0.1 Introduction

0.1.1 TORCS, Direct Perception, and Recurrency

As discussed in Part 1, **?** showed the feasibility of the direct perception paradigm as applied to a TORCS driving dataset. We then showed that implementing a recurrency within the data model offered predictive benefits, through a series of rigorous statistical analyses. The next step, then, is to build a recurrent model using state-of-the-art methods, and evaluate it on the TORCS dataset. To do this, we look to RNNS and LSTMs.

0.1.2 Recurrent Neural Networks

While **?** and many others have shown that convolutional neural networks (CNNs or convnets) have provided massive performance gains in image recognition, recurrent neural networks (RNNs) have proved very useful for temporal and sequential data. The idea behind RNNs simply relies on activation and excitation of neuron layers without external inputs – in other words, connections between units can form directed cycles, allowing for the network to exhibit dynamic and temporal behavioral qualities. Most practical applications of RNNs use a special network structure called Long Short-Term memory, or LSTM networks.

Long Short-Term Memory

In 1997, Hochreiter and Schmidhuber provided ground-breaking work on a new model of recurrent neural networks. Previous work by Hochreiter **?** showed that in training previous recurrent models, error signals "flowing backwards in time" either blew up or vanished. In the first case, weights would oscillate without convergence, and in the second, the network would be unable to learn to bridge the long time lags effectively **?**. They propose LSTM as a recurrent network architecture designed to overcome error-flow problems, and bridge long time intervals despite noise issues. After introducing the architecture, they conduct a series of experiments that show the consistent high performance of LSTM models in long minimal time lag data. For the experiments, they select arbitrary LSTM architecture (single memory cell, two memory cells, etc.) and compare to baseline RNNs, such as RTRL (Real-Time Recurrent Learning) and BPTT (Back-Propagation Through Time). Empirical evaluation in these experiments demonstrated the potential of LSTM models in machine learning efforts.

0.1.3 LSTM in Caffe

? proposed a Long-Term Recurrent Convolutional Network (LRCN) that combines the strength of CNNs in visual recognition problems with stacked LSTM modules for image description tasks. They demonstrated the viability of this approach through empirical evaluation, and were generous enough to provide a pull request to the official Caffe repository adding support for RNNs and LSTMs. **?** then adapted their LSTM and RNN layer implementations to the TORCS data, creating multiple model structures implementing a recurrency. For our nonparametric extension, we use one of said pipelines, included below from the work of **?**.

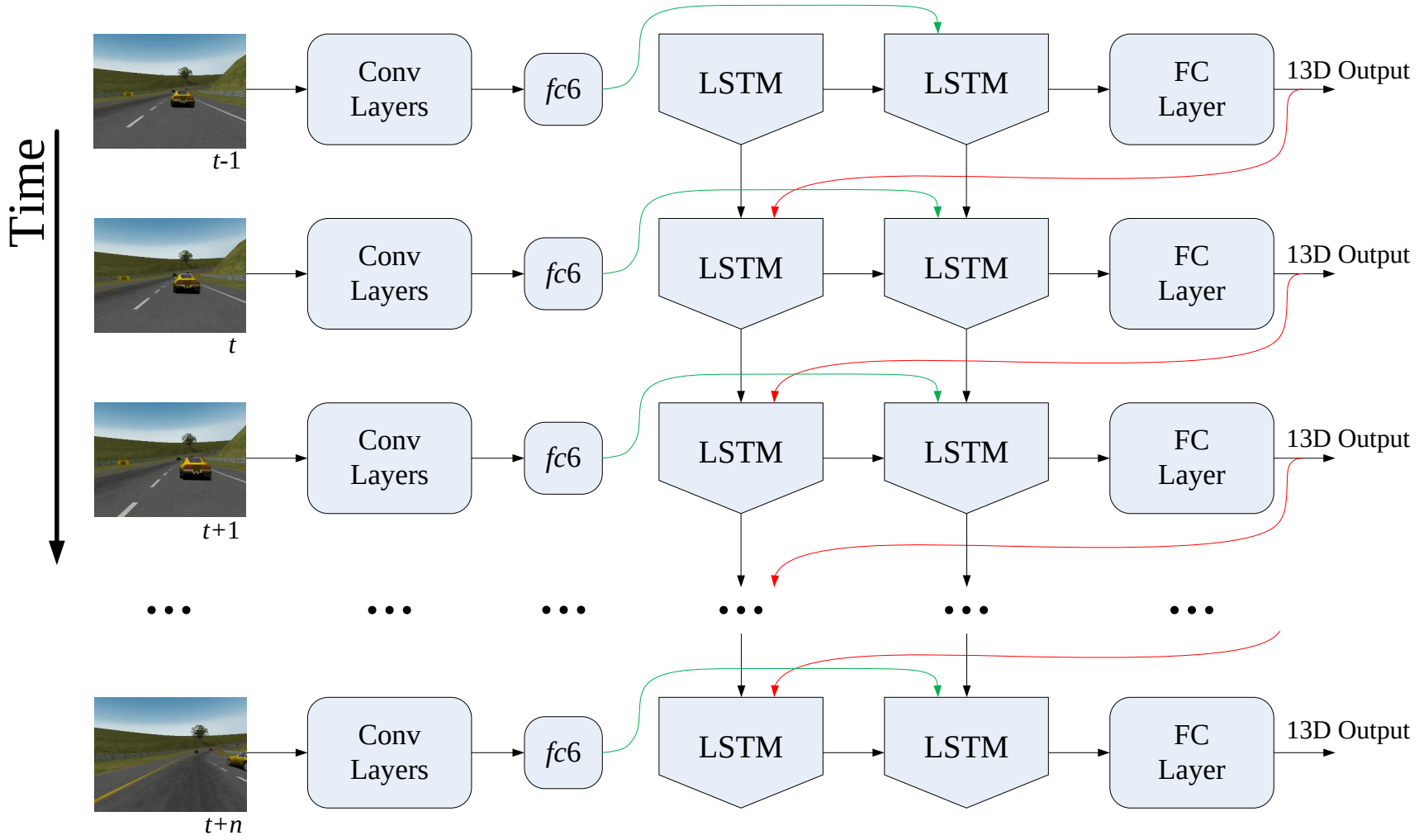

Figure 1: Chen's TORCS 2-Layer Skipped LSTM Structure

0.2 Nonparametric ReLU Implementation

In **?**'s LSTM structure, there are rectified linear unit (ReLU) layers for activation (not depicted in above figure), as is standard within deep networks. Our contribution is to implement nonparametric bases extensions for these ReLU activations, and examine the resulting effect on predictive performance. As proposed by Prof. Liu's SML group, the addition of nonparametric basis expansions into deep neural networks increase the model exploration space. Past work has shown that this enables the network to better capture the deep structures within the data, resulting in a more predictively powerful model. For the TORCS data and **?**'s LSTM structure, we create nonparametric modifications of the ReLU layer by implementing additive factors for sigmoid and Fourier bases. The forward passes for ReLU, nonparametric sigmoid ReLU, and nonparametric Fourier ReLU are as follows:

$$
\begin{aligned}
f(\mathbf{x}) &= \max(0, \mathbf{x}) \\
f_s(\mathbf{x}) &= \max(0, \mathbf{x}) + \beta_s \frac{1}{1 + e^{-\mathbf{x}}} \\
f_F(\mathbf{x}) &= \max(0, \mathbf{x}) + \beta_F^{(1)} \sin(\mathbf{x}) + \beta_F^{(2)} \cos(\mathbf{x})
\end{aligned}
$$

To implement our new layers in Caffe, we need to specify the backward pass computation as well. This consists of implementing the partial derivatives with respect to the input and the weight parameters β, in order to update the weights and propagate the error back through the network. The partials for backpropagation are as follows:

$$
\begin{aligned}
\frac{\partial f_s}{\partial \mathbf{x}} &= \mathbb{1}_{\mathbf{x}>0} + \beta_s \frac{1}{1 + e^{-\mathbf{x}}} \left(1 - \frac{1}{1 + e^{-\mathbf{x}}}\right) \\
\frac{\partial f_s}{\partial \beta_s} &= \frac{1}{1 + e^{-\mathbf{x}}} \\
\frac{\partial f_F}{\partial \mathbf{x}} &= \mathbb{1}_{\mathbf{x}>0} + \beta_F^{(1)} \cos(\mathbf{x}) - \beta_F^{(2)} \sin(\mathbf{x}) \\
\frac{\partial f_s}{\partial \beta_F^{(1)}} &= \sin(\mathbf{x}) \\
\frac{\partial f_s}{\partial \beta_F^{(2)}} &= \cos(\mathbf{x})
\end{aligned}
$$

0.3 Results

0.4 Discussion