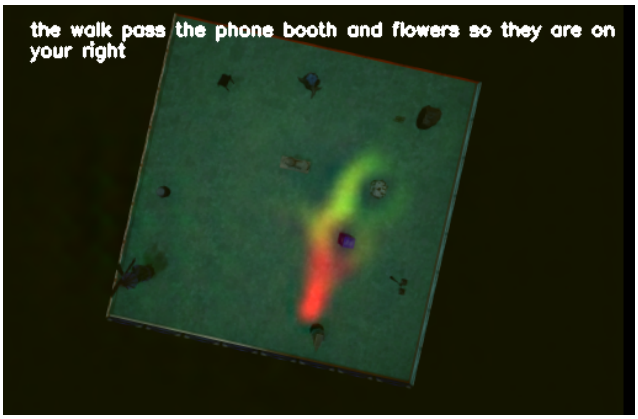



*figure 8*

From figure 7 & 8, we can see that the model can clearly understand which side should it go.

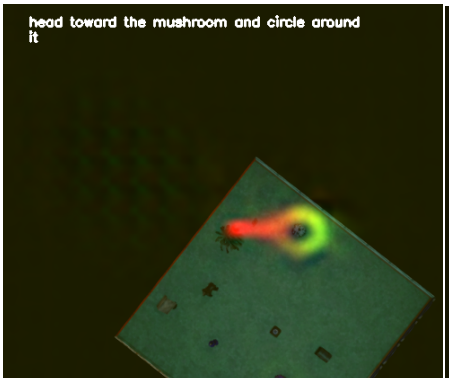



figure 9

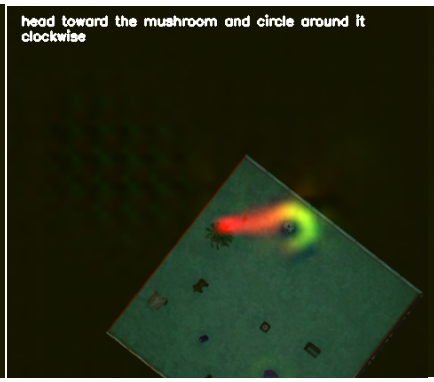



figure 10

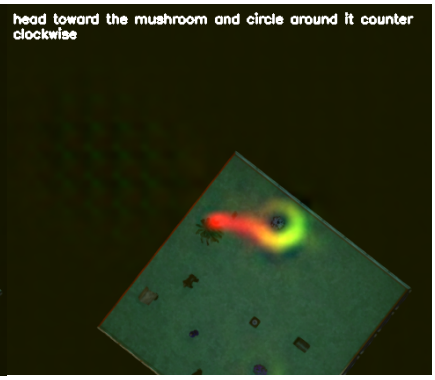



figure 11

From figure 9 & 10 & 11, we can see that the model does not understand circle around but also understand clockwise and counter clockwise.

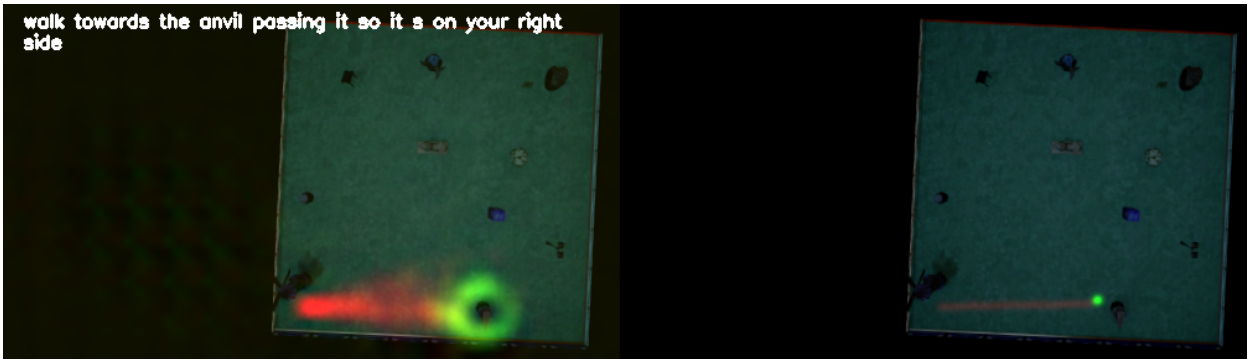
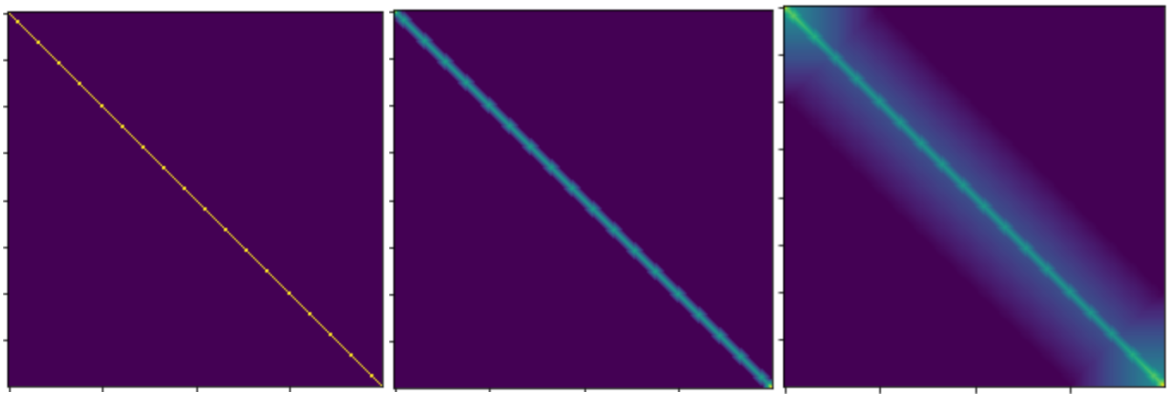



*Figure 12*

In figure 12, the instruction is little bit ambiguous to the model. “Passing it so it’s on your right side” is relatively difficult for the model to understand.

**Loss Function:**

By running those experiments, I found out that the current loss function is not very ideal since it cannot evaluate how close a prediction is. Then, I resampled the golden trajectory by blurring it with two different kernels so it might be able to recognize some close missed prediction.



Original Trajectory One Kernel Two Kernels

I used this two-kernel blurring to resample ground truth trajectories, and then, I used them to train a model with simple embedding. However, it did not improve the performance.

I also implemented binary cross entropy as its loss function but it is not working. Here is a prediction from a model trained with BCE loss: