Algorritmo

Making Music with and Visualization of Recurrent Neural Networks

Author:
Aimee Barciauskas
aimee.barciauskas@barcelonagse.eu

Supervisor:
Fernando Cucchietti
Scientific Visualization Group Leader, Barcelona Supercomputing Center (BSC)
fernando.cucchietti@bsc.es

2016 June 27

Abstract

This project is an exploration into neural networks, how to visualize them and how they may be used to produce music. It was done in collaboration with the Scientific Visualization Group of the Barcelona Supercomputing Center.

Project Overview

Apart from the current work (this paper and it's presentation), the project is presented at Sonar Music Festival as part of the Sonar+D creativity and technology marketplace. Visitors to Sonar+D participated in the music generation process by drawing and composing music using interactive applications and voting on sounds produced by these applications.

In addition to the interactive applications, it is important that the booth include an explanatory or educational component: Polling prior to the festival showed popular interest in learning about how artificial intelligence produces music. An integral part of the final presentation is a tool to visualize the network to visitors and educate them on how the new music had been created.

This project is organized as follows:

This project is comprised 2 parts:

1. Can machines demonstrate creativity through machine learning algorithms (neural networks)?
2. How can we visualize neural networks?

Though these parts are self-contained, in practice they are co-dependent. To visualize something one first has to understand it. Futher, the exercises of creating, viewing and interacting with a visualization support understanding.

Part 1: Can machines produce music?

A popular field of research is whether machines can learn to be creative and the most popular algorithms being used to develop this research are neural networks.

Using recurrent neural networks to generate music

Recurrent neural networks are networks built using neurons that have a "memory". During training in traditional feed-forward neural networks, neurons are trained on each input-output pair irrespective of sequence in the training data. Recurrent neural networks account for serially dependent data and as such have become popular in the applications of music generation and speech-to-text recognition and generation.

In practice, a special type of recurrent neuron called long short-term memory (LSTM) is used. Vanilla recurrent neurons often demonstrate vanishing or exploding gradients, and LSTM's avoid this problem using multiple gates which "forget" inputs which are learned to likely be irrelevant.

To develop understanding of RNN's, python code for each neuron type has been developed as part of this project.

Development of single neurons

To have a deep understanding neural networks, it is fundamental to understand the single neuron and how it contributes to the the network. This is reminiscent to fundamentals of dynamic programming: by solving multiple sub-problems, you can arrive at the optimal solution to the larger problem.

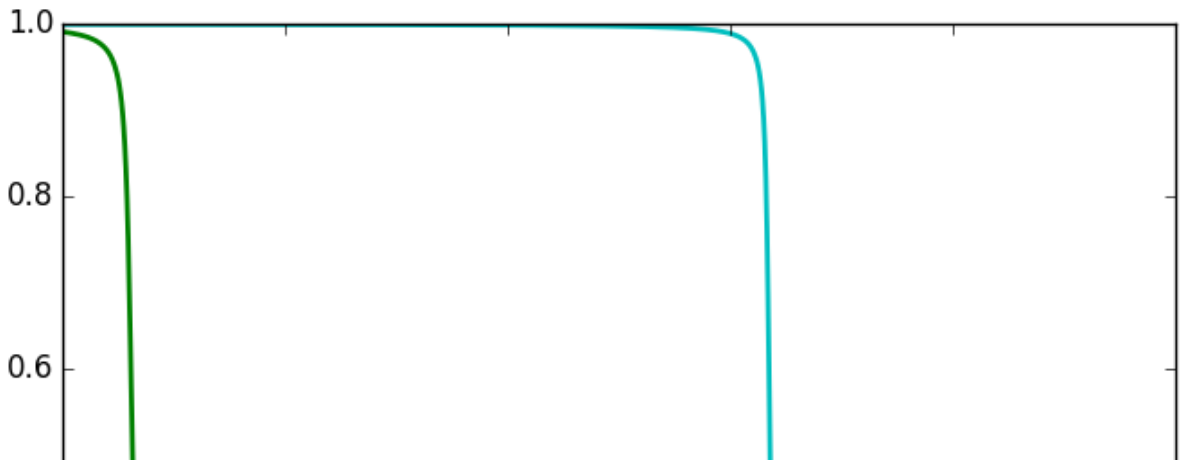
At the final layer, the output of the neurons is passed into a loss function. The loss calculated at the output layer is then chained back through the network in order to determine how each set of weights to each neuron is impacting the loss. The weights are updated accordingly. Subproblems (local gradients) are calculated locally but updated according to their contribution globally (the loss or error)

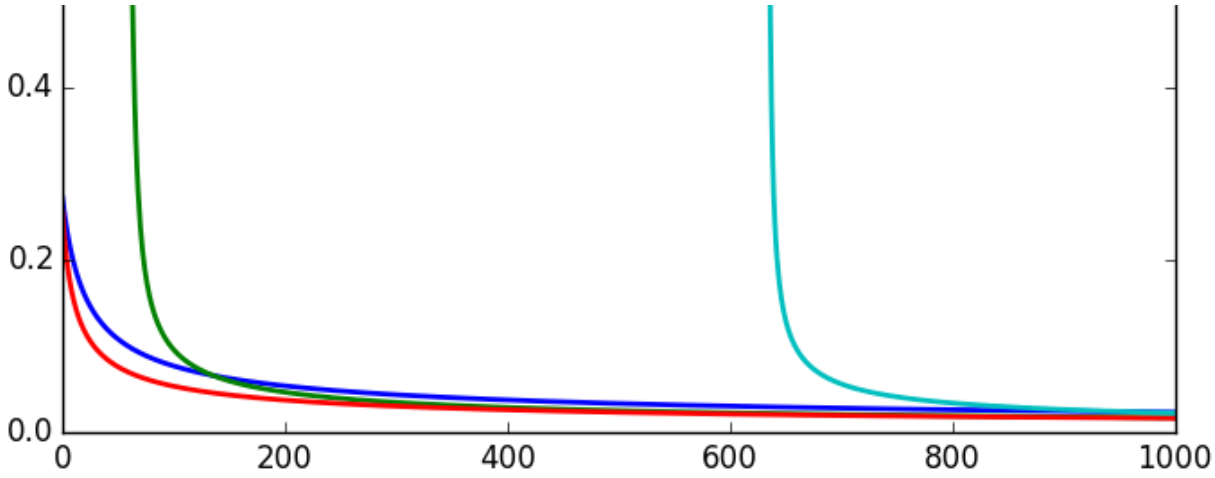
A single neuron

Below is the code of a single neuron followed by it's losses during training.

```
# Single neuron as a linear classifier
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# Simulate X and Y
# XOR pattern
X = np.array([[0,0,1],[0,1,1],[1,0,1],[1,1,1]])
Y = np.array([[0,1,1,0]]).T

# Initial weights, a set for class 0 and a set for class 1
# W is dimension (dim of X x number of classes); an array of weights
# python does it cols x rows
# 1 - like a nnet with 1 neuron
W = 2*np.random.random((3,4)) - 1
loss_all = []

# Sigmoid activation function
for i in range(len(X)):
    obs = X[i,:]
    y_obs = Y[i]
    losses = []
    iters = 1000
    # first neuron's weights
    # we have four sets for future times where we have more than one
 neuron
    weights_neuron1 = W[:,0]
    for iter in range(iters):
```

```
    for iter in range(iters):
        print 'starting iter: ' + str(iter)
        # calculate the input to the first neuron
        input_neuron1 = np.dot(weights_neuron1, obs)
        # calculate activation function
        # sigmoid in this case
        f_neuron1 = 1/(1+np.exp(-input_neuron1))
        # this is the ouptput of the neuron
        # if y = 1, f_neuron is the correct probability
        # if y = 0, 1-f_neuron1 is the correct probability
        true_class_prob = f_neuron1 if y_obs == 1 else (1-f_neuron1)
        loss_neuron1 = 1 - true_class_prob
        losses.append(loss_neuron1)
        dLdF = 1 if y_obs == 1 else -1
        dL = loss_neuron1 * dLdF
        # the gradient for this neuron and this input
        dW = dL * f_neuron1 * (1 - f_neuron1)
        #weights_neuron1 += np.dot(obs, dW)
        weights_neuron1 += obs * dW
    loss_all.append(losses)


x = range(iters)
line = plt.plot(x, loss_all[0],  x, loss_all[1], x, loss_all[2], x,
loss_all[3], linewidth=2)
plt.show()
```





A single recurrent neuron

To understand recurrent neural neurons at the lowest level, code for a single vanilla recurrent neuron and LSTM neuron were developed.

A simple RNN on beer.stackoverflow.com data

A single LSTM neuron

My Simple LSTM Neuron

Music Generating Networks

In one sense, music generation is both a classic application of recurrent neural networks: At each time step, the current step is used to predict the next step, and this is what RNN's are built to do. However there are varied approaches in how to represent a timestep and discretize music. In fact, a deep understanding of digital music will improve future efforts.

Music Generation Experiment

A network using the unique set of chords from the training set is used as a classification task for the neural network. For example, given the training set is comprised 381 unique chords, this is the set of possible classes in the classification task.

The upside of this approach is it presents a classic paradigm with which to represent the music generation task. The downside of this approach is it restricts the output space to the input space. This is resasonable if the goal is to generate music which sounds like the training set. However it misses the mark if the goal is a more creative generation process.

Source code

Technical Specifications

- Keras library for 2-layer LSTM architecture
- AWS GPU-backed AMI

Music Generation for Sonar

Members of the Scientific Visualization group used a neural network composed of 6 LSTM layers of around 150 neurons per layer. The input to the network is a long vector representing the 5 tracks of 5 instruments, having some variation in space due to octave range. An additional dimension is included indicating whether the note is a note previously held.

Throughout Sonar+D, the network was used to create "generations" of songs. Each generation used a progressively better set as ranked by the audience at Sonar.

Part 2: How can we visualize neural networks?

Overview and Motivation

There are 2 objectives in visualizing neural networks: to educate and to understand.

Though neural networks have been around for a while, there is little understanding of why and how neural networks work. This has motivated some academic work in visualizing neural networks to understand what's going on in the different components and levels of neural network architecture and improve performance.

Neural networks are being used in many every day applications and those interested should have an opportunity to understand how they work. The objectives are to provide transparency and satisfy curiousities.

It is the expectation that the audience for the visualization are people with no prior knowledge of computer science or machine learning. However, it was found the application as a standalone feature is probably insufficient material to educate such an audience (see **User Experience Testing**). For Sonar, this is acceptable as the author and colleagues were available to use the visualization as an educational aid and not a standalone resource.

Literature Review

What follows is a review of current efforts in visualizing neural networks focused on both convolutional and recurrent neural networks.

Visualizing and Understanding Convolutional Networks, Zeiler and Fergus

What concepts does a neural network learn to be important for image classification? How do concepts develop after intial layers? Visualization of a convolutional network using a deconvolutional network (or deconvnet) leverages how a deconvnet maps feature activities back to the original pixel input space. Deconvnets can be used to invert a trained neural network to glean learned concepts and visualize them. Details of the inversion are discussed in fine-grained detail in the paper.

Zeiler and Fergus produced visualizations using the popularized convnet architectures developed by LeCun et. al. and Krishevsky et. al.. The visual analysis helped them beat the AlexNet 2012 single-model result by 1.7% using smaller strides (2 vs. 4) and smaller filters (7x7 vs. 11x11).

DeepVis, Jason Yosinski

Jason Yosinski, DeepVis, 2015 published an open source tool, DeepVis, for visualizing a neural network in real time, with an image or camera feed. The tool is a dashboard of the network, enabling the user to visualize and interact with intermediary results of the network.

The strength of DeepVis may be in its simplicity. For example, it includes activation plots values which enable the user to see all the data.

> Although this visualization is simple to implement, we find it informative because all data flowing through the network can be visualized. There is nothing mysterious happening behind the scenes. (pg 4)

Though the features of DeepVis are not innovative themselves, these visualizations yielded new and surprising intuitions:

- Layers demonstrate locality; that is layers become detectors of different real-world objects like flowers or faces. This suggests that intermediate layers become responsible for different concepts important in a well-trained classifer.
- When an image does not include anything from the training set of classes, the real-time probability vector exposed a high sensitivity to small changes in input. In other words, shifting in your chair could mean instead of a cat you are classified as a lamp.
- Although higher layers exhibit greater sensitivity, lower level computations are robust:

> ...the network learns to identify these concepts simply because they represent useful partial information for making a later classification decision.

Yosinski and Zeigler both expressed surprise at finding the features learned by intermediate layers are discernable and important to final classification. This is cool because it suggests the network learns the concepts like text or wheel before it becomes important in classifying books and cars. However, Yonsinski also comments:

> That said, not all features correspond to natural parts, raising the possibility of a different decomposition of the world than humans might expect. These visualizations suggest that further study into the exact nature of learned representations — whether they are local to a single channel or distributed across several — is likely to be interesting. (pg 9)

The features of the tool and links on how to install it are available here: DeepVis.

Visualizing what ConvNets Learn, Andrej Karpathy

Visualizing What Convnets Learn lists and describes useful ways to visualize a variety components of neural networks, including layer activations and visualizing high dimensional feature topologies.

Karpathy also has a cool interactive tool to demonstrate classification by a neural network in 2-dimensions: ConvnetJS demo.

A Neural Network Playground, Daniel Smilkov and Shan Carter

The Neural Network Playground developed by Daniel Smilkov and Shan Carter offers a user-friendly and attractive interactive tool for understanding at a high level how neural networks work and how you might tune them

tune them.

Neural Networks, Manifolds, and Topology, Chris Olah

Colah and Karpathy both provide helpful animations of how space can be warped such that non-linear data is linearly seperable. One thing I like about Colah's article in particular is he breaks this down into it's component parts (weighting, translation and activation function) and provides a visualization of that process in fine-grained detail.

Colah's intuitive explanations and visualizations of the challenge of warping space such that naturally non-linear data may be linearly seperated. I highly recommend a reading to gain an intuition through simple examples. Colah builds on the intuitions gleaned from this simplified examples to theorize about lower-bounds on the dimensionality requirements of neural networks (i.e. the Manifold Hypothesis).

Colah also has some very good articles on visualizing high-dimensional data, which is relevant but perhaps tangential to this topic:

- Visualizing Representations: Deep Learning and Human Beings
- Visualizing MNIST: An Exploration of Dimensionality Reduction

Inceptionism: Going Deeper into Neural Networks, Google

Google also "inverts" a trained neural network classifier to compose images from noise, resulting in the well-known inceptionism of a trained neural network which can be used to create continuous surrealist abstractions of an image class. The methodology is, given you have a trained classifier and you want to determine what types of image patterns the classifier has learned, you start with an image of random noise and tweak the image until you get something the classifier finds to be a banana with high probability. This results in the surreal images now identified with Google's inceptionism idea and help identifiy flaws in a training set.

Understanding LSTMs, Christopher Olah

Olah uses simple diagrams and simplified computational graphs to explain Long Short-Term Memory Networks. He shows through sequential highlighting of a single LSTM neuron how it works like a conveyor belt with multiple gates.

The Unreasonable Effectiveness of Recurrent Neural Networks, Andrej Karpathy

Karpathy's blog post on RNN's has been referenced in just about every other blog post about RNN's I've come across (including mine). He uses super-simple diagrams to show how RNN's can be "unrolled". These super-simple diagrams reflect Karpathy's view of neurons and layers as "building blocks" in neural

These super-simple diagrams reflect Karpathy's view of neurons and layers as "building blocks" in neural networks, and thus can be used to demonstrante how inputs and outputs can have varied manifestations: one-to-one, many-to-one, one-to-many, many-to-many.

This post also includes a very cool visualization of how different neurons "fire" given some input: one can see how different neurons have responsibility for different features of the inputs.

Literature Review: Conclusions

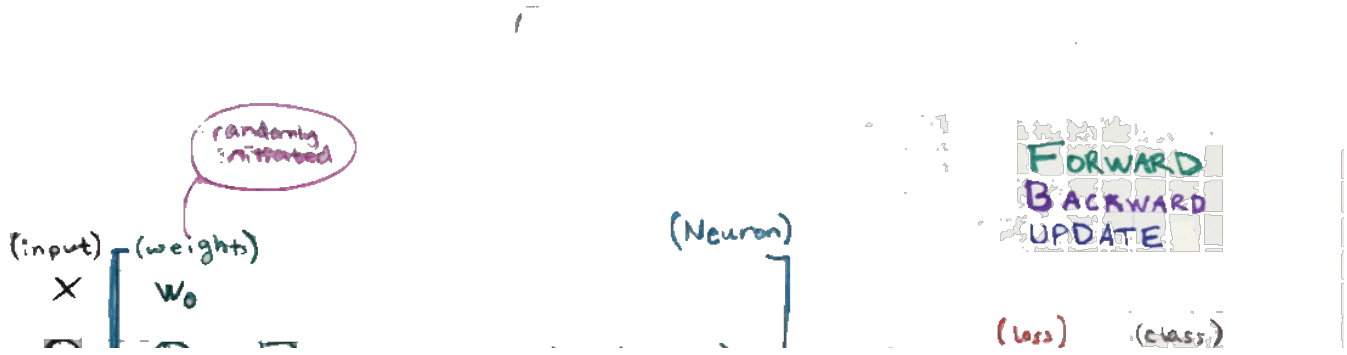
- Approaches by academics have been targeted at research interests, and while not directly instructive in creating experiences for a general public, they still offer insight into neural networks: For example, that intermediate layers also may be identifying and isolating important concepts for classification (and thus are more interpretable than originally thought).
- There have been few attempts at interactivity and none at gamification as a way to tackle the education objective.
- Many of the references included so far are in the domain of image classification, so it will be a thought adventure in how to generalize learnings and approaches into other domains and architectures.

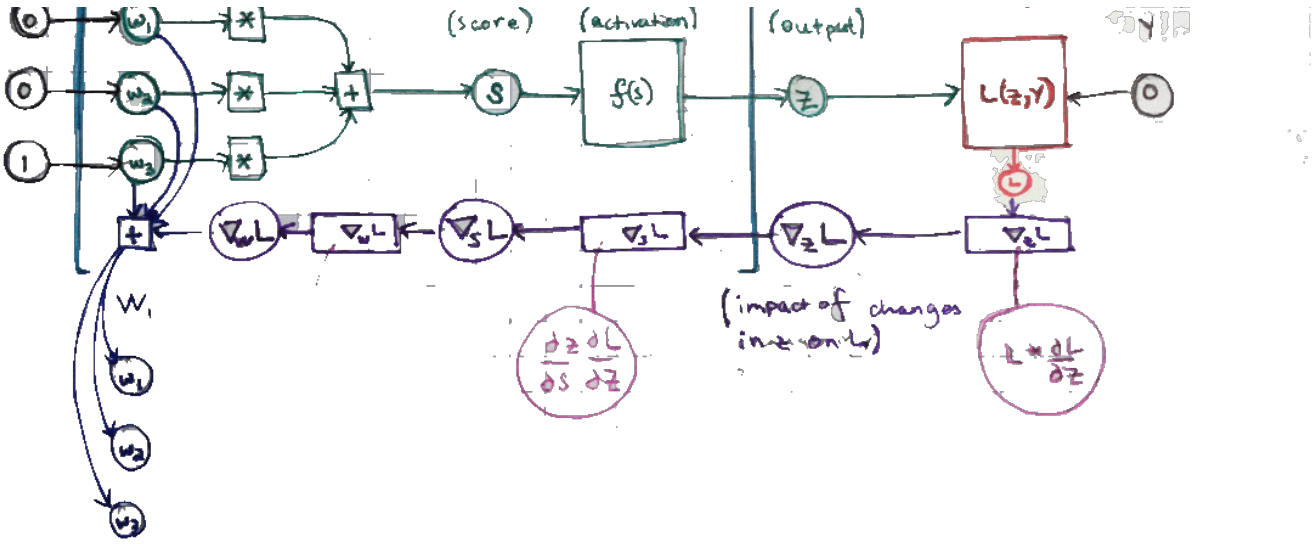
Creative Process

An important part of the creative process is developing a low-level understanding of neurons and neural networks as described in Part 1 of this paper. The second part is sketching ideas on paper and discussing them with colleagues.

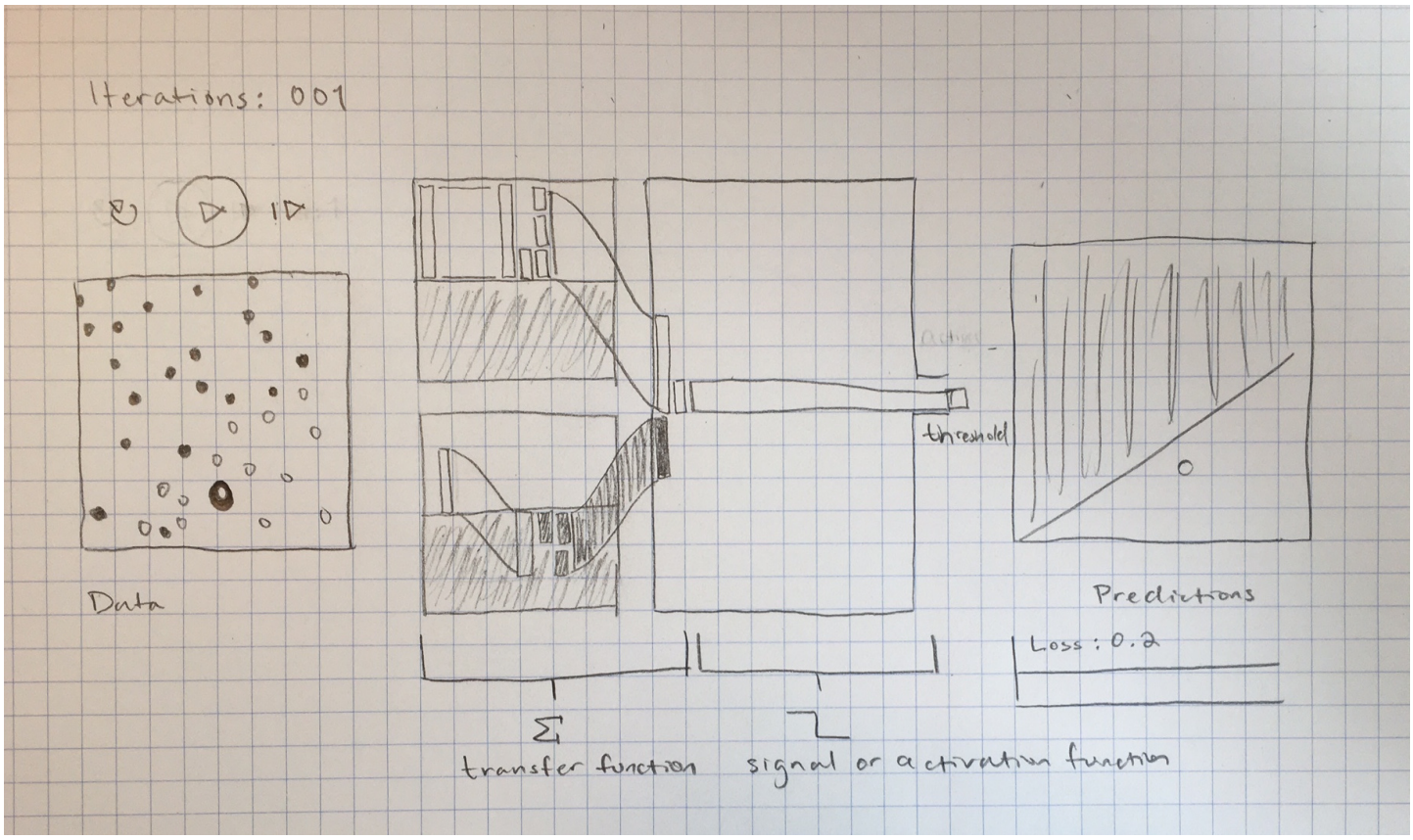
Visualizing a Single Neuron

A vital component in neural networks is back-propagation. The image below visualize back-propagation as a modification on the computational graph.

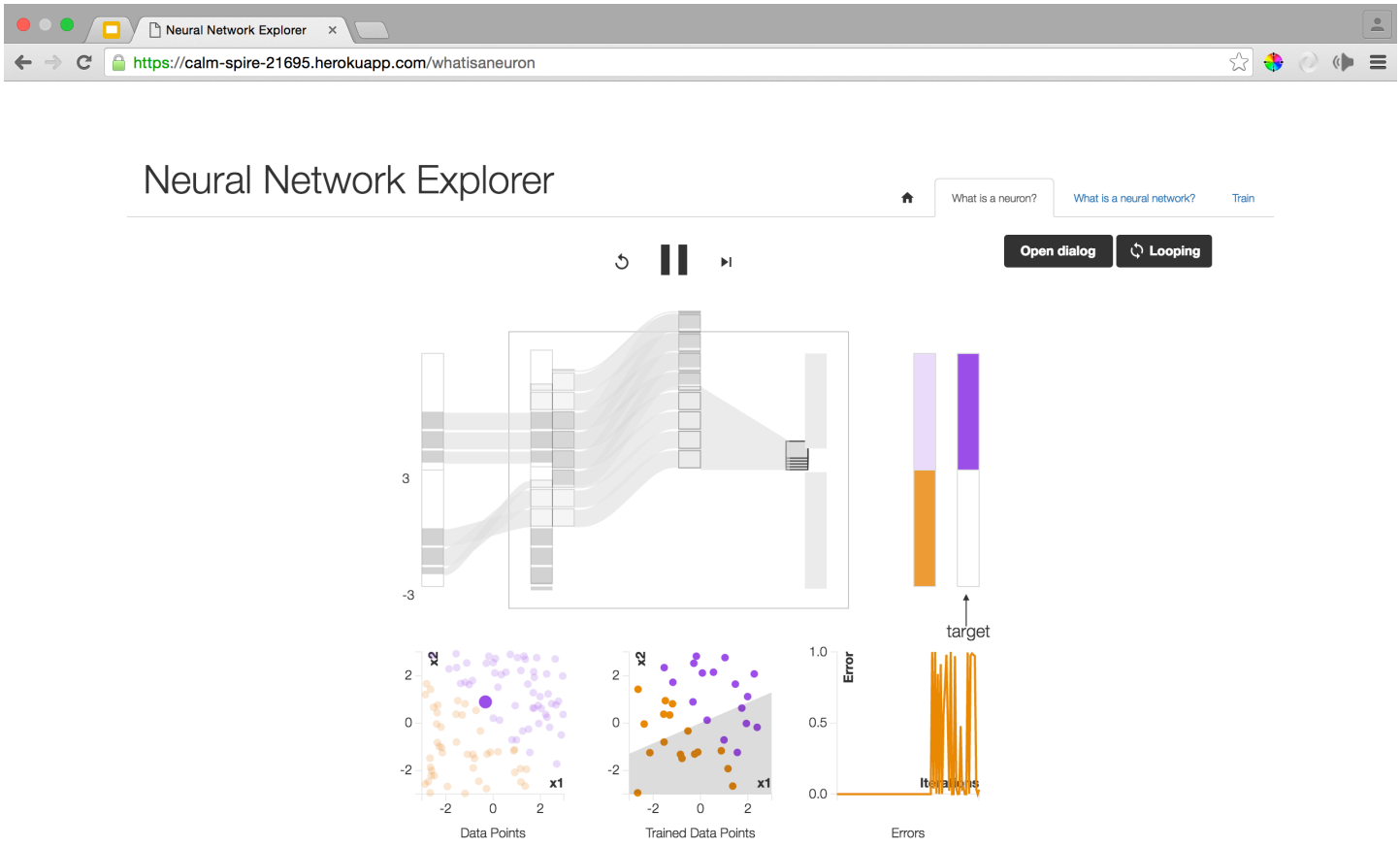




The sketch for the final product:



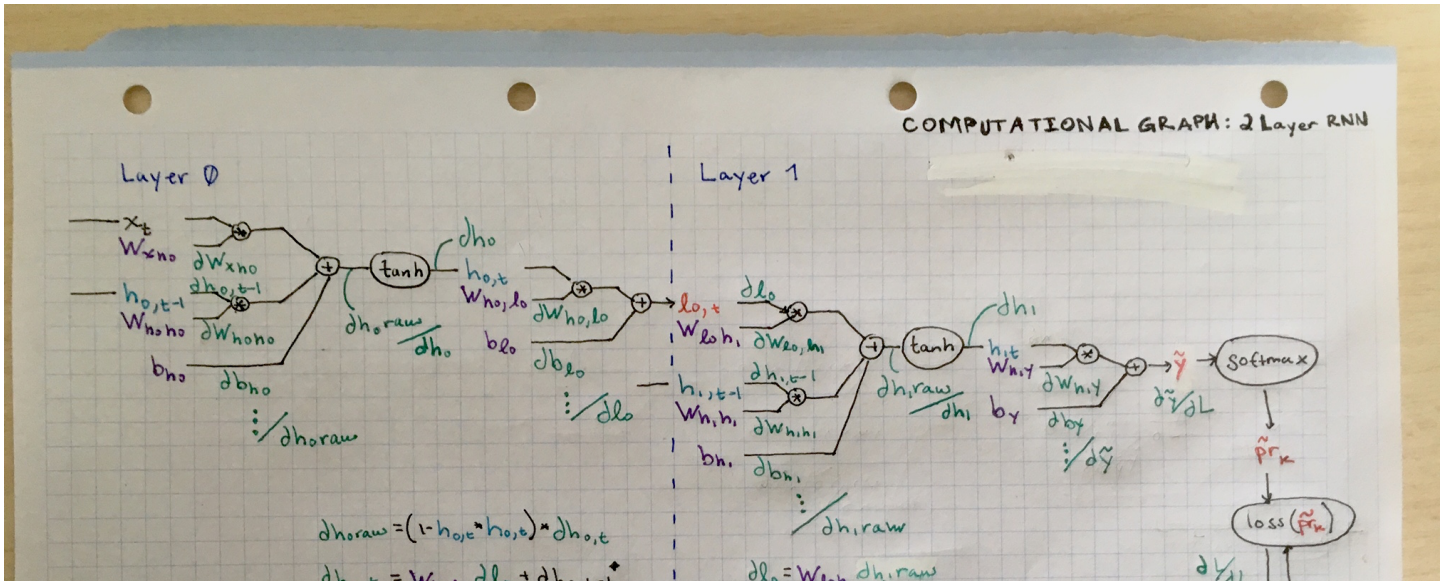
And screenshot:

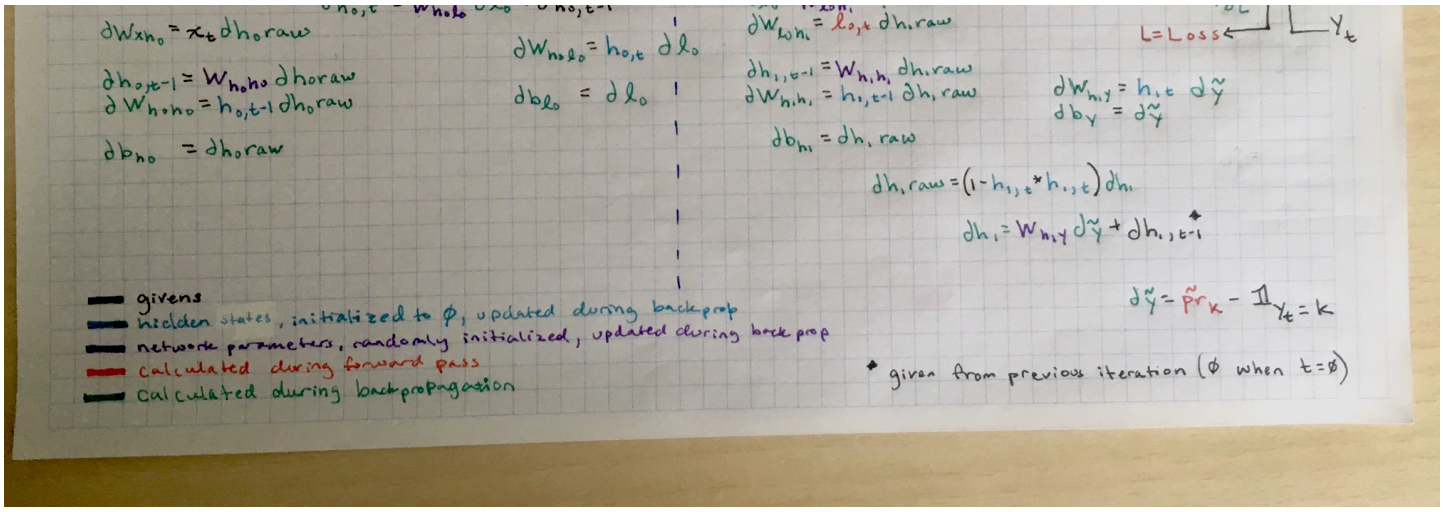


More on this process can be found on my blog: [Visualizing Neurons for a Neural Network]

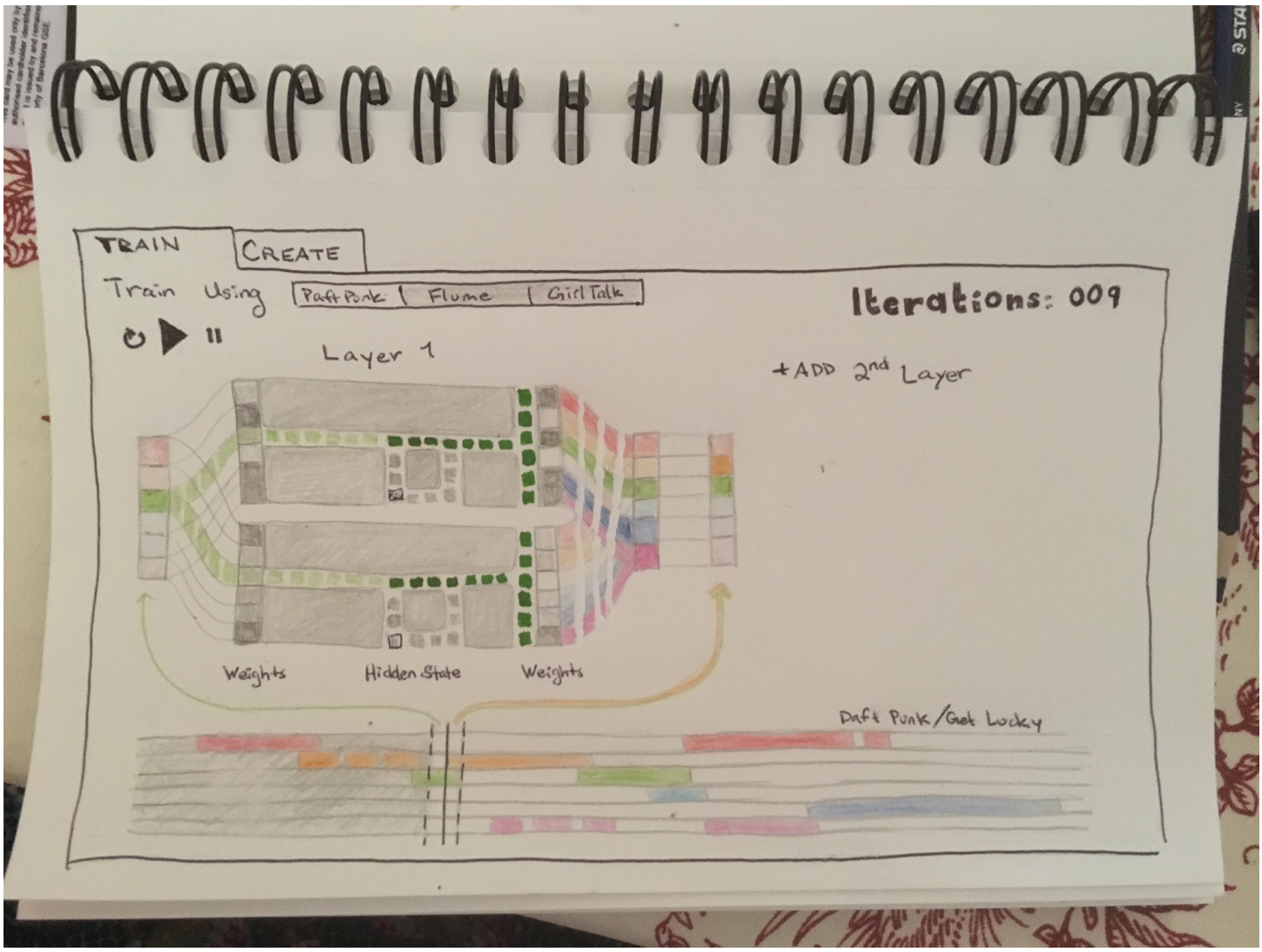
Visualizing Recurrent Neural Networks

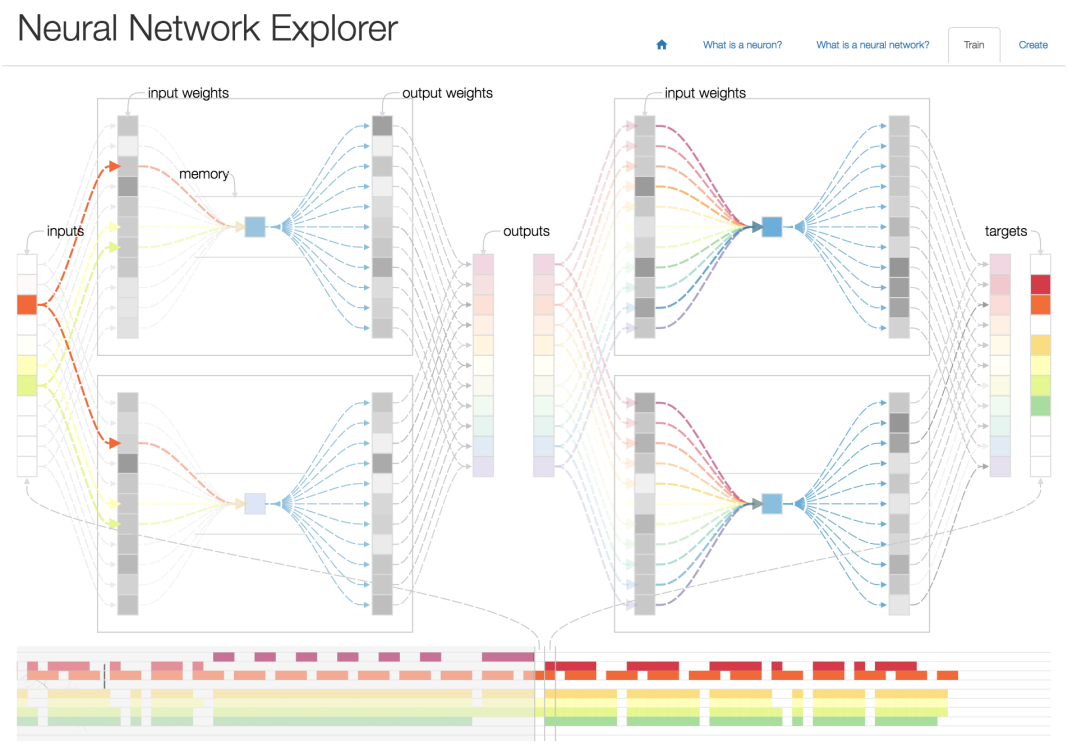
To understand recurrent neural networks, the most useful tool is a computational graph:





The sketch for the final product:





User Experience Testing

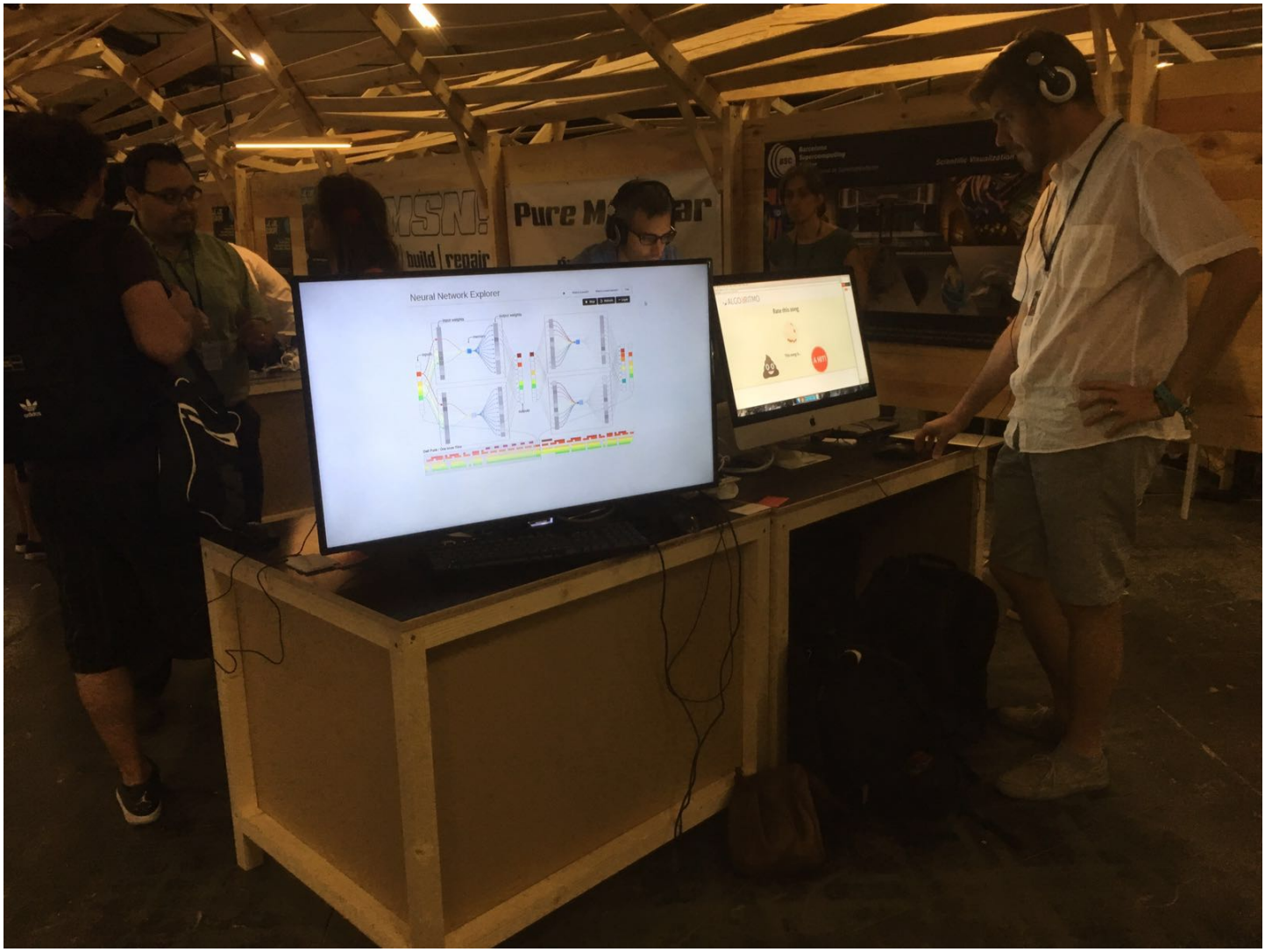
Three user experience testing sessions informed various improvements to the project. The testing sessions were incredibly enlightening and humbling. Of course when working on a project, you are too close to it to possibly judge how other's will experience it, and these sessions exposed the degree to which this is true.

In an interactive visualization the following lessons were learned:

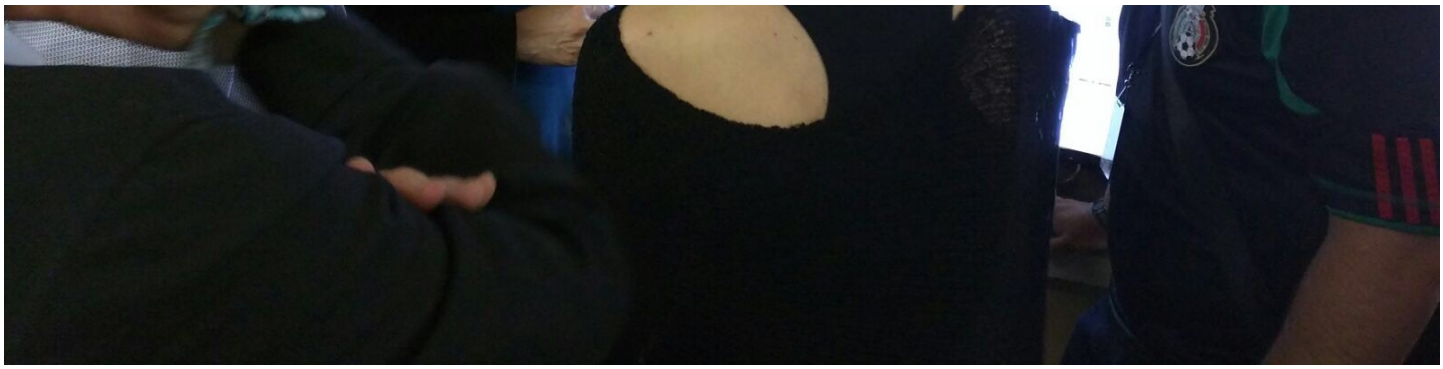
- Users expect to be able to click on everything
- All components should be labeled, either statically or using hovers
- The content in these applications really requires a narrative and explanation. Walkthrus are included for guided learning.

Final Product

The final visualization can be found at this website, and is showcased as part of the BSC booth at Sonar+D:


Neural Network Explorer


Level 3



Conclusions