Abstract

Apparition, a crowdsourced system is used for rapid-prototyping the interface design. The key feature of canvas provides a Wizard-of-Oz experience[8] to the designer working with the software, neath the canvas the paid workers manage these micro-tasks with no overlapping and unique micro-task. In this paper, we applied Long Short-Term Memory Recurrent Neural Networks [1] to learn sequential data of animation. We discuss how we designed the LSTMs network so as to learn the animation interactive behavior.

1 Introduction

Long Short Term Memory (LSTM) are a special type of Recurrent Neural Networks (RNN) that was designed to model temporal sequences and their long range dependencies [7]. Machine Learning techniques have provided and utilized to tackle learning problem in the field of computer vision, speech recognition as well as natural language processing. We are looking at the application of specialized neural networks to learn the interactive behavior through animation data.

An effective design practice would be to include rapid prototyping in similar lines as it can be easily practiced in any software development process to better the quality of the software with active feedback. However, the most rapid interactive prototyping can be easily broken down into micro-tasks through the sketch. [8]. Apparition, a crowdsourced system has been proposed and evolved through different iteration with an inclusion of different features to support the ideology of rapid prototyping the design practice.

In this paper, we implement the methodology to fuse the learning of LSTM with the rapid prototyping system of Apparition. The user interface design behaviors could be captured in the form of animation so as to define the interactive behavior of the elements. The animation data form the sequential data which will require deep learning so as to evaluate and predict the behavior of the same. This work could be further extended to bringing automation
in the rapid-prototyping of design development. The key aspect of designing the LSTM so as to learn the micro-tasks and hence coordinate the interactive behavior of GUI elements.

2 Literature Study

2.1 Apparition: Crowdsourced User Interface that comes to Life as you sketch them

Walter S. Lasci et al introduce a crowdsourcing technique and tool for prototyping interactive systems in time it takes to describe the idea [8]. The system, Apparition provides a canvas that provides a Wizard-of-Oz experience to the designer during sketching the design. Apparition is the first crowdsourcing system to enable self-managing, real-time crowd coordination [8]. The authors describe the canvas in more details about the build and framework. The real-time coordination between workers is enabled using many different techniques like Naïve coordination fails, Write locks for effectively distributed coordination, support for concurrent editing and also protection against malicious workers.

2.2 Understanding LSTM Networks

In this essay [1] the author explains the intricate working of the LSTM network. The author initially sheds light on the working of Recurrent Neural Networks and how they resemble how a human brain works at thinking with persistence. Recurrent Neural Network is a powerful model that learns temporal patterns in sequential data [3]. RNNs have been applied successfully to solve problems like speech recognition, language modeling, translation, image captioning and so on. However, in practice, RNNs are unable to overcome the problem of long-term dependencies. Now the author introduces to the limelight of the entire article that is LSTMs Long Short-Term Memory Networks. LSTMs are a special kind of RNNs that are capable of learning long-term dependencies. The author next explains the core idea behind the working of LSTMs and then a step-by-step detail. The LSTMs basically consists of two important internal gates viz. "forget gate layer" and "input gate layer". The forget gate layer is responsible for deciding which aspect of information needs to be retained while the rest is forgotten. The input gate layer is responsible for deciding what new information needs to be retained and store. At the end of the article, the author explains different variants of LSTMs and how they differ with the classic architecture.

2.3 Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustics Modeling

In this paper, the authors have explained the complex correlations of speech which they successfully modelled using Long Short-Term Memory Recurrent Neural Networks. A complete and precisely detailed architecture of different
LSTMs are listed in the paper along with the relative performance with respect to temporal sequential data. LSTM is a specific RNN architecture that was designed to model temporal sequences and their long-range dependencies more accurately as compare to RNN [7]. The approach of using separate linear projection layer right after the LSTM layer allows reducing the number of parameters has been highlighted in the paper. This allowed the authors to increase the model memory while still being able to control the number of parameters.

2.4 Real-time Drawing Assistance through Crowdsourcing

In this paper, the authors analyze the crowdsourced drawing database and build a spatially varying model of artistic consensus at stroke level [2]. This artistic consensus is then plugged into the model along with a simple-stroke correction method to improve the strokes in real-time. The authors developed a simple gaming app, DrawAFriend on iPhone platform, which focuses on the face or face portraits. The primary aim of this paper is to determine from a large dataset if the system can identify many drawings by the same artist and many drawing of the same subject by different artists.

2.5 Answering Visual Questions with Conversational Crowd Assistants

In this paper, Walter S. Lasecki et al demonstrate the benefits of using multiple crowd workers instead of just one, with respect to improvements in latency and accuracy. The authors introduce Chorus: View, a crowdsourced system to assist users over a course of longer interactions [4]. This system is capable of answering sequential questions quickly and effectively. It uses continuous and reliable conversation with user about a video stream which indeed aids sequential answering of the question.

3 Methods

Apparition is meteor based application that provides a canvas over which the user can either sketch the interface prototypes or use speech input for the same. The backend is supported by Python modules and MongoDB database for manipulating and storing the data that we collect from the animations. Here, animations are recorded as a sequence of pictures in time. So it generates this animation effect of the object at hand moving through the space of canvas.

The canvas provides a smooth and seamless platform for interaction between users of the application. So to initiate the platform there are certain prerequisites that are needed on your system. Since the fact that it is a meteor application, where meteor is a JavaScript Library, we require NodeJS. The backend basically consists of Python and MongoDB database. So to run the application we issue the command ‘meteor’ from the terminal, which initiates the
server over the localhost. To access the application we need a browser window to log into a URL as below:

http://localhost:3000/?role=worker&workerId=some-worker&session=test6

Figure 1: Apparition UI (brick game example)

The figure 1 is a snapshot of the UI of the apparition application. The application contains animations over the right subsection. Over the right subsection there is the toolbox which can be used to sketch the UI elements and add animations to the application. To create an animation we first sketch the desired object. Next we want to make this object move and record those movements in the form of animation. To start recording an animation, we hit the key ‘a’ on the keyboard. Carefully enter the animation title as this is important for the lstm engine to understand which animation is a training example while which one is a test animation. Now, hit the ‘a’ key again to stop recording the animation. Once we are done creating the animations that successfully capture the intended behavior of UI elements we then move forward to generating features from these animations. At the backend we have a python module called FeatureSpace.py which can be used to generate three different documents. The FeatureSpace generates different files which have the following, a file to contain the training data, another for testing data and lastly a dictionary file that acts like an indexing for using these files to further manipulate data on and off the database. The next layer in the architecture is lstm_learn.py, this module is mainly responsible for learning the animation from the training and testing data that we earlier collected from a particular session. There are certain helper
modules such as lstm_utils.py and reader.py for efficient working of LSTM. Now this module generates an output file which contains the predictive model inscribed into animation data, which is then uploaded to the Apparition session over the canvas using FeaturesSpaceWithReversal.py module.

Here we are contemplating animation data as time sequence data with respect to capturing behavior of an element over a period of time. This can that be fed into the LSTM network to learn the behavior of different objects and hence generate a predictive model based on the data provided. The animation data provides a major role is increasing the efficiency of the model. One of the greater challenges for the scope of this project was to generate enough data for rapid test results and further improvements in the LSTM model application.

In this paper, we propose few data augmentation tools that can work around to generate more data points from the existing data recorded in terms of animation and the effect of such data augmentation tools over the effectiveness in learning done by LSTM networks. Below, are mentioned few tools which were designed and developed, the effective results are listed after their detailed implementation section.

Since the animations recorded over the canvas were majorly done by human interactions, there could be some errors involved, for e.g. an animation could have fewer data points if the speed of the moving object is fast. Also, the error can be introduced with devices such as mouse, track-pad, etc. To work around these errors and have valid data points to learn from we introduced new modules to process the animation data before the learning phase. Linear Interpolation, this module will take each animation for each object and depending on the length of the animation, i.e. the data points in the given animation log, linearly interpolate between the start and the end point of the sequential data.

\[
feature_i = i \times \frac{(feature_m - feature_1)}{m}
\]  

(1)

In the above equation, the features are all contents of feature space from the feature_dict. These features can be listed as follows:

1. x
2. y
3. width
4. height
5. rotationAngle
6. time

Working with the above mentioned features the next tool in the data augmentation suite was to develop a jitter function that will randomly jitter data points along the actual traced path of the animation. There are two approaches here that were successfully implemented. The first approach involved selecting
a random value of ‘j’ between (0,1) such that it will be deciding factor as to use
the weighted sum of two x or y co-ordinates to compute the new data point.

\[ feature_i = w_x \times x_i + (1 - w_x) \times x_m \] (2)

\[ feature_i = w_y \times y_i + (1 - w_y) \times y_m \] (3)

In the above equations the terms \( w_x \) and \( w_y \) are both jitter weights that are
again chosen by a random generator.

The second implementation of the jitter function was based on the use of
slope of a line created by two data points in the given data. To create a third
new point of data we compute slope of the given two points in the data sequence
and then by varying the y-intercept of the slope of the line we compute the new
y co-ordinate along with the x co-ordinate computed using the earlier function.

\[ m = \frac{y_2 - y_1}{x_2 - x_1} \] (4)

\[ c = y_1 - m \times x_1 \] (5)

\[ x_i = w_x \times x_1 + (1 - w_x) \times x_2 \] (6)

\[ y_i = m \times x_i + c \] (7)

Here, \( m \) is the slope and \( c \) is the y intercept of the original line.

Another set of tools from the data augmentation suite includes a constant
shift in x as well as y coordinates as per the user input during run-time. The
description of each tool from the data augmentation suite is listed below in the
Implementation section.

4 Implementation

The entire application constituted of these five stages. The important break-
downs of the application are as follows:

1. Record Animation
2. Features Extraction
3. Data Augmentation
4. LSTM learning
5. Features Reversal

The following figure gives the architectural perspective of the entire appli-
cation. The data augmentation phase is the new proposed phase that was
implemented. There are different modules that are designed and developed.
A new visualization module was implemented so as to test the new output at
a faster rate. The visualization module mimics the output over the x-y plane
representing the canvas from the Apparition application. Various tools from the
suite can be listed as follows:
In first phase we record animations over the canvas, that is provided by the Apparition platform. These animations can be recorded either over a single object or multiple objects. The animation data is stored in the MongoDB server with respect to each object and respective sessions. The next phase in the architecture is the feature extraction phase. The feature extraction phase connects a client to the MongoDB and generates three new files with different functionality, viz. sessionId_train.json, sessionId_test.json and sessionId_feature_dict.pkl. The train as well as test files contain the animation data in form of events for each object. This phase is implemented in the FeatureSpace.py module. The features of the animation are extracted and the framework around these features could
be accessed further by the use of sessionIdFeature_dict.pkl file. The next phase includes the collective effort of designing a suite of data augmentation tools that would help to generate more data in terms on new animation files that can be used to further test the effective learning of LSTM network. These tools form the basis of generating more effective data points in terms of manipulating the before listed features of each animation events and then computing new values in terms of a new file. Various tools from the suite can be listed as follows.

1. concat_json.py
2. linear_interpolation.py
3. jitter.py
4. jitter_slope.py
5. linear_constants.py

As described before the linear interpolation data, tries to smooth the animation, by interpolating the features of the animation in time with respect to x and y coordinates. So for each object and each animation take the starting and end point of the animation in the time sequence frame and linearly interpolate through the entire time sequential data. This will output data points which are relatively linear in terms of time and space.

Next up from the data augmentation suite is the tool of jittering the data points so as to understand the effective learning of the animation data by the LSTM model. There are two methods in which this jittering was implemented. First, was by randomly assigning the weights to the x and y coordinates to compute the new coordinates. The equations 2 and 3 can be useful to understand this implementation. Second, was computing the slope of the current two points at hand and then perturbing the y intercept value so as to gain a minute jitter in the new data point, which is perpendicular to the slope. Refer equation 4
to 7 for the computations of new x and y coordinates as well as slope and y intercept computations.

The next set of tools do the basic function of shifting the entire animation path by a constant x or y value for all the data points along the animation sequence. This new shift in the data points is taken as the input from the user during running the python module. Another tool from the proposed data augmentation suite is the tool to concatenate two or more json files. This helps to process a huge json file in terms of multiple animations in a single json file and reduces the processing time of the user by running the command of LSTM processing just once instead for all sets of json output files.

The last and the most effective tool from the tools generated is the visualization tool, which replicates the functionality of feature space reversal instead of actually reverse uploading the features back to canvas, instead it uses matplotlib to plot the data points in terms of x and y coordinates as a path of animation.

5 Results

The outputs of each of the data augmentation tools are listed below and discussed as follows:

Figure 4: Training Example

Figure 5: Linear Interpolation

Figure 6: Weighted Jitter

Figure 7: Jitter using Slope

In figure 4 the training example is captured in motion. The training data was then processed further by the linear interpolation module so as to refine the animation in terms of coordinate plane and time. The effective results
were captured in motion in figure 5. For complete visualization of the said session, we implemented the visualization module in the data augmentation suite. Each of the animation file can be run through this module to have a graph representation of the animation data in terms of path along the time sequence. The visualization plots of each of the above figure can also be found below at figure 8 and figure 9 respectively. In figure 8 the animation file is a concatenated version of training as well as testing examples from a session. In this session, the object, an ellipse moves from the left side of the canvas to the right side with different speeds in each of the paths from the three. The path with the shortest length, which is midway of the canvas, is the testing data. In figure 9 the animation seems rather smooth as compared to the training examples on the right.

The next session example was constructed using the ideology of the brick out game. Here the list of different training animations were the movement of the ball from wall to the floating user object and then back rebounding towards the brick arrangement. The figure 10 gives a motion view of the training example, in which the ball moves from the user object then rebounds from the brick arrangement and displaced towards wall and then back towards the user object. This animation data was then processed by the tool of jittering using weights and the following results (figure 11) were obtained.

The results of these session were then run through the visualization tool to
get the plot references of each data point over the canvas. The resultant plots can be found below in terms of figure 12 and 13. The visualization module could be ran further on different results that were accumulated during the implementation of the data augmentation suite. This facilitated the faster visualization tool which could work around the similar feature_dict file from each session so as to maintain the object information of each session.

Figure 12: Training Example Plot

Figure 13: Weighted Jitter Plot

6 Conclusion

The data augmentation suite consists of a generic framework which can be used to plug in different mathematical function over the animation data and compute the desired results. This design of generic framework helped me on faster development of the different data augmentation tools. The results obtained from data augmentation tools are somewhat exceptional and we are excited to test the resulting effects on the LSTM learning model. Which can then be used to further modulate the LSTM learning variables and weighted networks to propagate better learning results.

References

[1] Christopher Olah Understanding LSTM Networks, August 2015
Appendices

<table>
<thead>
<tr>
<th>Week#</th>
<th>Implementations</th>
<th>Report Updates</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>System setup. Download and install Anaconda python 2.7. Checkout necessary code from the git repo. Construct a working branch of the repo.</td>
<td>Start with reading reference materials</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>Background reading, begin coding, data analysis.</td>
<td>5 citations in bib file. One paragraph summary of at least one paper, describe desired data, list methods and software.</td>
<td>10</td>
</tr>
<tr>
<td>#</td>
<td>Task Description</td>
<td>Citations/Summaries</td>
<td>Hours</td>
</tr>
<tr>
<td>----</td>
<td>---------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>3</td>
<td>Further reading and coding. Construct test, training examples (one training, one test) and run machine learning loop (see README.md Apparition/meteor_version/lstm)</td>
<td>10 citations in bib file. One paragraph summary of at least three papers.</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>Further reading and coding.</td>
<td>One paragraph summary of at least six papers</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>Basic background reading finished.</td>
<td>Introduction includes clear statement of research questions addressed. Background section in decent shape.</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Create different test scenarios. Be prepared to demonstrate them and summarize in report. Create lots of screenshots.</td>
<td>Write abstract</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>Fix bug w/ reset. Why doesn’t it appear on some animations? (It happens when stop animation is pressed when worker is still moving element.) Discover new bugs.</td>
<td>Methods Section</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>Data Augmentation suite scripts. One takes all training examples from all sessions and the test from the given session (this may be as simple as file concatenation). Another script takes each example in a session and replaces with one where the start and end events are the same, but each intermediate event is a straight-line interpolation between start and end.</td>
<td>Methods section</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>Ubuntu System re-install, get project git repo branch to updated branches</td>
<td>Methods section</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>Dump meteor database currently located in Apparition.old. Send dump to Dr. Homan. Install dump in current repo.</td>
<td>Methods section update. Start with Implementation section</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>Merge git branch with Sang’s git branch. Update Meteor application. Remove JavaScript error</td>
<td>Update implementation section</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>Update jitter function with more generic implementation to use <code>feature_dict</code> only.</td>
<td>Updated implementation section</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>Research Photoshop and after effects method for visualization of animation in a picture. Strobing effect on animation object.</td>
<td>capture outputs</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>Visualization plot function developed.</td>
<td>update implementation section</td>
<td>12</td>
</tr>
<tr>
<td>15</td>
<td>Poster update. Capture output files and push to git branch.</td>
<td>Finalize report with all sections complete</td>
<td>12</td>
</tr>
</tbody>
</table>