Low Level Entity State Sequence Mapping to High Level Behavior via a DeepLSTM Model

Joshua Haley, Victor Hung PhD, Robert Bridgman, Norb Timpko, Robert Wray PhD
Soar Technology Inc.
Ann Arbor, Michigan 48105, USA
{first.last}@soartech.com

Abstract - The authors present a Deep Long Term Short Term Memory (LSTM) model and input representation that maps sequences of low level entity state to a specific high-level behavior that produced the data track. This multipart model is capable of spatial reasoning via a Convolutional Neural Network (CNN) preprocessing the 3D input sequence vectors into higher level features and temporal reasoning is provided by a Deep LSTM network which is fed into a fully connected Feed Forward Neural Network. We present initial results on a test dataset of synthetically generated data indicating ability to determine behavior type from examples.

Keywords: Deep Learning, Machine Learning, RNN, LSTM

1 Introduction

Simulation training technology, such as military simulations, use computational models of human behavior to populate the simulation environment [1]. These models vary in the technology and methods used to implement them [1-3], their overall fidelity and robustness to human behavior [4-6], and their ability to adapt to the ways in which different users of the system employ these models [1, 5].

Regardless of what technology is used or what fidelity is intended, human behavior models deployed in actual applications are subject to two competing requirements that are often at odds with one another. First, models typically undergo a validation process that ensures that the models that are used to train human decision makers generate a rich and valid range of responses appropriate for the situation. This validation process is itself a significant challenge [6, 7]. However, validation also tends to be in tension with the operational reality that the real-world behavior of humans and systems in these environments is constantly changing. The models that generate behaviors in these systems must then adapt to new operational requirements as they emerge.

Historically, there have been two differing paths to incorporating new behavior requirements in these systems. The first is to refactor the existing behavior models to include the new behavior. This approach can take considerable time but supports on-going validation of general behaviors. The other approach is to transform sensor observations of the new behavior into a “script” that can reproduce the exact behavior in a replicated setting. This approach has the advantage of taking little time and labor resources to implement, but is very brittle for use in any circumstances other than the exact setting in which the behavior was observed.

The work we describe attempts to find a middle ground between these competing approaches. We employ observations of real-world scenarios to re-create simulation-based scenarios. However, rather than simply developing a “script” that replays the scenario, we interpret the events that occurred in the scenario and then map those events to existing (and validated) models available in the simulation system. We use CNNs to develop an understanding of what individual models can do and how to parameterize them, and then use that understanding to perform the mapping. In many cases, multiple models may need to be used at different points in the scenario to provide the type of behavior observed at that point in time.

This approach offers several advantages. First, it reduces the need for specialized expertise to create complex, realistic scenarios and the time required to develop and test such scenarios. It preserves the validation of existing models, using them as the foundation for generating scenario experiences not readily available with the default settings. The resulting scenarios are also interactive, allowing users to drive experience and exploration (e.g., “what if”) along pathways not directly observed in the real-world, while also not requiring lots of manual control or direction to achieve those results.

In the remainder of the paper, we introduce specific example maneuvers that are representative of the kinds of maneuvers and tactics one might encounter in operational domains. We then describe how we use a systematic approach to generating variation across encoded behaviors to learn a model of the behaviors that can then be applied to recognition of specific maneuvers and the patterns governing their behavior. This result is a significant, initial step toward a capability that will realize the benefits outlined above.
2 Domain Description

In order to illustrate the computational problem and our approach to it, we summarize a simplified air-to-air tactical combat domain. In real world settings, the tactics that would be explored would be more sophisticated, but the domain we describe here provides an illustration of the computational challenges that can be understood without deep knowledge of the domain and that can be implemented into a dataflow as illustrated in figure 2.

To explore this domain, we use a distributed simulation environment that executes scenarios consisting of initial positions of entities, and behavior programs associated with the entities. The behavior program representation is roughly comparable to those of hierarchical task networks [8]. These behavior programs allow simulation entities to react to changing context and to perform actions and maneuvers in response.

2.1 Illustrative Maneuvers

As an example, we introduce three geospatial maneuvers to accomplish an intercept task where one aircraft must intercept and then begin pursuit of the opposing force aircraft. They are defined below:

- Maneuver A: The entity of interest flies past the target and then banks. In this maneuver, the relative bearing of the entity of interest to the target is configurable.
- Maneuver B: The entity of interest keeps a constant bearing to the target and then falls behind after reaching the target. The initial constant bearing is configurable.
- Maneuver C: The entity of interest holds a relative altitude to the target and then changes altitude at a specified distance from the target. The relative altitude is configurable.
In addition to the configuration parameters described per maneuver above, the distance at which the maneuver starts can vary for each maneuver as well. A visualization of maneuver A is illustrated in figure 1.

2.2 Sample Generation

To better model the sequence to behavior mapping, we must fully explore the search space exposed by the variables associated with the behavior graph. We use a software utility we call the “diversifier” that takes in a base scenario and then does a combinatorial exploration across all of the variables associated with a behavior. Continuous variables are discretized by setting a step size as well as minimum and maximum bounds. In addition to the behavior configuration variables, initial position and heading were varied within a set threshold. This led to a dataset of 2236 Maneuver C, 1322 Maneuver A, and 1199 Maneuver B examples. These scenarios were run in the distributed simulation environment, and positional information in 3D cartesian space for each scenario was collected per frame. We call each individual scenario run a sequence, and each sequence contains a single geospatial maneuver as described above. The variation of configurations and initial positions allows our model to be trained for a general representation of the maneuver space, so when an unknown track with unknown configuration variables is presented, it can be better classified.

3 Model

Within our aviation domain, there are aspects of spatial and temporal data. To exploit the spatial components of the sequences, we use CNNs that have been shown to be useful for such tasks [9]. The temporal features of the data will be recognized using Recurrent Neural Networks (RNN), specifically a deep LSTM model. The models are implemented in Keras [10], a Python machine learning library built on top of Google’s Tensorflow library [11].

3.1 Input Representation

In the Geospatial maneuver domain, the relative angles and relative distances within 3D space are the largest differentiators between different tactics. A Naïve approach would be to simply precalculate those relations and use that as input into the model. However, given the domain of interest, the number of entities per scenario are not known apriori and thus the input representation would be dependent upon the scenario. Instead we chose to use a discretized 3D world representation presented as a sequence of time slices that offers the following benefits: 1) It is invariant to the number of entities in the scenario as the input. 2). It preserves the spatial relationships between entities, which allows for CNN preprocessing to be used to extract features of those relations.

To generate a sequence representing a single scenario all positional inputs are for all entities in the scenario are extracted from game logs and down sampled to a lower time resolution of five second increments. All positions are then set to be relative to the entity being observed, which we call the target. The size of the 3D matrix is preset to a fixed size of 28x28x28 where the resolution of each block is determined by the maximum position offset between any entity and the target. Therefore, the resolution of each block may be different between sequences but will be consistent within any single sequence. The target is then assigned to the center position and represented by value of ‘1’. For all other entities, their relative coordinate in the discrete world is calculated and allied entities are represented with a value of ‘1’ and enemies are represented by a value of ‘-1’. This is done for every time slice to form a 4D tensor as a single training instance for the model. The input data is already bounded by the range [-1,1], so no additional normalization is used. The labels have a combination of binary and real valued features, a feature wise normalization of the labels is performed before training.

3.2 Output Representation

The output layer of the model is an 11 valued vector corresponding to the selected maneuver in a one-hot encoding as well as the configuration variables associated with the behavior. This output is put through an inverse normalizer to compare to the expected outputs.

3.3 CNN Preprocessing

Convolutional Neural Networks, also known as shift invariant or space invariant artificial neural networks are commonly applied to analyzing visual imagery [9, 12, 13]. They have been used extensively for image recognition and classification and proved to be very accurate and resistant to confusion due to changes of orientation or positioning within the input [9].

We use a 3D CNN to extract features from each 3-dimensional time slice as described in [14] to then be later classified by the recurrent and fully connect components. We use a standard convolutional approach [9, 15], but use Average rather than Max pooling; otherwise opposing forces represented by negative values would be potentially dropped from the representation [16]. After 6 stages of Convolution and pooling as described in Figure 3, the final output is flattened into a single 2048 valued vector. These convolution and pooling transformations are applied for every time slice in the sequence and, thus, the output is a sequence of higher level features extracted from the spatial data.

3.4 LSTM Sequence Classifier

LSTMs as proposed in [17] are a type of recurrent neural network developed to deal with vanishing gradient issues present in other RNNs. They are relatively insensitive to gap length within sequences and are useful for detecting...
relations within sequence data. They have been shown to be useful for text and speech recognition [18, 19]. We hypothesize that LSTM’s ability to encode sequence data will allow the model to classify the differences between the behaviors given the sequence of positional states. A Deep LSTM model stacks multiple LSTM layers so that the output of a layer is a function of all inputs seen thus far, and successive layers are functions of those outputs until the current iteration in the sequence. This deep LSTM stacking has been shown to pick out more complex temporal relations than single LSTM models with greater accuracy [20].

Our particular LSTM architecture as illustrated in figure 4 will use multiple layers that allow us to calculate relations on the sequence across multiple time horizons. In particular, our LSTM architecture has two layers which employ a many-to-many semantic and return full sequences of outputs for the input sequences. This is then fed into two time-distributed fully connected networks and is applied independently to all timesteps in the sequence. This fully connected layer allows the model to try and identify functional relations between features. The final sequences output by these fully connected layers is input to a LSTM layer employing a many-to-one semantic which calculates a single output based the entire sequence and fed through a fully connected layer to generate the output as previously described.
The dimensions of outputs per layer for the model are given in Table 1. Please note that `None` indicates a non-fixed size and thus that layer will operate on any number of inputs in a tensor that match the rest of the shape parameter. For example, a shape of `(None, 1)` would be a non-fixed single valued vector that can be any length.

3.5 Training

The CNN feature extraction model is a set sequence of data transformation with a set weight initialization, which does not require additional training. We use the default Keras weight initialization kernel for these purposes. The DeepLSTM model is trained using a mean squared error loss function and the Adam optimization function [21]. We trained the DeepLSTM model using 300 batched epochs the first 50 of which are illustrated in figure 5 using a validation set of thirty randomly selected instances. The validation or test loss was lower than the training loss, which indicates a lack of over fitting and potential for improved performance with more training.

4 Results

Recall that the model was simultaneously classifying both the type of maneuver used to produce the data track as well as the configuration variables used for the specific variant. We look at the results of both outputs independently.

4.1 Maneuver Identification

Looking at all instances the model was able to correctly classify which maneuver was employed to create the positional data track 91.2% of the time. In particular, Maneuver C, which was very separable given that it was the only maneuver that caused change in altitude, was correctly predicted over 99% of the time. Maneuvers A and B had about a third of their samples confused with each other when predicted with the model. Post-experiment analysis showed that Maneuvers A and B for large offset angles resulted in identical positional sequences. The results are skewed toward a higher accuracy due to an imbalance in the count of each maneuver instance used in the training toward the more accurately identified maneuver C. The accuracy values of the validation data set follow that of the entire training set indicating a lack of over fitting.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Model Architecture</th>
<th>Output Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv1</td>
<td>3D Convolution</td>
<td>(None, 64, 28, 28)</td>
</tr>
<tr>
<td>Pool1</td>
<td>3D Average Pooling</td>
<td>(None, 64, 14, 14)</td>
</tr>
<tr>
<td>Conv2</td>
<td>3D Convolution</td>
<td>(None, 64, 14, 14, 128)</td>
</tr>
<tr>
<td>Pool2</td>
<td>3D Average Pooling</td>
<td>(None, 32, 7, 7, 128)</td>
</tr>
<tr>
<td>Conv3a</td>
<td>3D Convolution</td>
<td>(None, 32, 7, 7, 256)</td>
</tr>
<tr>
<td>Conv3b</td>
<td>3D Convolution</td>
<td>(None, 32, 7, 7, 256)</td>
</tr>
<tr>
<td>Pool3</td>
<td>3D Average Pooling</td>
<td>(None, 16, 3, 3, 256)</td>
</tr>
<tr>
<td>Conv4a</td>
<td>3D Convolution</td>
<td>(None, 16, 3, 3, 512)</td>
</tr>
<tr>
<td>Conv4b</td>
<td>3D Convolution</td>
<td>(None, 16, 3, 3, 512)</td>
</tr>
<tr>
<td>Pool4</td>
<td>3D Average Pooling</td>
<td>(None, 8, 1, 1, 512)</td>
</tr>
<tr>
<td>Conv5a</td>
<td>3D Convolution</td>
<td>(None, 8, 1, 1, 512)</td>
</tr>
<tr>
<td>Conv5b</td>
<td>3D Convolution</td>
<td>(None, 8, 1, 1, 512)</td>
</tr>
<tr>
<td>Padding</td>
<td>Zero Padding Layer</td>
<td>(None, 8, 3, 3, 512)</td>
</tr>
<tr>
<td>Pool5</td>
<td>3D Average Pooling</td>
<td>(None, 4, 1, 1, 512)</td>
</tr>
<tr>
<td>Flatten</td>
<td>Flatten</td>
<td>(None, 2048)</td>
</tr>
<tr>
<td>LSTM1</td>
<td>LSTM (Many to Many)</td>
<td>(None, 60, 60)</td>
</tr>
<tr>
<td>LSTM2</td>
<td>LSTM (Many to Many)</td>
<td>(None, 60, 60)</td>
</tr>
<tr>
<td>Time Distributed 1</td>
<td>Time Distributed Dense</td>
<td>(None, 60, 512)</td>
</tr>
<tr>
<td>Time Distributed 2</td>
<td>Time Distributed Dense</td>
<td>(None, 60, 128)</td>
</tr>
<tr>
<td>LSTM3</td>
<td>LSTM (Many to One)</td>
<td>(None, 60)</td>
</tr>
<tr>
<td>Dense3</td>
<td>Dense (Fully Connected)</td>
<td>(None, 11)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maneuver Type</th>
<th>Maneuver Identification Accuracy (Percentage)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
</tr>
<tr>
<td>A</td>
<td>72%</td>
</tr>
<tr>
<td>B</td>
<td>69.2%</td>
</tr>
<tr>
<td>C</td>
<td>99.6%</td>
</tr>
<tr>
<td>All</td>
<td>91.2%</td>
</tr>
</tbody>
</table>

TABLE 2 Maneuver Classification Accuracy
4.2 Maneuver Configuration Variables

The model did a poor job of predicting the configuration variables associated with the maneuver. In particular we have identified two issues that caused suboptimal performance.

First, we noticed a binning of values where several close configuration variables (for instance, 2500, 2750, 3250m for altitude) would all be mapped to the same value. This was due to the resolution of the discrete block in the input representation. The model was not given information discriminating the differences in input. One possible mitigation strategy is to increase the number of blocks in the input representation thus increasing the resolution. Another possible mitigation strategy is to set a fixed resolution and only represent entities “close enough” to the entity to affect the entities reactions. We will investigate further in follow up research.

Secondly, we noticed a biasing effect of default values. With our fixed label representation, we included a default value for configuration variables not used for the selected maneuver. These essentially were “don’t care” values as far as the simulation environment was concerned but caused the model to predict toward these defaults. In future work we plan to use different models that are essentially a collection of binary classifiers rather than a single categorical classifier allowing each model to only have variables directly related to the specific maneuver. Another potential mitigation being investigated in future research is the use of randomly selected default values using Gaussian selection to prevent a bias from occurring in the single categorical regression model.

5 Acknowledgment

The authors thank their many teammates for conceiving and implementing many of the concepts reported here. We also thank Dr. Heather Priest and Mr. Samuel Parmenter for their contributions to the problem definition and solution context that informed the approach. The opinions expressed here are not necessarily those of the Department of Defense or the sponsor of this effort: the Naval Air Warfare Center Training Systems Division (NAWCTSD). This work was funded under contracts N68335-17-C-0574.

6 References