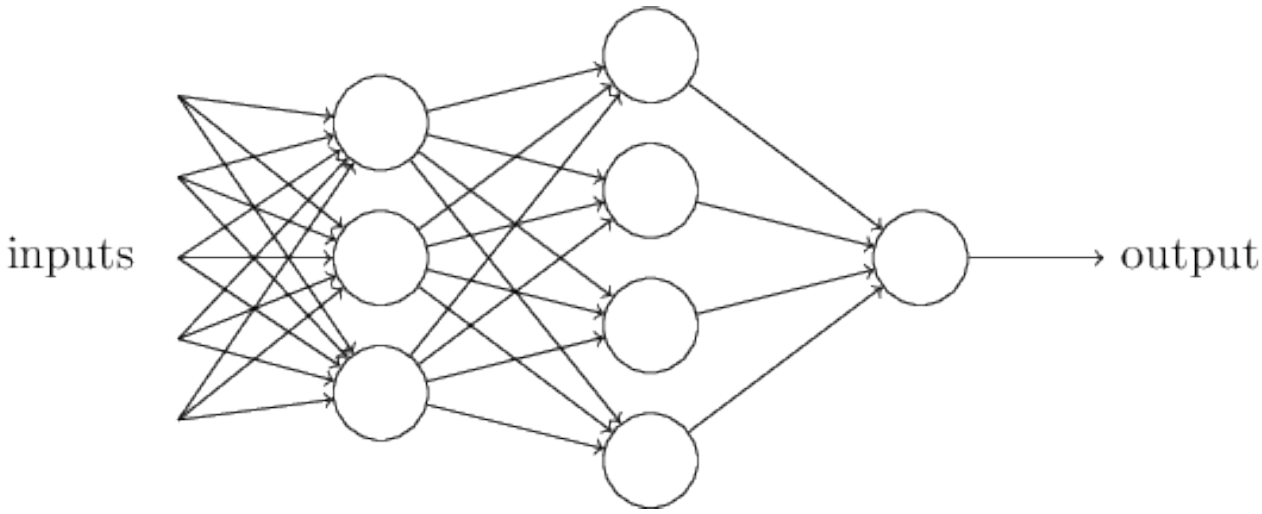
Artificial Neural Network - number recognition. (Neuron.h, Net.h, Map.h, gamewidget_3.h, gamewidget_3.cpp)

http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/
https://en.wikipedia.org/wiki/Artificial_neural_network
http://neuralnetworksanddeeplearning.com/chap1.html

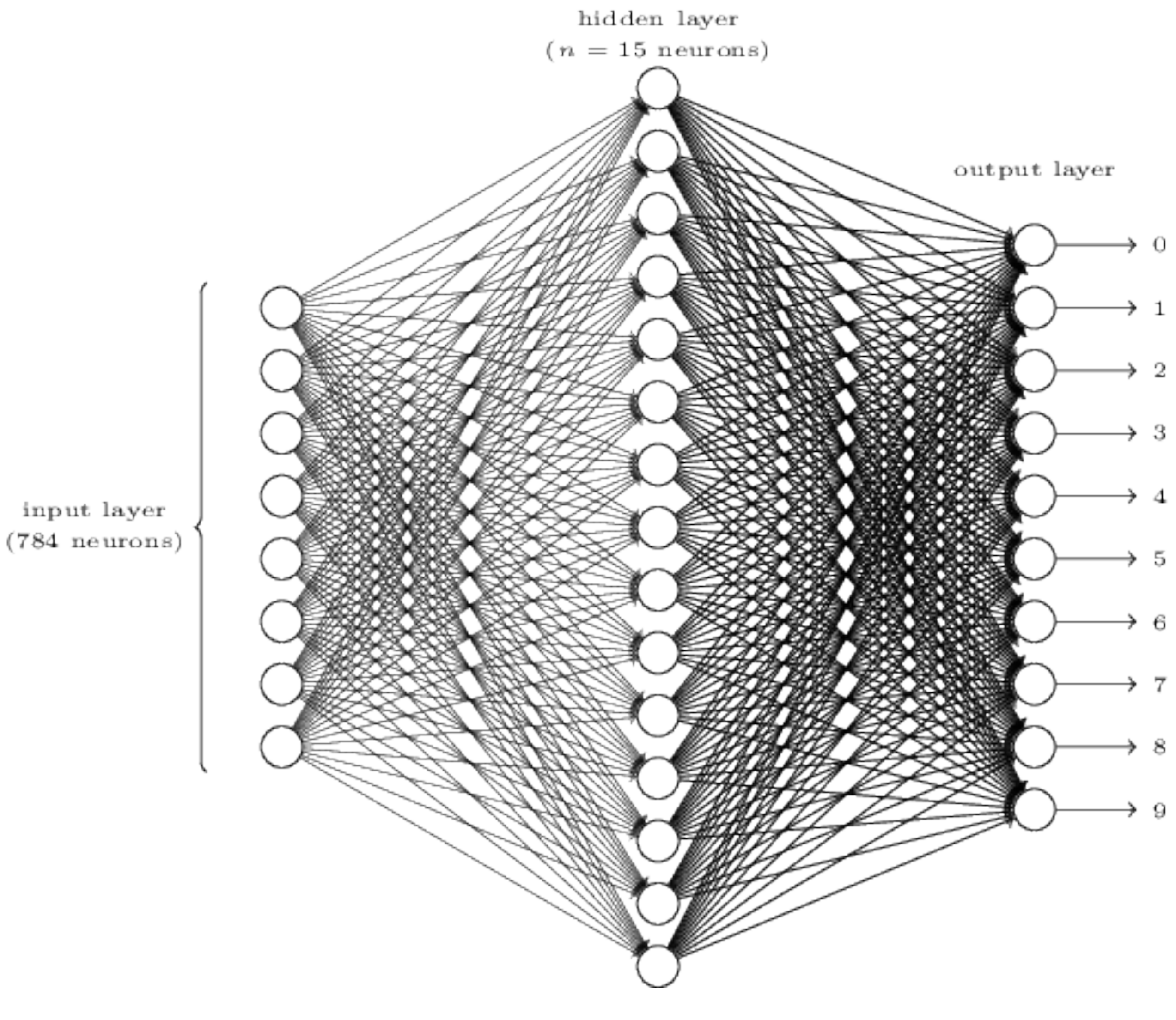
The idea is to take a large number of handwritten digits, known as training examples,

and then develop a system which can learn from those training examples. In other words, the neural network uses the examples to automatically infer rules for recognizing handwritten digits.



In this network, the first column of perceptrons - what we'll call the first *layer* of perceptrons - is making three very simple decisions, by weighing the input evidence. What about the perceptrons in the second layer? Each of those perceptrons is making a decision by weighing up the results from the first layer of decision-making. In this way a perceptron in the second layer can make a decision at a more complex and more abstract level than perceptrons in the first layer. And even more complex decisions can be made by the perceptron in the third layer. In this way, a many-layer network of perceptrons can engage in sophisticated decision making.

To recognize individual digits we will use a three-layer neural network:



The input layer of the network contains neurons encoding the values of the input pixels. Our training data for the network will consist of many 28 by 28 pixel images of scanned handwritten digits, and so the input layer contains 784=28×28 neurons.
The second layer of the network is a hidden layer. The example shown illustrates a small hidden layer, containing just n = 64 neurons.
The output layer of the network contains 10 neurons.

3.1. NeurNet Qt

It is recommended to implement Neural Network as a separate file with:

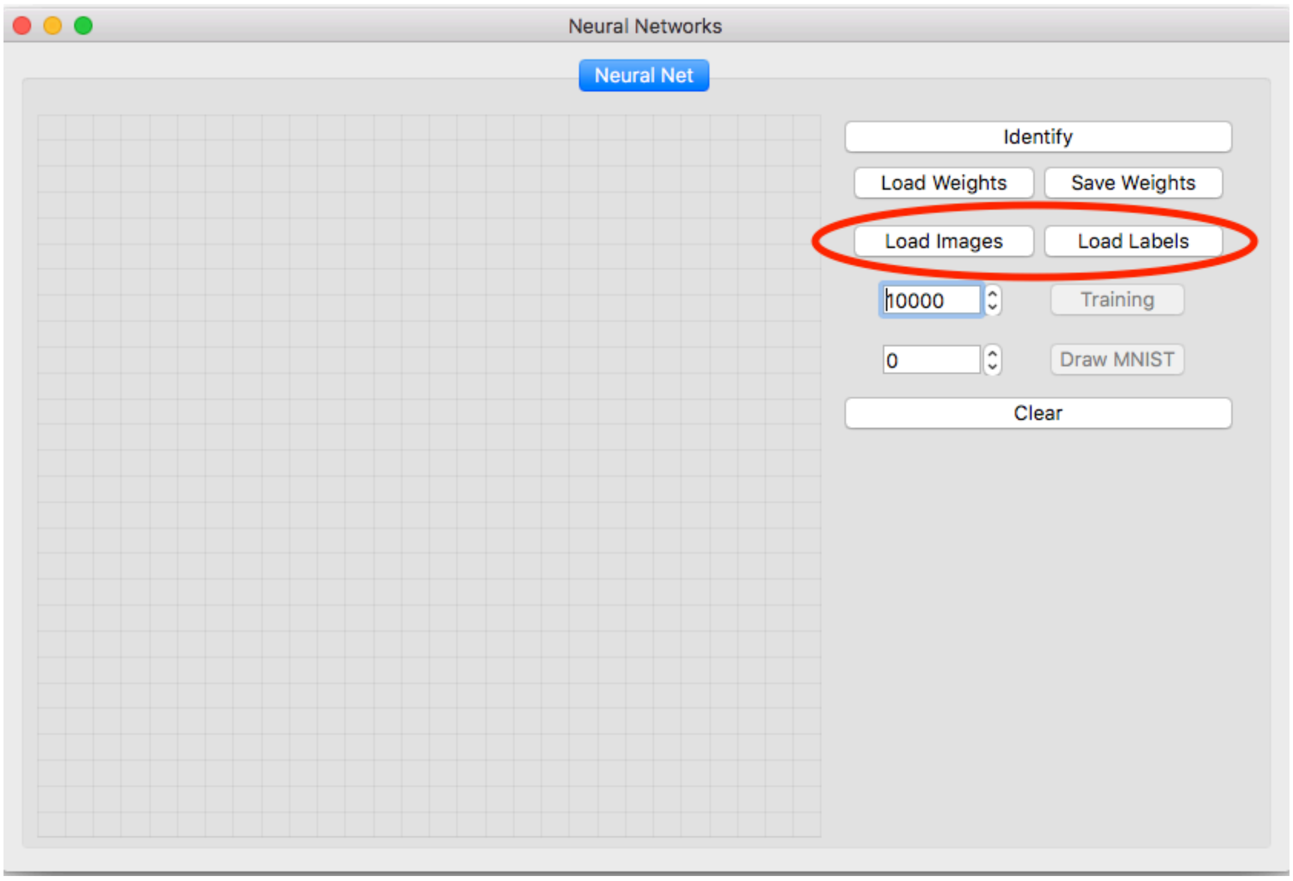
- Class Neuron with Perceptron parameters.
  - calculation output gradients
  - calculation hidden gradients
  - random weights
  - activation function
  - updating input weights

- Class Map with XY coordinates.
  - set and get value
  - brush tools
  - clear/delete map

- Class Net - include variables and functions for the Neural Network:
  - std::vector <Layer> of neurone layers - parameters of every perceptrons.
  - calculations of Weights, gradients, input/output values etc.
  - realisation of training Neural Network.

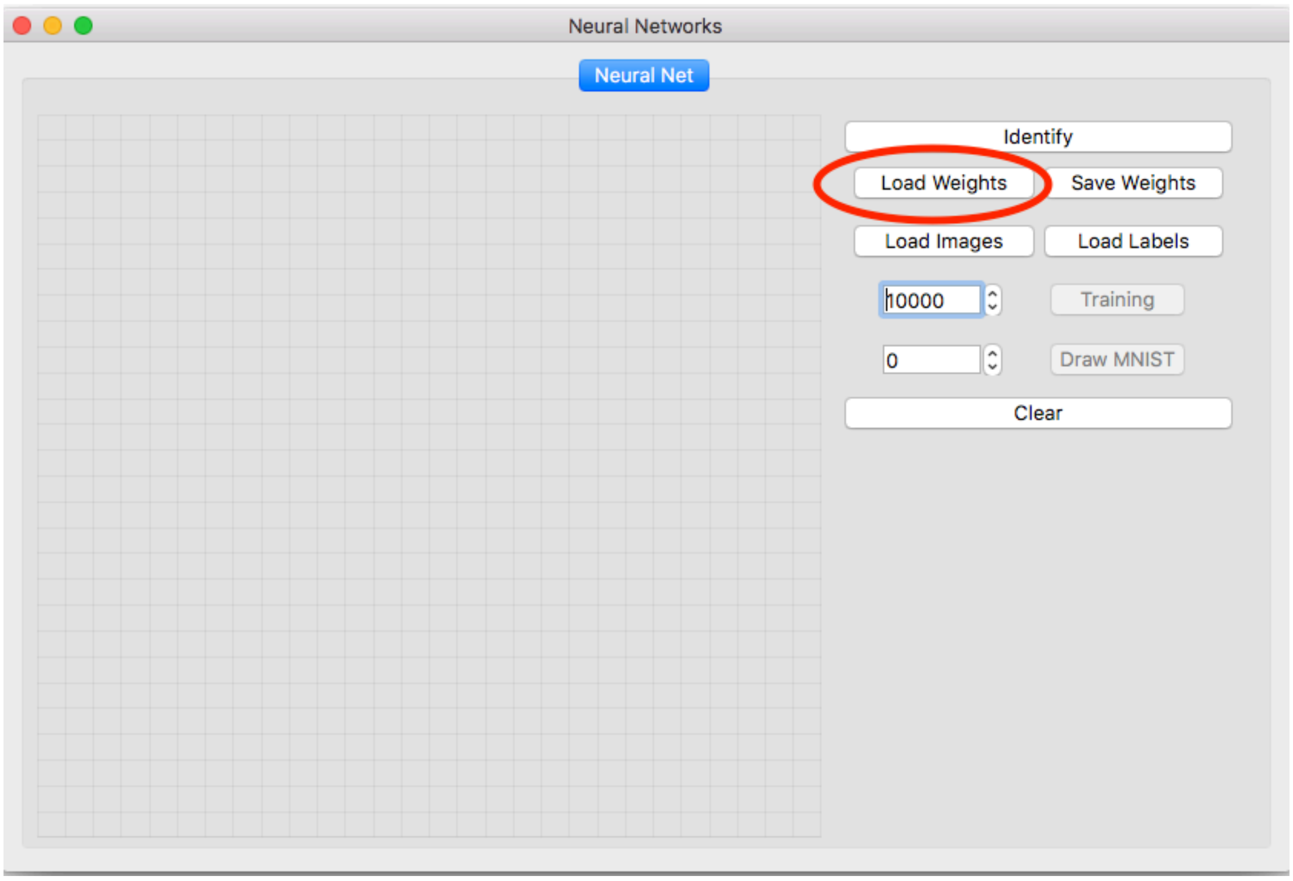
Instructions for use of the program:

1. Neural Net has two modes of operation - direct mode (**Identify** button) and training mode (**Training** button)
2. **Training** mode:
    1. you need to upload MNIST information - file with images and file with labels. Use buttons “Load Images” and “Load Labels” (both files are in the current folder)



    2. you can choose the number of training pictures (up to 60000) and if you want you can draw them with press the button “Draw MNIST”
    3. After training you can save the neural network weights by click the button “Save Weights”

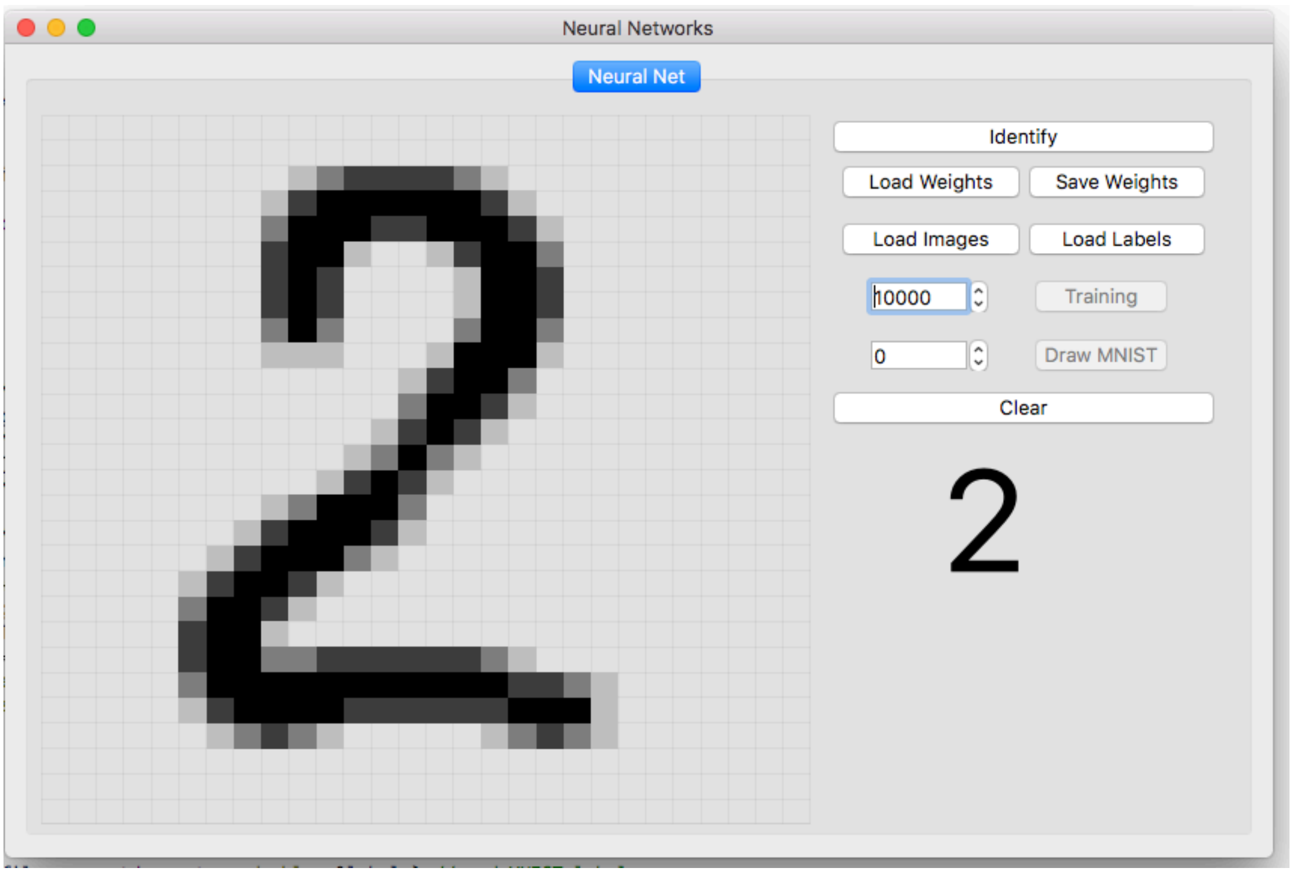
3. **Identify** mode:
   1. you need to have the neural network weights after training, which you need to load with the button (Load Weights)



   2. you draw a number on the map with the mouse and then click “Identify” and get the answer number under the button “Clear”
   3. you can clear the map with the button “Clear”

Notes:

1. Run once **Training** mode with 60000 training pictures and save the neural network weights — next time you can just upload weights and **Identify** your drawn number
2. Try to draw your numbers like MNIST data (you can see it with “Draw MNIST” button. Don’t forget to upload images and labels firstly) : use the central part of map and a large scale because the program is sensitive to position and to size of your image.


Neural Networks
Neural Net
Identify
Load Weights
Save Weights
Load Images
Load Labels
10000
Training
0
Draw MNIST
Clear
2