

# Neural Networks

A (very\*) light introduction

W209 final project Fall 2016

Kyle Hamilton, Jun Luo, Walter Alfredo Erquinigo Pezo, Linfang Yang

# What problem are we trying to solve?

**Goal:** The visualization is an introduction to Neural Networks with a view to facilitate the learning process for incoming Machine Learning students. With this intuition we expect that it will be easier to learn more complex concepts.

The animations and illustrations should provide intuition, but are not designed to be comprehensive. For this reason, we intentionally skipped many details.

**Audience:** The expected audience are students with some very rudimentary understanding of linear models, and a notion of what a Neural Network is.

A background in math is not required.

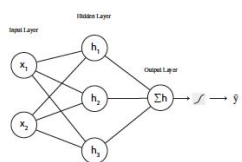
# Idea evolution

The following screens contain initial designs and concepts.

These illustrations show the complexity of networks and the challenge of simplifying these representations without losing important intuition.

It quickly became apparent that showing detailed math equations and trying to account for every operation was cumbersome and confusing.

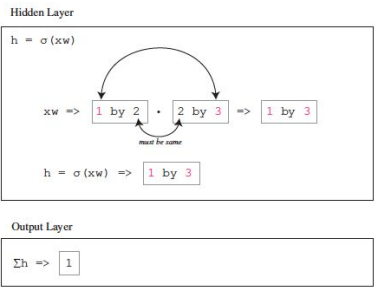
**We decided to focus on one particular aspect of NN mechanics that is not typically visualized - namely the model parameters. This focus makes our project unique in a saturated space.**



**INPUT**  
2 dimensional vector  
 $x: [x_1 \ x_2]$   
shape: 1 by 2

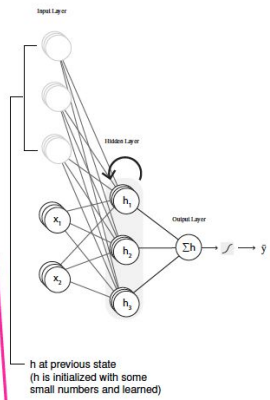
**WEIGHTS**  
Initialize weight matrix with the appropriate shape:  
depends on  $x$  dimension and hidden layer dimension  
 $w: \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$   
shape: 2 by 3

**OUTPUT**  
 $\hat{y}: y$   
shape: 1



## Recurrent Neural Network

1

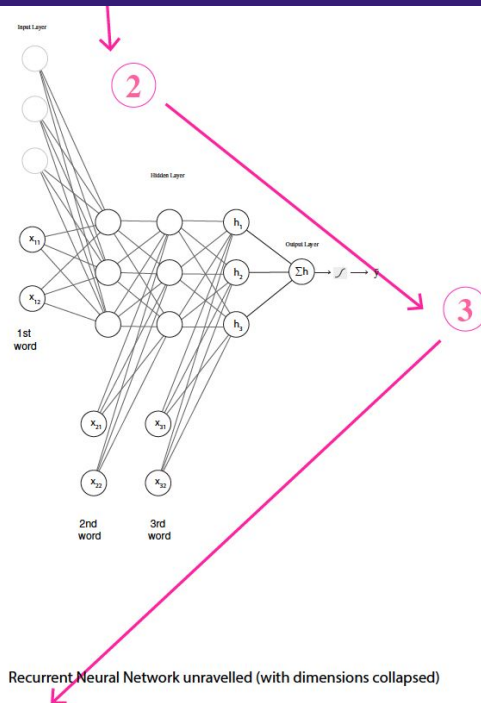


**INPUT**  
3x2 dimensional matrix - a sequence (sentence) of 3 words, where each word is a 2 dimensional embedding  
Each row of  $X$  is a word embedding concatenated with a previous state.

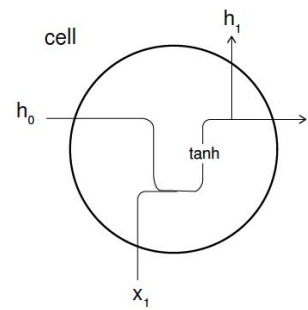
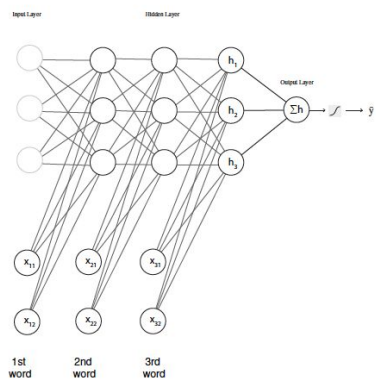
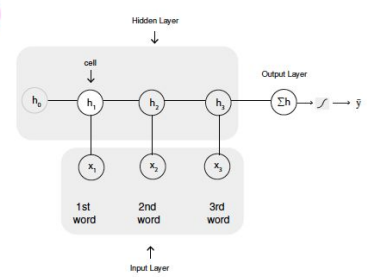
$x: \begin{bmatrix} x_{11} & x_{12} & h_{11} & h_{12} & h_{13} \\ x_{21} & x_{22} & h_{21} & h_{22} & h_{23} \\ x_{31} & x_{32} & h_{31} & h_{32} & h_{33} \end{bmatrix}$   
shape: 3 by (2 + 3)

**WEIGHTS**  
Initialize weight matrix with the appropriate shape:  
depends on  $x$  dimension and hidden layer dimension  
RE-USE FOR EACH ROW OF  $X$  - So each word gets the same weights

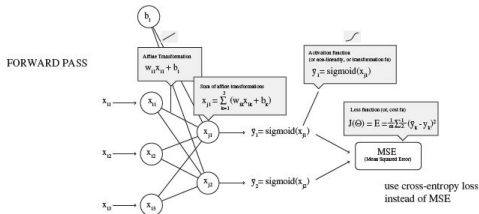
$w: \begin{bmatrix} w_{a11} & w_{a12} & w_{a13} \\ w_{a21} & w_{a22} & w_{a23} \\ w_{b11} & w_{b12} & w_{b13} \\ w_{b21} & w_{b22} & w_{b23} \\ w_{b31} & w_{b32} & w_{b33} \end{bmatrix}$   
shape: (2 + 3) by 3



4

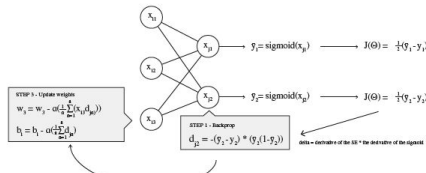


$X: [x_1, x_2, x_3]$	$\rightarrow$	3 dimensional input vector
$W: [w_1, w_2, w_3]$	$\rightarrow$	3 dimensional weight vector
$b: [b_1, b_2, b_3]$	$\rightarrow$	3 dimensional bias vector
$\hat{Y}: [\hat{y}_1, \hat{y}_2]$	$\rightarrow$	2 dimensional output vector "Y hat" -(predictions for 2 class logistic regression)
$Y: [y_1, y_2] \in \{0,1\}$	$\rightarrow$	2 dimensional target vector (training labels)

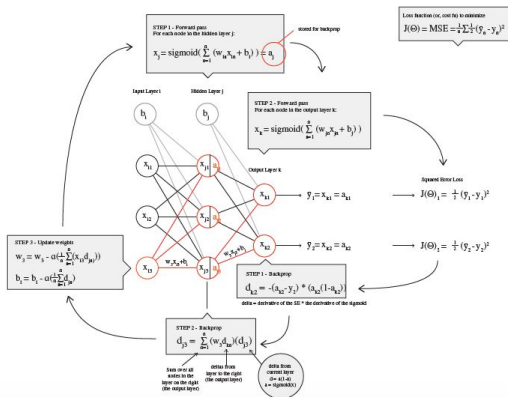


### BACK PROPAGATION

Gradient Descent to minimize the loss (ERROR)



Neural Network with 1 hidden layer (for simplicity, without regularization)  
Based on [http://ufldl.stanford.edu/wiki/index.php/Backpropagation\\_Algorithm](http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm)



Reference [http://ufldl.stanford.edu/wiki/index.php/Backpropagation\\_Algorithm](http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm)

$$\begin{aligned} \frac{\partial}{\partial W_{ij}^{(0)}} J(W, k) &= \left[ \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial W_{ij}^{(0)}} J(W, k; x^{(i)}, y^{(i)}) \right] + \lambda W_{ij}^{(0)} \\ \frac{\partial}{\partial W_{ij}^{(1)}} J(W, k) &= \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial W_{ij}^{(1)}} J(W, k; x^{(i)}, y^{(i)}) \end{aligned}$$

For an output node, we can directly measure the difference between the network's activation and the true target value, and use that to define  $\delta^{(a)}_i$  (where layer  $a$  is the output layer):

For each output unit  $i$  in layer  $al$  (the output layer), set

$$d_i^{(s_i)} = \frac{\partial}{\partial z_i^{(s_i)}} \frac{1}{2} \|y - h_{WD}(x)\|^2 = -(y_i - a_i^{(s_i)}) \cdot f'(z_i^{(s_i)})$$

For  $l = a, b, c$ ,  $W_{l,1}, W_{l,2}, W_{l,3}$  denote

For each node  $i$  in layer 1, set

$$\delta_i^{(j)} = \left( \sum_{j=1}^{p_{i+1}} W_{ji}^{(j)} \delta_j^{(j+1)} \right)$$

Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b, x, y) = a_j^{(l)} \delta_i^{(l+1)},$$

**Update weights**

$$W_0^{(j)} - W_V^{(j)} = \alpha \frac{\partial}{\partial W_V^{(j)}} J(W, \mathbf{t})$$

$$b_1^{(2)} - b_1^{(1)} = -\alpha \frac{\partial}{\partial W_1} J(W, \xi)$$

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T,$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

James Kunz 2: <https://github.com/datasci-w266/main/blob/master/assignment>

James Kunz 2:  $\operatorname{argmax}_{\theta} L(D, \theta)$ 

James Kunz 2:  $\text{sigmoid}(x) = s(x)(1-s(x))$

James Kunz 2:  $1 - \tanh^2$

# Data

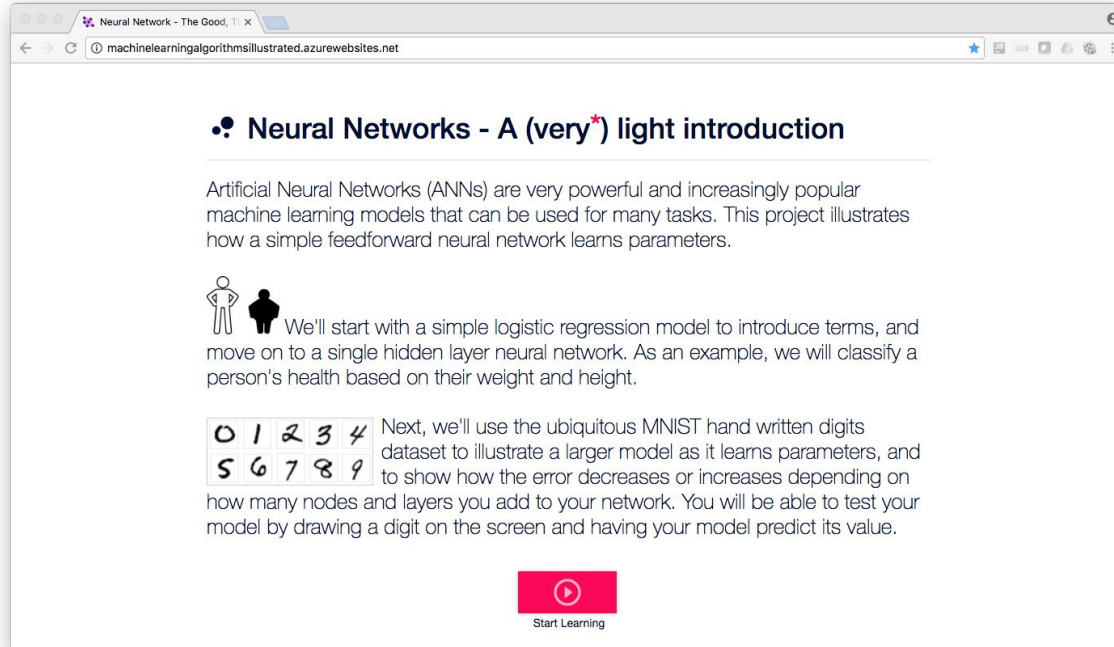
## Learn page

We introduce concepts with a mere **6-8 data points** arbitrarily selected to illustrate a problem. We then use those data points to **pre-train** two simple models and capture intermediate data such as weights and losses, for the chart animations. This data lives in the project github repository.

## Play page


We generate our dataset by training a neural network model **on the fly** using the **MNIST dataset**. To be clear, the data we are visualizing are the intermediate results (weights and losses) of the network, and not the MNIST dataset itself.


# Demo




**• Neural Networks - A (very\*) light introduction**

Artificial Neural Networks (ANNs) are very powerful and increasingly popular machine learning models that can be used for many tasks. This project illustrates how a simple feedforward neural network learns parameters.

 We'll start with a simple logistic regression model to introduce terms, and move on to a single hidden layer neural network. As an example, we will classify a person's health based on their weight and height.

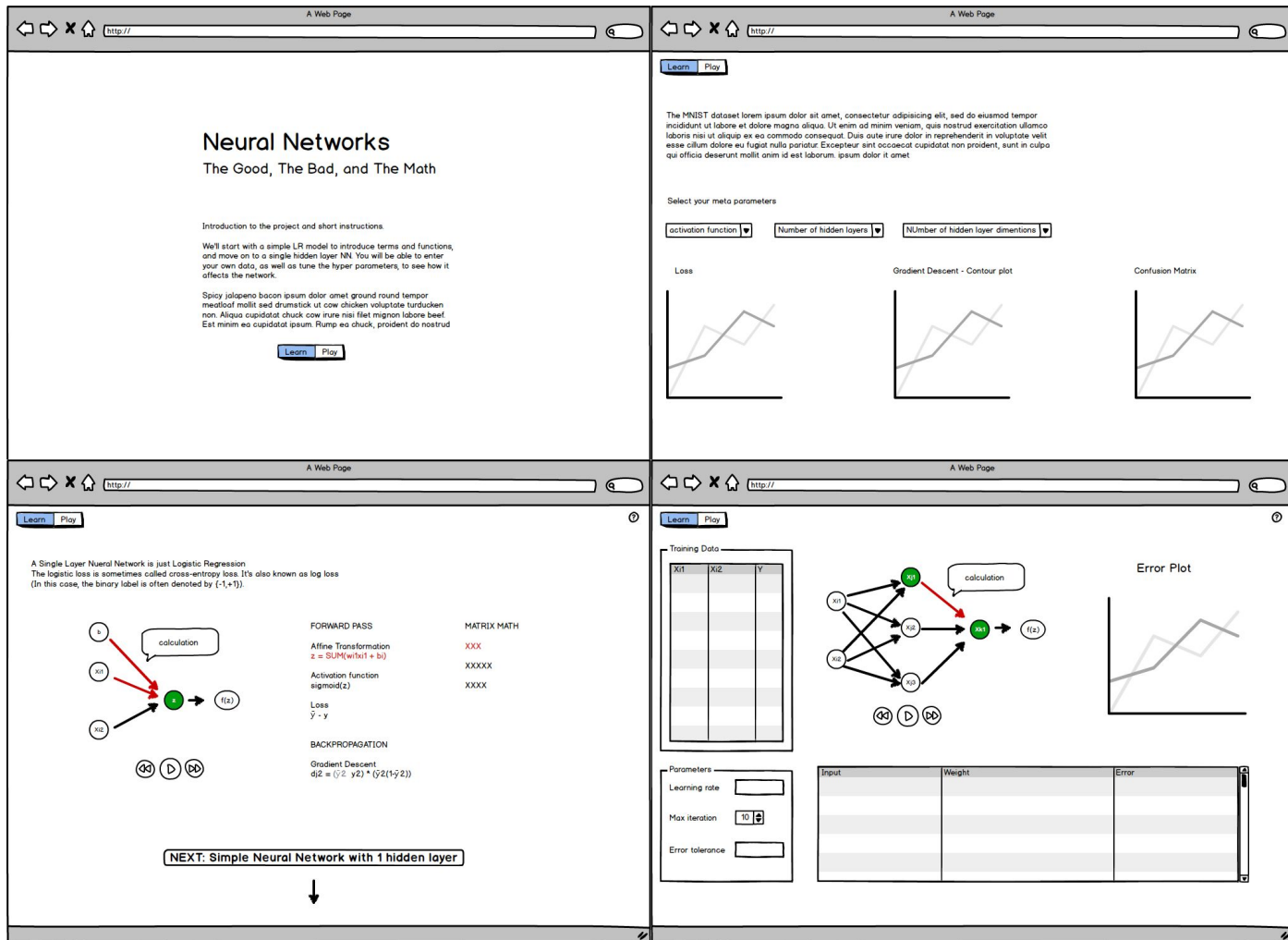
 Next, we'll use the ubiquitous MNIST hand written digits dataset to illustrate a larger model as it learns parameters, and to show how the error decreases or increases depending on how many nodes and layers you add to your network. You will be able to test your model by drawing a digit on the screen and having your model predict its value.

  
Start Learning

<http://machinelearningalgorithmsillustrated.azurewebsites.net>

First steps to  
simplification

Original  
Balsamiq  
Mockups





# Iterations of the Learn view

Learn

To demonstrate the mechanics of a simple feedforward neural network, we'll use a toy example consisting of a 2 dimensional feature vector <weight, height> to predict a binary outcome of healthy or unhealthy. You may notice, that a Neural Network without any hidden layers is just Logistic Regression. The point of the exercise is to train a model that produces outcomes as close to the true values as possible. Or in other words, to "minimize the loss".

This is done in two main phases, namely, a feedforward phase where all the data passes forward through the network so that we can calculate the loss. In other words, on average, how off are we from the ground truth. In the second phase, we backpropagate through the network using gradient descent so that we can update our model parameters (weights). We repeat these two phases until the algorithm converges, or until our error is sufficiently small.

Model - Logistic Regression

Input layer: 1  
Output layer: 1

Data & Decision boundary (hyperplane)

Training Loss

Model parameters (weights)

Model - Neural Network

Input layer: 1  
Hidden layer: 1  
Output layer: 1

Data & Decision boundary (hyperplane)

Training Loss

Model parameters (weights)

That was all great until a couple of tall skinny guys came in catching up hairballs, and our data was no longer linearly separable. Hidden layers to the rescue!

Play

To start playing, let's make a model.

Draw a digit in the box below, and click the "recognize" button.

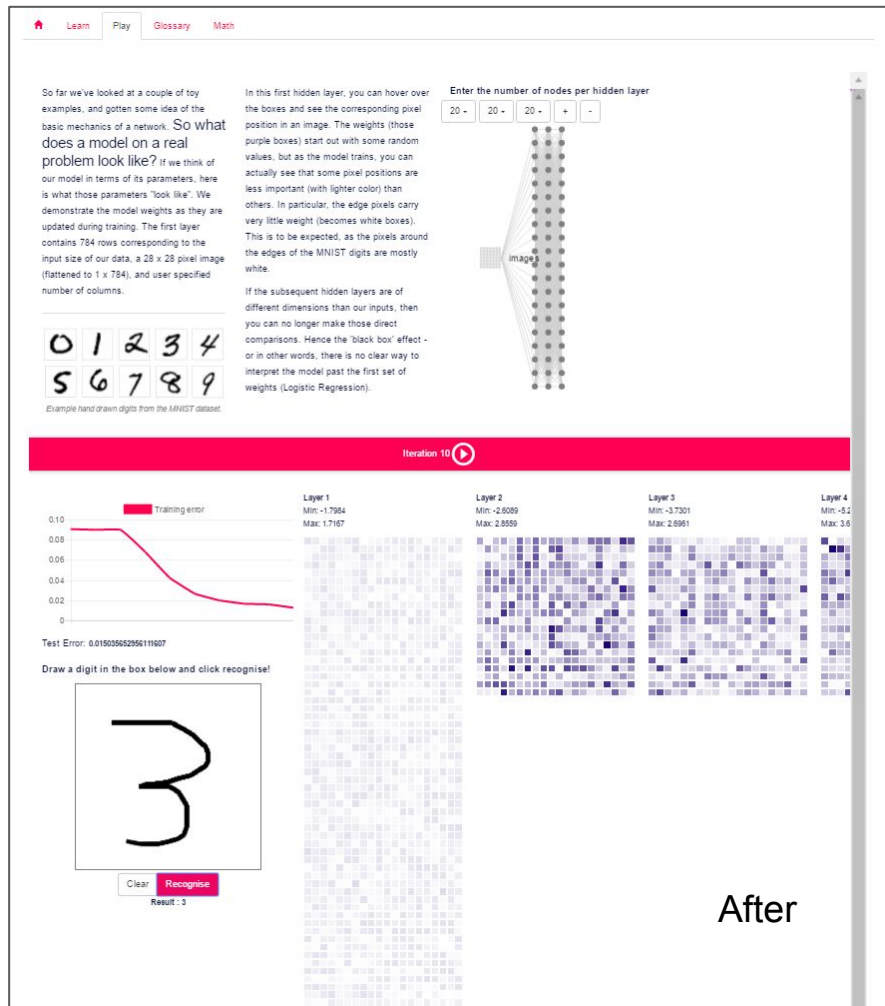
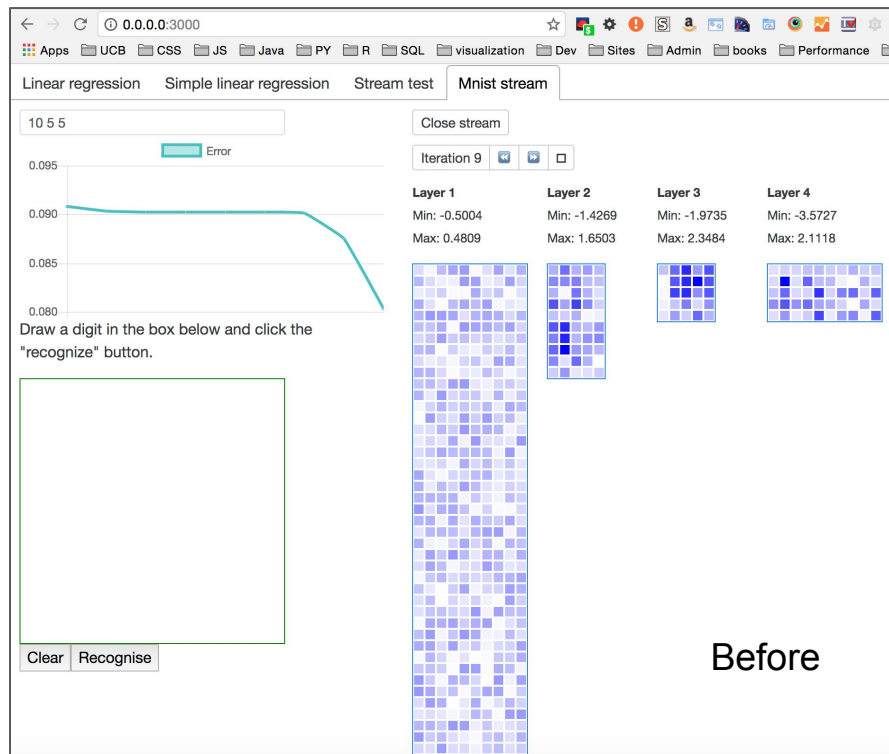
Clear Recognize Result: ?

Learn

On the first iteration of the project it was difficult to make the segue from the Learn view to the Play view. An important detail that made sense of this transition, was the addition of the weights boxes animations on the Learn view which are reflected in the Play view, and the addition of the network graph illustration on the Play view which is reflected in the Learn view.

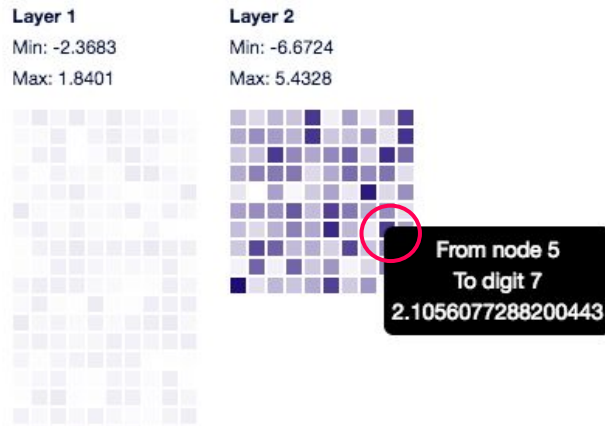
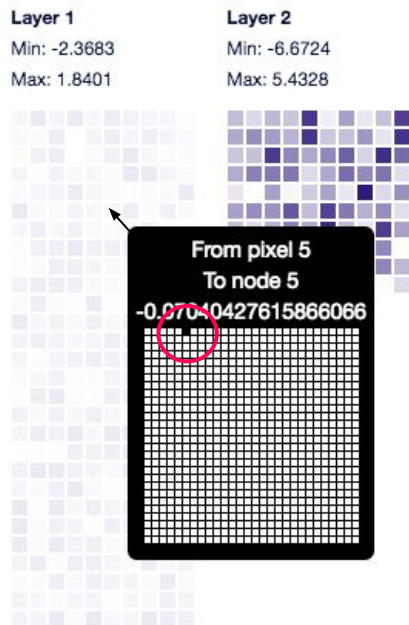
# Iterations of the Play view

## Consolidating colors and style

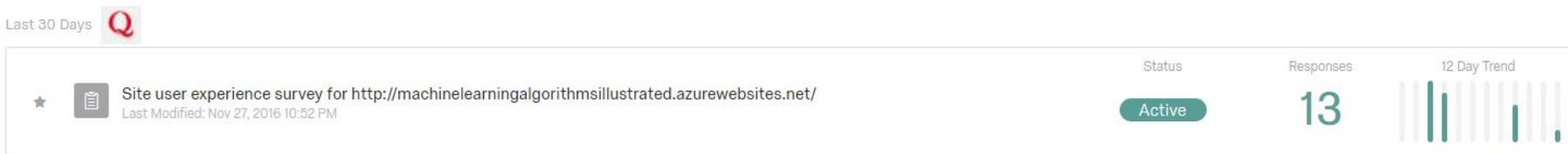


# Insights from the Play view

The color of each box and its position in the data matrix allows the user to clearly see which pixels are more important than others. Important meaning weight absolute value is much greater than 0.



# Major takeaways from usability study



Most users **skipped** reading the texts.

Users were **confused** about which buttons to click.

Users were confused about **math terms**, such as LR, NN.

Users didn't understand the significance of 'linear separability' and "decision boundaries"

User cannot make **connections** between charts, such as weight line and weight box are the same thing

# Consolidated Usability Test Results

## Must

- Add clear labels to charts (Done!)
- Use text format (font and color) to highlight key terms (Done!)
- Provide clear instructions or visual indicators to guide user through intended actions (Done!)
- Improve animation responsiveness (Done! can be further improved)

## Should

- Improve description and instructions so that users can easily understand what they are reading (Done!)
- In "Play" page, provide example and brief introduction of MNIST dataset (Done!)

# Consolidated Usability Test Results (continued)

## Could

- Add another section to show how a linear model fails when applied to nonlinear separable data.
- Create animation of backpropagation in NN learning

## Won't

- It is very difficult to explain all the mathematical details behind neural network learning. It does not seem to be helpful to add more math.

# Sample survey results

Q1 - Thanks for taking the time to answer this survey. It will take you about 15 minutes for this survey.

Section 1, prerequisites(Answer before visiting site) What's your math level in terms of machine learning?Site url: <http://machinelearningalgorithmsillustrated.azurewebsites.net/>

#	Answer	%	Count
1	Not much.	37.50%	3
2	Understand simple logistic regression, and gradient descent.	12.50%	1
3	Understand simple neural network and knows about layers and weights.	25.00%	2
4	Understand different activation function and loss function.	25.00%	2
	Total	100%	8

Q19 - Did this project help you get some insight into the mechanics of NNs?

#	Answer	%	Count
1	Yes	50.00%	4
2	Maybe	37.50%	3
3	No	12.50%	1
	Total	100%	8

Q20 - Was there anything particularly frustrating or unclear?

#	Answer	%	Count
1	Yes, home page	12.50%	1
2	Yes, play page	25.00%	2
4	Yes, learn page	25.00%	2
5	Yes, TL;DR page	12.50%	1
3	No	25.00%	2
	Total	100%	8

Q21 - What did you like best?

#	Answer	%	Count
1	Home	12.50%	1
2	Learn	37.50%	3
3	Play	37.50%	3
4	TL;DR	12.50%	1
	Total	100%	8

# Neural Networks - The Good, The Bad, and The Math

Introduction to the project and short instructions.

We'll start with a simple LR model to introduce terms and functions, and move on to a single hidden layer NN. You will be able to enter your own data, as well as tune the hyper parameters, to see how it affects the network.

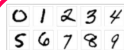
Learn Play

## ☛ Neural Networks - A (very\*) light introduction

Artificial Neural Networks (ANNs) are very powerful and increasingly popular machine learning models that can be used for many tasks. This project illustrates how a simple feedforward neural network learns parameters.



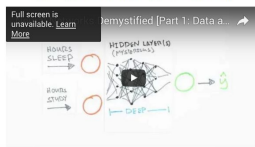
We'll start with a simple logistic regression model to introduce terms, and move on to a single hidden layer neural network. As an example, we will classify a person's height based on their weight and height.



Next, we'll use the ubiquitous MNIST hand written digits dataset to illustrate a larger model as it learns parameters, and to show how the error decreases or increases depending on how many nodes and layers you add to your network. You will be able to test your model by drawing a digit on the screen and having your model predict its value.



If you are not at all familiar with Neural Networks, @stephenswch explains them really nicely in a short video series below:



## Changes to the home page after usability testing

- Change the title to reflect revised project scope
- Add explanatory text. Short and sweet in big letters
- Add icons to break up the text.
- Remove choice of buttons on home page
- Make the button on the homepage the same as the play buttons in Learn and Play, so that the user can learn its meaning naturally.
- Move the video link to the home so it doesn't distract on the Learn view.



# Changes to the Learn view after usability testing

The screenshot shows a web application interface for learning. At the top, there's a navigation bar with tabs: Learn, Play, Glossary, and Math. The 'Learn' tab is active. Below the navigation bar, there's a main content area with several sections:

- To demonstrate the mechanics of a simple feedforward neural network.** This section explains the process of training a model using gradient descent to minimize the loss.
- Model parameters (weights):** This section explains that the parameters of a model determine how much weight to assign to each feature in the data. It also mentions that the parameters are represented by purple squares in the charts below.
- Training loss:** This section explains that a function of the difference between estimated and true values for an instance of data is used for parameter estimation.
- Decision boundary (hyperplane):** This section explains that a hypersurface partitions the underlying vector space into sets, one for each class. It also mentions that the decision boundary is represented by a red line in the charts below.

Below the text sections, there's a large red bar with a play button icon. Below this bar, there's a paragraph explaining that a logistic regression model will classify all the points on one side of the decision boundary (the red line) as belonging to one class and all those on the other side as belonging to the other class. It also mentions that the decision surface is a hyperplane, or in the case of this 2 dimensional toy example, a line, then the classification problem is linear, and the classes are linearly separable. It ends with the instruction: "Press play above to see this model learn a linear decision boundary."

At the bottom of the page, there are three charts:

- Model - Logistic Regression:** This chart shows the input layer (4) and output layer (3). It also shows a diagram of a neural network with 4 input nodes and 3 output nodes.
- Data & Decision boundary (hyperplane):** This chart shows a scatter plot of data points (Height in ft vs Weight in lb) with a red line representing the decision boundary. The data points are colored based on their class (purple for one class, black for the other).
- Training Loss:** This chart shows the training loss over iterations (0 to 160). The loss starts at approximately 1.0 and decreases rapidly, reaching near zero by iteration 100.
- Model parameters (weights):** This chart shows the output layer weights. It lists the inputs and the corresponding weights: 4.055958809 and -3.9429125428.

- Add a Glossary tab

- Remove redundant back/forward buttons

- Provide definitions of each term as it relates to the charts. The terms mirror the chart labels.

- Provide illustration showing exactly where in the network the weights play a role. Further enforced with shape and color (purple boxes). Also included on the "Math" view.

- Add large text explanations of decision boundaries and linear separability

- Add scrolling to top of page upon Play button click to bring the relevant animation into the fore.

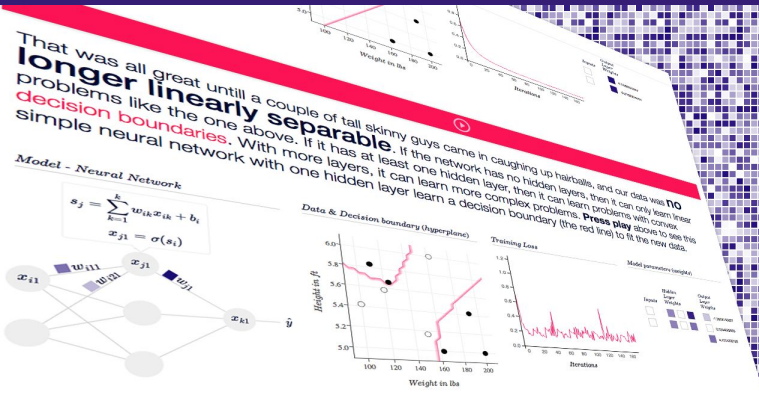
- Add appropriate graph labels

# Tools & technology choices



- HTML/CSS/JS (jQuery)
- D3
- Plotly.js
- Node.js
- React
- Brain.js
- Balsamiq for storyboards
- Photoshop/Illustrator for graphics and comps
- Azure App Services as website hosting solution(continuous deployment)
- AWS VM for compute intensive tasks such as training NN models on the fly
- Github for source control
- IPython notebook for training models offline
- Berkeley survey service as user survey solution (<https://berkeley.qualtrics.com>)

*\*Tools explored as options that weren't used in the project: Flask, TensorFlow, Docker container cluster, Conrec.js*



Berkeley project site:

<https://www.ischool.berkeley.edu/projects/2016/neural-networks-very-light-introduction>

Machine learning algorithms illustrated site:

<http://machinelearningalgorithmsillustrated.azurewebsites.net/>

Project GitHub urls:

[https://github.com/kyleiwaniec/ML\\_algorithms\\_illustrated](https://github.com/kyleiwaniec/ML_algorithms_illustrated)

<https://github.com/yanglinfang/machinelearningalgorithmsillustrated>

User survey link:

[https://berkeley.qualtrics.com/SE/?SID=SV\\_8odiD2hhiltQfvT](https://berkeley.qualtrics.com/SE/?SID=SV_8odiD2hhiltQfvT)