Sproj – TODO

Bard College

Hayden Sartoris

April 2018

Contents

1 Model

The model trained and tested here represents ... stuff

1.1 Data

Insofar as we treat ANNs as providing arbitrary function approximation, training a network requires input data representing the known data about the system we wish to model, as well as output data we wish the network to produce from the inputs. More generally, input data usually entails information that is easy to acquire about the process being modeled, while output data, or labels, correspond to a dataset that is difficult to acquire generally. Of course, this means that the first step in training a neural network is to assemble a sufficiently large set of inputs and outputs in order to fully, or at least approximately, characterize the problem at hand.

In our case, we wish to map from (relatively) easily available data about biological networks, individual neuron spike times, to network structure. While such data exist, generating our own allows us to better analyze the results of the algorithm.

1.1.1 Generation

In order to demonstrate the validity of our algorithm for graph convolution, we opt for a simplified form of the kind of data that would be used in a real-world setting. To this end, we create adjacency matrices representing simple, small-n toy networks.

FIG: 3 neuron model & associated adjacency matrix

Binary values are used throughout these toy networks: either a connection exists or it doesn't; either a 'neuron' is spiking or it isn't. To produce spiking data, we create an n-vector $\mathbb{S}$ representing the current state of the toy network, with random neurons already spiking based on a chosen spike rate. From here, the process is as follows, where $\mathbb{M}$ is the adjacency matrix:

$$\underset{n\times n}{\mathbb{M}} \times \underset{n\times 1}{\mathbb{S}^t} = \underset{n\times 1}{\mathbb{S}^{t+1}}$$

Additonally, $\mathbb{S}^{t+1}$ may have one or more neurons spike randomly, as determined by the spike rate of the simulation.[1] All values are clipped to the range $[0, 1]$, to avoid double spiking.

At each step, $\mathbb{S}$ is appended to an output matrix, which is saved after simulation is complete. For t simulation steps, the completed output has shape $(n \times t)$.

APPENDIX BIT: have a section on how spike rates were determined for each network, as well as an example of producing data for the 3-neuron case.

In most ANN implementations, feeding various data with the same label attached to it results in the network learning to ignore the input data and always spit out the desired label, rendering it useless. However, due to the unique structure of our model, this sort of overfitting is impossible (SEE SOME ARCHITECTURE SECTION). Therefore, we must merely construct a suitably representative generator network, meaning that it contains all of the inter-neuron relationships we expect to see in the data we ultimately feed in to test.

1.1.2 Restructuring

The model accepts data in the form of a spike-time raster plot of dimensions $(n \times t)$, where n is the number of neurons and t is the number of timesteps being considered. The axes are reversed in comparison to the data created by the generator, and thus in the process of loading in the spike trains we transpose the matrices to the expected dimensionality. Additionally, it is not always necessary to use the full number of steps generated, depending on the size of the generator network in question, as well as its spike rate. In such a scenario, we truncate the time dimension appropriately.

EXAMPLE HERE

[1]SEE APPENDIX

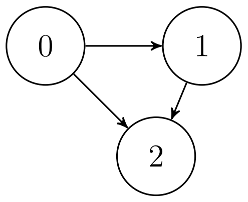
2 Architecture

Our model can be broadly described as convolutional, as it is entirely n-independent, relying instead on the application of matrices at each layer to all possible pairs within n. Beyond this, the model also incorporates data from adjacent nodes in its evaluation of a given potential edge, meaning that another important aspect of convolution, locality, is included.

3 Results

3.1 3-neuron generator

We first consider a generator network consisting of three nodes connected as in FIGURE HERE (should contain visual structure and adjacency matrix). All weights are binary, and a spike rate of .25 was used.[1]



	0	1	2
0	0	0	0
1	1	0	0
2	1	1	0

Figure 3.1: Network structure and adjacency matrix of the generator.

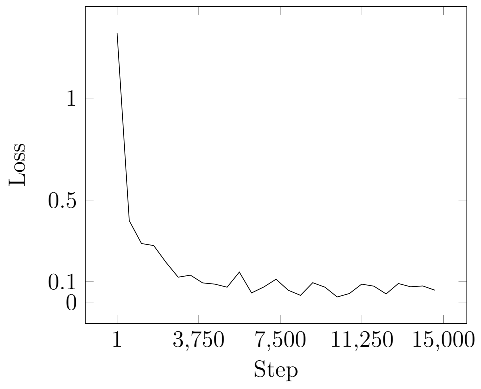
While reconstructing a graph comprising only three nodes is not much of a feat, this simplified case allows us to demonstrate that our convolutional approach is capable of reconstruction at all. Furthermore, the small network size requires few timesteps and a small interlayer featurespace; i.e., $b, d < 10$. This results in a relatively simple set of transitions, allowing us to explore and understand the inner workings of the network.

3.1.1 Example Model

The following data are pulled from a model trained on data produced by the generator in figure 3.1. Figure 3.2 demonstrates the model's loss over time. In this example, b and d were pushed down in order to allow for better

[1]SEE APPENDIX for information on spike rates

comprehension of the internal mechanics; the loss tends to converge more effectively and evenly given more computation power.

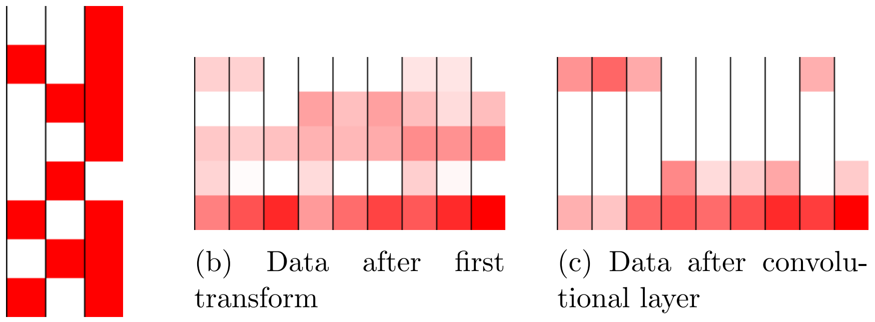


b (timesteps)	8
d	5
Batch size	32
Learning rate	.0005
Training samples	17984
Validation samples	4512

Figure 3.2: Training loss and parameters for model described in 3.1.1

Trained Network Operation

Here, we examine the internal operation of the trained model over a single input.



(a) Input

(b) Data after first transform

(c) Data after convolutional layer

Figure 3.3: Path of data through network, up to final transform