Generating Natural Language Descriptions of Trajectories Using Long Short Term Memory Neural Networks

Rodolfo Corona and Rolando Fernandez

I. Problem Description

Given a point-cloud $p \in P$ and a manipulation trajectory $t \in T$, our goal is to output a free-form Natural Language (NL) description $l \in L$ that describes the trajectory t:

$$f : T \times P \mapsto L \tag{1}$$

II. Background on LSTMs

LSTMs are a variant of the Recurrent Neural Network (RNN) architecture which attempt to alleviate the vanishing gradient problem which traditional RNNs can suffer from.

LSTMs accomplish this through the use of *gates*, which, change the cell's state at each time step based on the current time step's input and the last time step's output. Intuitively, at each time step, an LSTM cell decides which information to forget and which new information to incorporate through the use of its gates. Formally, this computation takes the form of the following gate update equations:

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ c_t &= tanh(W_c x_t + U_c h_{t-1} + b_c) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \end{aligned}$$

And the cell state update and output equations:

$$\begin{aligned} C_t &= f_t C_{t-1} + i_t c_t \\ h_t &= o_t \cdot tanh(C_t) \end{aligned}$$

III. Methods

A. Robobarista Dataset

Sung et al. developed the Robobarista framework in order perform Deep Multimodal embedding of Point Clouds and NL Descriptions to Trajectories, as shown in Equation (2). Sung et al. aimed to create a system for manipulation planning which could generalize to different objects with similar parts to be manipulated [1].

$$f : P \times L \mapsto T \tag{2}$$

Using the Robobarista framework Sung et al. collected an extensive dataset of consisting of triplets of the form (Point Cloud, NL Descriptions, and Trajectory). The dataset consists of 116 objects with 249 parts along with 250 NL instructions, for which there was collected 1225 manipulation trajectories.

Each of the 116 objects is split into parts depending on the number of NL instructions that pertain to it and the original point cloud scene is segmented with respect to the part frames, being stored as a Comma Separated Value (CSV) file consisting of [x,y,z,r,g,b] values.

Furthermore, each of the objects have multiple manipulation trajectories for each part stored as a list of waypoints [1]. For our experiments we split the dataset into 5 folds of train and test data in order to perform 5 fold cross validation at the end of our experiments.

B. Baseline with Point Clouds

For our initial baseline we created a pipeline that utilizes the Point Clouds and NL descriptions in the Robobarista dataset to perform inference using K-Means clustering and K-Nearest Neighbors (Figure 1). We first train a K-Means model and a

KNN model using the training data for each of the 5 folds.

The Point Cloud key points are taken from the segmented object part point cloud CSV files. We create two vectors from the CSV files, one is a vector of all the Point Clouds in the training set where the Point Clouds themselves are vectors of key points and the other is a vector of all the key points in the training set.

Utilizing the K-Means algorithm in the Python SKLearn package, we input the vector of all the key points in the training set into the K-Means algorithm with a cluster count of 50 and get back trained K-Means model. With the K-Means model we extract a closest cluster prediction for the key points in each individual Point Cloud in the vector of all the Point Clouds in the training set. Then using the cluster predictions we compute a bag of features (BOF) vector for each Point Cloud based on the number of occurrences of each cluster.

The BOF vectors are then used as input with a neighbor count of 1 to the K-Nearest Neighbors classifier in the Python SKLearn package and in return we get a trained KNN model. We are then able to use this KNN model to determine the nearest neighbors of a given Point Cloud.

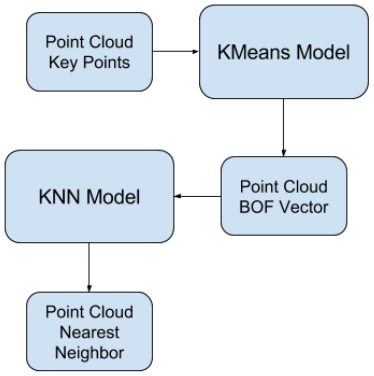

Fig. 1. Baseline with Point Clouds

C. LSTM Implementation

We currently implement an encoder-decoder LSTM neural network architecture. Specifically, our architecture uses one LSTM layer to encode the inputted sequence of trajectory waypoints t into an embedding h. This embedding is then fed into another LSTM layer which decodes it into a sequence of words y. We use the Keras[1] library with a TensorFlow[2] backend to implement our network architecture.

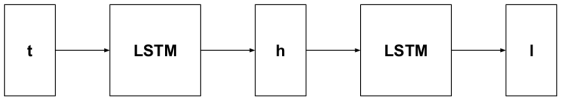

Fig. 2. Our current LSTM encoder-decoder architecture. x_t represents a trajectory waypoint sequence, h_t represents the embedding generated by the encoder, and y_t is the word sequence generated by the decoder.

D. Preprocessing

After analyzing the Robobarista dataset, it was found that the longest trajectory consisted of 18 waypoints. Therefore, all trajectories were fitted into tensors of 18 waypoints, with zero vectors used to pad trajectories with less than 18 waypoints.

Similarly, it was found that all phrases in the dataset consisted of less than 18 words. Each phrase was converted into a tensor consisting of 18 one-hot vectors of dimensionality $|V|$, where $|V|$ is the size of the training set vocabulary.

E. Training

Our current architecture consists of 50 hidden units in both the encoder and decoder. We have trained our network over 1000 epochs with a batch size of 80, a mean squared error loss function, and RMSProp optimizer with a learning rate of 0.001.

F. Testing

During testing, we feed a trajectory tensor to our network which then generates a sequence of one-hot vectors for phrases. Because every phrase in our dataset ends with a "." token, we generate phrases of 18 tokens and prune them up to the first appearance of the "." token.

IV. Results

A. Baseline with Point Clouds

For the baseline we conducted experiments using the data from each of the 5 folds, running a total of 5 trials. For each trial we created a

[1]https://keras.io/

[2]https://www.tensorflow.org/

gold reference consisting of the already known NL descriptions for each of the point clouds and a test reference consisting of the NL description returned by using the trained K-Means and KNN models.

Each point cloud in the fold test data is first inputed in the K-Means model to extract the BOF vector representation, then the BOF vector is inputed into the KNN model to retrieve the nearest neighbor of the point cloud, and lastly we look up the NL description for the given neighbor. With the completed the test and gold reference files we are then able to use the sentence comparison package METEOR to evaluate the accuracy of the Baseline with Point Clouds .

On average the baseline performed poorly as only reaching its highest performance in the first fold achieving a final Meteor score of 0.172. Figure 3, shows the performance statistics for each sentence compared in Fold 1. Figures 5 and 6, show the individual sentence comparisons results for two sentences from the Fold 1 test set. These results were expected as the Baseline only performs a simple lookup based on nearest neighbor inference and is not able to adapt properly.

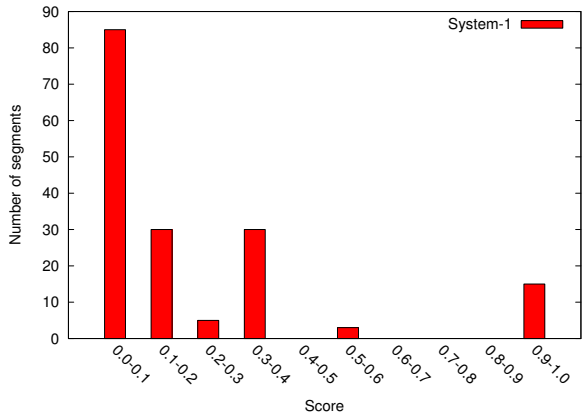

Fig. 3. Meteor Score for Fold 1 Test Data

B. LSTM Implementation

V. Difficulties

VI. Outlook

VII. Future Work

For the conclusion of the project, we propose to implement a two layer LSTM architecture as in [2]. Additionally, this architecture will incorporate point cloud data by concatenating the bag

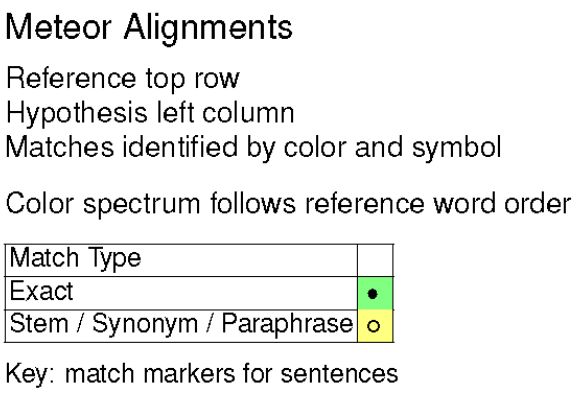

Fig. 4. Meteor Alignment Key

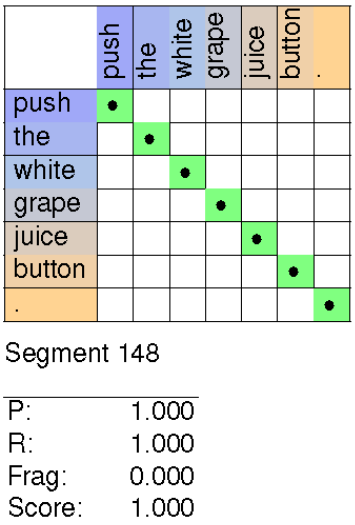

Fig. 5. Meteor Alignment for Fold 1 Sentence 148

of feature vectors produced from a point cloud to the trajectory embedding produced by the LSTM encoding layers. Finally, we will continue to experiment with different parameters for our LSTM architecture such as through different numbers of hidden units and training epochs.

VIII. Justification of Progress

References

[1] J. Sung, S. H. Jin, I. Lenz, and A. Saxena, "Robobarista: Learning to manipulate novel objects via deep multimodal embedding," *arXiv preprint arXiv:1601.02705*, 2016.

[2] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence - video to text," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

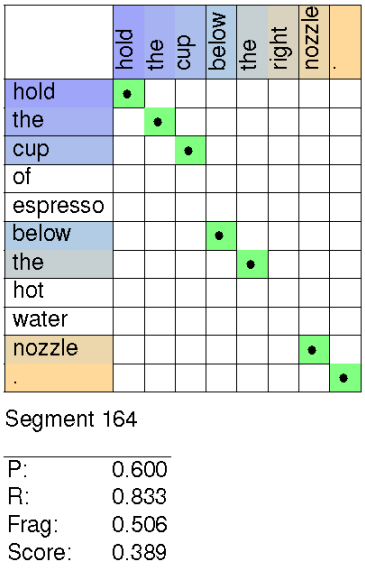

Fig. 6. Meteor Alignment for Fold 1 Sentence 164

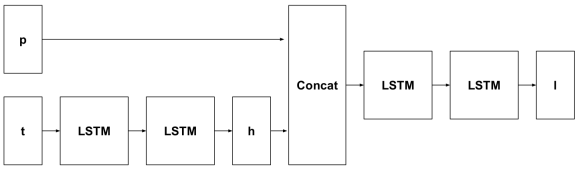

Fig. 7. The proposed final architecture for our LSTM model. The network will use a double layer LSTM encoder to generate trajectory embeddings. The two layer decoder will then be trained on the concatenation between the embedding and the point cloud bag of features vector to generate word sequences.