Applying LSTM in GUIs to learn interactive behavior

Ravi Kumar Singh
Guided by Dr. Christopher M.Homan

ABSTRACT
Apparition is a crowdsourcing [11] system for rapid prototyping interface design. It has a shared canvas through which designers interact with the paid workers. The canvas provides a wizard-of-oz experience, where the designer seems to be working only with the software, while under the hood the canvas helps crowdsourced workers to co-ordinate tasks so that no two workers overlap over the same task and no tasks remain untouched.

In this paper, we have applied long short-term memory (LSTM) recurrent neural network to learn interactive behavior in animation.

1. INTRODUCTION
Machine learning techniques have been widely used in computer vision, images, signal and natural language processing. But it has not been widely explored for animation. Learning animation is an arduous work due to its nuances and various attributes.

We are using long short-term memory (LSTM) recurrent neural network to learn animations. Figure 2 shows the repeating module in LSTM. LSTM works in 4 interactive stages. During the first stage, LSTM cell removes all the information which are not needed for generating the output. During the second stage, LSTM saves all the information which required for the output using sigmoid and tanh function. Finally, in the third and fourth stages, LSTM decides what it is going to output using another sigmoid and tanh function.

We have defined animations as a sequence of images that, when viewed in rapid succession, create an illusion of continuous change over time. Unlike traditional RNNs, LSTM is best suited for the time series application where the gap between the relevant information and the place where the information needed is more. This is one of the many reasons why LSTM outperforms tradition RNNs, Hidden Markov Models [12] and other time series learning methods [13]. We have represented the state of each element in time as a point in a vector space with the two distinct sets (start and end) having various features like, shape, x coordinate, y coordinate, Id, transformation, x-center, y-center and radius etc.

We are storing the animation’s data into MongoDB [14] (our back-end database) and plugging it into LSTM model after parsing. Figure 1 shows the Apparition having the canvas and a pool of paid workers. Here, 1 and 5 represents the animation’s elements, 2 represents the menu, 3 represents the list of animations’ name and 4 represents the to-do list prepared by the paid workers. Here, the designer has requested for a user interface design which has divided into smaller tasks and the paid workers are working collaboratively and created a to-do list to complete the assigned tasks.

Figure 1: [1] Apparition having the canvas and a pool of paid workers

Figure 2: [3] Repeating module in LSTM having 4 interactive stages

2. RELATED WORKS
Lasecki et al. [6] introduced Apparition for rapid prototype user interface designs using crowdsourcing. It uses automatic gesture recognition algorithm to identify various strokes that a designer draws on the canvas (a version of SVG-Editor). If the algorithm can’t classify and convert
sketches into UI, it then requests the paid workers to begin work in real-time. The designer can provide instructions to the workers in the form of audio also. For providing coordination between workers, the system assigns different colors to each worker by using write-lock mechanism. Apparition maintains a to-do list which coordinates messages between workers. To overcome the problem of concurrent editing, the authors have introduced replace-at-layer mechanism. To avoid malicious workers, the system provides points to each worker and pick a worker based on the points.

Lasecki et al. (2015b) [8] extended the previous work by Lasecki et al. (2015a) [6] and introduced Chorus: a crowdsourcing based conversational assistant system. It allows the group of users to interact with each other and assist them using memory space. The authors also introduced a “Working Memory” section in the workers’ user interface which allows the workers to add important and meaningful conversation. This becomes useful while working with future workers in the real-time.

Limpaecher et al. [9] collected a large collection of drawings using a mobile game (DrawAFriend). After collecting the data, the authors analyzed it and built a model for artistic consensus in the real-time. Later, the authors implemented - correction vector field, an auto-correction algorithm which improves the strokes in the real-time.

Lasecki et al. (2011) [5] introduced a LEgion:SCRIBE, an end-to-end system that allows deaf or hard of hearing people to request for captions in real-time using crowdsourcing approach. The system allows multiple workers to provide simultaneous input that is then combined into a final answer. The system is based on real-time human computation and multiple sequence alignment.

Lasecki et al. (2011) explored crowd sourcing [7] and introduced Chorus: View: a system that assists the blind user and the paid workers in a long video conversation taken from the user’s mobile device to answer common visual questions more quickly and accurately. Users have given an option to either record a video or to do a live streaming. Later, the server chooses workers using Mechanical Turk and responds to user’s queries using a collective (group) chat interface. Once the workers done with the response, a screen reader on the user’s phone performs text to speech conversion. The worker can also provide feedback to the user on how to frame the required information. The user interface in the system has two main components: a) video streaming section b) chat window. Authors have used OpenTok for streaming video and an interface similar to Chorus for chat window.

Bernstein et al. (2011) [2] tried to resolve the issue of crowd latency in the crowd sourcing application. The authors have argued that within 10 secs the user loses his/her focus on the interaction dialogue. Hence, to provide response to the user’s queries within ten seconds, authors have introduced two techniques. The first technique, the retainer model, in which the workers get wage for the wait and then when the help is needed they are supposed to respond quickly. The second technique, rapid refinement, where they establish the agreement between crowdworkers and later they choose the worker based on work history. This approach produces result which has better quality as compared to the first approach. Additionally, the authors also tested these two approaches against two crowd sourcing application a) Adrenaline, a crowdpowered camera b) Puppeteer and A/B, which examine creative generation tasks.

3. METHODS

Apparition is an application that allows users to create interface prototypes using voice and sketch. It is built on top of Meteor[10] (NodeJS library) and has MongoDB as its backend. The application starts after executing - “sudo meteor” command and after running the below URL: http://www.cromalab.net:3000/?workerId=izzy&role=user&code=true&session=demoGame

The URL has below mentioned parameters:

a) role: role can either be a user, a worker or a manager. Based on the role of the person the interface gets changed.

b) workerId: A unique Id which is assigned to each worker.

c) code : code can either be true or false. It allows the user to input a code for any UI elements. It is enabled only for users.

d) session: A unique session generated for each animation.
e) search: This enables or disables the Icon search feature. If search is false, then Icon search feature is disable.

Figure 3 shows the worker interface in Apparition.

Figure 3: [3] Worker interface in Apparition

The numbers in the Figure 3 represents below:

a) 1 represents a canvas where the user and worker can draw their sketches.
b) 2 represents a toolbox to accelerate the process of creating UI sketches.
c) 3 represents the search function.
d) 4 represents a to-do list which indicates the tasks currently assigned to the workers but not yet completed.
e) 5 represents “in-progress” markers, which allows workers to know where they are working currently.
f) 6 represents “Minisketch canvas” to allows workers to create a stand alone sketches.
g) 7 represents “RIQ (revert image query)” which allows workers to get the images through Google image.
h) 8 represents various animation functions which user can perform like creating, updating, stopping and deleting the animation.

We have defined animation as a sequence of images that, when viewed in rapid succession, create an illusion of continuous change over time. We are creating animation using Apparition. After opening below URL: http://localhost:3000/?workerId=ravil76&session=erv76&role=worker select the shape of animation from the toolbox provided in the left side of the screen and draw the sketch on the canvas. After drawing the sketch on the canvas, press “a” and write the animation name appeared on the text box. Now, press “command” and select or drag the sketch over the canvas. To play animation, press “p.”

After creating the animation over the canvas, the ani-
ation's data gets stored in the MongoDB in the below-mentioned collections inside the "Meteor" database:

a) animationlog: This collection holds unique entry for every animation. The collection has:
   i) id : Unique id given to every animation.
   ii) name : Name given to the animation.
   iii) session : Session in which the animation was created.
   iv) workerId: Worker who created the animation.
   v) replay-count: How many times the animation was replayed.

Below is an example of the data stored in the animationlog collection:

```json
{
   "id" : "anmt−1475195727808",
   "name" : "AnimationName",
   "type" : "default",
   "session" : "erw5",
   "workerId" : "ravi5",
   "start" : 1475195727811,
   "active" : true,
   "speed" : 1,
   "replay_count" : 1,
   "visual" : {
      "id" : "R5fZ6DFM78ry69398"
   }
}
```

b) eventlog: This collection holds information about every events associated with the animation. The collection has:

a) Type : Type of the Animation.
b) Action : Animation's movement.
c) Session: Session under which animation has been created.
d) Animation: Unique Id assigned to each animation.
e) Worker: Worker Id under which the animation has been generated.
f) Timestamp: Time-stamp when the animation was drawn.
g) Transform: Various transformation which an animation go through, like, translation, rotation, reflection

h) X: X coordinate of the animation.
i) Y: Y coordinate of the animation.
j) Width: Width of the animation.
k) Height: Height of the animation.
l) Rotation angle: Angle with which the animation has been rotated.
m) Compared Diff: Difference between animation's start and end phase.

Below is an example of data stored in the eventlog collection:

```json
"type" : "change",
"action" : "move",
"session" : "erw5",
"animation" : "anmt−1475195727808",
"changeType" : "move",
"timestamp" : 1475195733798,
"elements" : {
   "start" : {
      "type" : "ellipse",
      "id" : "ravi5_1",
      "session" : "erw5",
      "lock" : "false",
      "code" : "{}
   },
   "kind" : "content",
   "layerIndex" : "1",
   "stroke" : "#000",
   "fill" : "#fff",
   "x" : 51,
   "y" : 78,
   "width" : 67,
   "height" : 57,
   "rotationAngle" : 0
}
```

After creating and storing the animation data into MongoDB, the data is then converted into feature space. We mapped the data stored in the MongoDB to the features vector using below-mentioned scheme:

i) The first 5 indexes in a feature vector represent the x coordinate, y coordinate, Height, Width, RotationAngle respectively.

ii) The next 10 indexes in a feature vector represent “Type” of the animation. Animation can be line, circle, triangle, square, ellipse, pentagon, hexagon, octagon, rect and rhombus. We defined a map to store these types and assigned a value of ’1’ in the feature vector’s index corresponding to the “Type” of the animation.

iii) The next 6 indexes in a feature vector represent “transform” of the animation. “Transform” can be “matrix”, “create”, “translate”, “rotate”, “transpose” and “end”. We defined a map to store these “Transform” and assigned a value of ‘1’ in the feature vector’s index corresponding to the “transform” of the animation.

iv) The next 20 indexes in a feature vector represent “id” of the animation. “id” is a unique identification which has given to every animation. We defined a map to store these “id” and assigned a value of ’1’ in the feature vector’s index corresponding to the “id” of the animation.

v) The last index represent the total number of features present in a feature vector.

We also normalized the data by dividing each elements of the feature vector with the canvas size of 580. We stored the animation data for two models: a) small model having 38 instances b) large model having 392 instances.

There are various long short-term memory (LSTM) recurrent neural network libraries available [4]. Few of them are:

a) RNNLIB : Implemented in C++.
b) Rnnscript : Implemented in Python.
c) OCRopus : Implemented in Python.
d) PyBrain : Implemented in Python.
e) JANNLab: Implemented in Java.
f) lstmlib: Implemented in C++.

We explored below-mentioned python libraries for our application:
a) Rnnscript: It is built on the top of RNNLIB.
   Pros: This can be trained easily and incrementally.
   Good for animations.
   Cons: Some features not bound with python.
b) OCropus:
   Pros: Fully implemented in Python.
   Open source.
   We can add our Python modules.
c) PyBrain:
   Pros: Fully Python implementation.
   Cons: Fairly slow for large dataset.

Later, we decided to go with Tensorflow [15] as it goes well with our environment.

LSTM is a special kind of recurrent neural network which is capable of avoiding long term dependencies problem. Like, recurrent neural network, LSTM also has a chain like structure with repeating modules but repeating modules in RNN has a very simple structure (like single tanh function) while the repeating modules in LSTM is complicated and having 4 interactive stages. LSTM is best suited for application where the gap between the information needed is more. Consider an application where we need to predict the last word in a sentence, “I grew up in France and I speak fluent French [3]”. Here, we can use LSTM to find the next word in the sentence. It works in 4 stages:

a) First, LSTM decides what information to remove from its cell state using the sigmoid layer called "forgot gate layer". It takes the previous output and the current input and produce a number between 0 and 1 for each number in the cell states [3]. So, in our example, whenever the LSTM finds the next subject, it will try to forget the gender of the previous subject.
b) Next, LSTM tries to find what all new information its trying to save. This works in 2 stages, first, using a sigmoid layer called "input gate layer" which decides what values be updating and second stage is create candidate for the new values using the tanh function [3].
c) Next, LSTM tries to update the old cell state with the next cell state [3].
d) And finally, LSTM decides what information will be generating. First, it applies a sigmoid function which decides what to generate and later it applies the tanh function through the multiplication of cell state and the output of the sigmoid function [3].

4. IMPLEMENTATION

We built the LSTM model in 4 phases. Figure 3 shows the flow chart of the entire application.
a) Data Gathering: This phase has already being covered in details in the section - Methods.
b) Creating and populating the feature space: This phase has already being covered in details in the section - Methods.
c) Building the LSTM model: After populating the feature vectors, we built the LSTM model. We started by training the LSTM model with the training data and later tested the LSTM model with the test data. The configuration setting while training the LSTM model are:
i) learning-rate = 1.0
   ii) init-scale = 0.1

Later, the model calculates the loss by using mean square method between the output generated by the model and the target vector.
d) After building the LSTM model and after generating the next step of the animation, we plugged the generated features vectors back into the Apparition. Since, we earlier normalized the data by dividing each element of the vector by 580, so, we multiplied each elements of the generated vector by 580 before plugging the vector into the Apparition.

We are first finding the animation Id based on the session Id of the animation and then finding the "Type", "transform" and "id" from the maps which we have created along with "x", "y", "width", "height", "rotationAngle" from the generated feature vector and then inserted those into the "eventlog" collections of the "Meteor" database of the Apparition.

How to run each stages:
a) Draw Animation: To draw animation in Apparition run "sudo meteor" command followed by opening below link:
   http://localhost:3000/?workerId=ravi76&session=erw76&role=worker

Now, draw the sketches on the canvas using the toolbox provided on the left hand side of the screen. Now, press “a” and write the animation name appeared on the text box. Now, to create animation, press "command" and select or drag the
sketch over the canvas. To play animation, press "p".
b) Feature Extraction: After creating the animation, the
animation data is converted into feature space. This can be
achieve by running the below command:
python FeatureSpace.py erw76 (here erw76 is the “session”
of the animation created in the stage a).
c) LSTM model: After populating the features space, we
are building the LSTM model to predict the next step of
the animation using training(Apparition_train_75.txt)
and testing data(Apparition_test_76.txt). This can be
achieved by running the below command:
python lstm_apparition6.py –data_path=/Users
/Downloads/simple-examples/data –model small. Here, /Users
/Downloads/simple-examples/data is the location where data
resides.
d) Plugging Animation: After building and running the
LSTM model, you can see the next step of the animation
(the last array in the console). Since we had normalized
the data, so we will need to multiple each elements of the
above array with 580 and then run the below command to
plug newly generated animation back into Apparition by
running:
python FeaturesSpaceWithReversal.py erw76 (here erw76 is the "session" of the animation created in the stage a).
P.S: Before running the above command make sure to edit
FeaturesSpaceWithReversal.py file and replace var "array"
with the array we got after multiplying with 580 and var
"previousArray" with last array of the given animation.

After running all the four stages, new entry will get cre-
ated in the MongoDB with next step of the animation.

5. RESULTS
We experimented with two models. We started with a
smaller model having 48 instances and later we experimented
with larger model having 392 instances. Figure 4 shows the
graph between Training perplexity vs number of Epochs for
the smaller model while Figure 5 shows the graph between
Train perplexity vs number of Epochs for the larger model.

6. CONCLUSION
We gathered the animation data from the Apparition and
converted the data into features vector which LSTM can un-
derstand. Later, We built the LSTM model and generated
the next step of the animation and plugged the generated
features vectors back into the Apparition. We built two
model for testing the application. A smaller model having
just 48 instances and a larger model having 392 instances.
Since, training perplexity is an indication of how well a
model predicts a sample. A low training perplexity indi-
cates the model is good at predicting the sample. From the
Figure 3, we can see that as the number of epochs is increas-
ing the model is getting better and better in predicting the
next step of the animation. Similarly, from the figure 4, we
can see LSTM model is getting better and better with large
number of data instance in predicting the next step of the
animation as compared to the smaller model.

References
[2] M. S. Bernstein, J. Brandt, R. C. Miller, and D. R.

Figure 5: Flow chart of the entire application

Figure 6: Flow chart of the entire application

Karger. Crowds in two seconds: Enabling realtime
crowd-powered interfaces. In Proceedings of the 24th
Annual ACM Symposium on User Interface Software
and Technology, UIST ’11, pages 33–42, New York, NY,
USA, 2011. ACM.
[3] Colah. Lstm — colah, the free encyclopedia, 2016. [On-
line; accessed 15-July-2016].
2016].
[5] W. Lasecki, C. Miller, A. Sadilek, A. Abumoussa,
D. Borrello, R. Kushalnagar, and J. Bigham. Real-time
captioning by groups of non-experts. In Proceedings of the
25th Annual ACM Symposium on User Interface Software
and Technology, UIST ’12, pages 23–34, New
York, NY, USA, 2012. ACM.
and M. S. Bernstein. Apparition: Crowdsourced user
interfaces that come to life as you sketch them. In
Proceedings of the 33rd Annual ACM Conference on
Human Factors in Computing Systems, CHI ’15, pages
1925–1934, New York, NY, USA, 2015. ACM.
[7] W. S. Lasecki, P. Thila, Y. Zhong, E. Brady, and
J. P. Bigham. Answering visual questions with con-
servational crowd assistants. In Proceedings of the 15th
International ACM SIGACCESS Conference on Com-
puters and Accessibility, ASSETS ’13, pages 18:1–18:8,
New York, NY, USA, 2013. ACM.
Allen, and J. P. Bigham. Chorus: A crowd-powered


