ISTA 421/521 Introduction to Machine Learning

Lecture 25: Autoencoders and Deep Learning

Leon Palafox

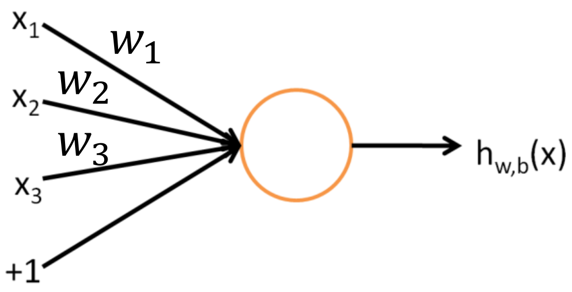
leonp@lpl.arizona.edu

4 December 2014

Perceptron

- We learned about the perceptron

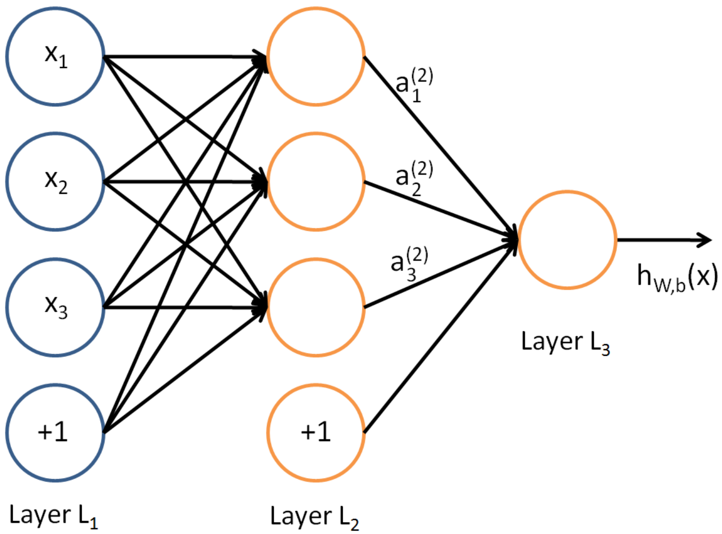
$$h_{w,b}(x) = f\left(\sum_{i=1}^{3} W_i x_i + b\right)$$



- Fundamental unit of NNs, look and smell like logistic regression.

Neural Network

- Is a set of fully connected perceptrons.



$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

Gradient Descent for NNs

- The cost function for the overall network is:

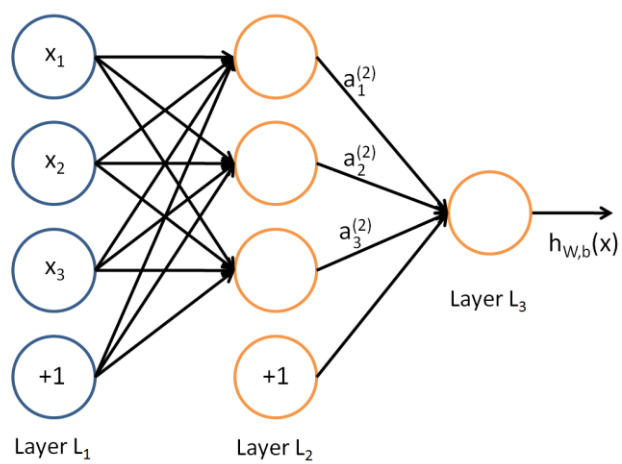
$$J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2$$

- Given the compact representation of the network:

$$\begin{aligned} z^{(2)} &= W^{(1)} x + b^{(1)} \\ a^{(2)} &= f(z^{(2)}) \\ z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\ h_{W,b}(x) &= a^{(3)} = f(z^{(3)}) \end{aligned}$$

In General

$$\begin{aligned} z^{(l+1)} &= W^{(l)} a^{(l)} + b^{(l)} \\ a^{(l+1)} &= f(z^{(l+1)}) \end{aligned}$$



Gradient Descent for NNs

- We want to compute an "error term" δ, that will measure the error of a node I in layer 'l'.

For the output layer:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

For the middle layers

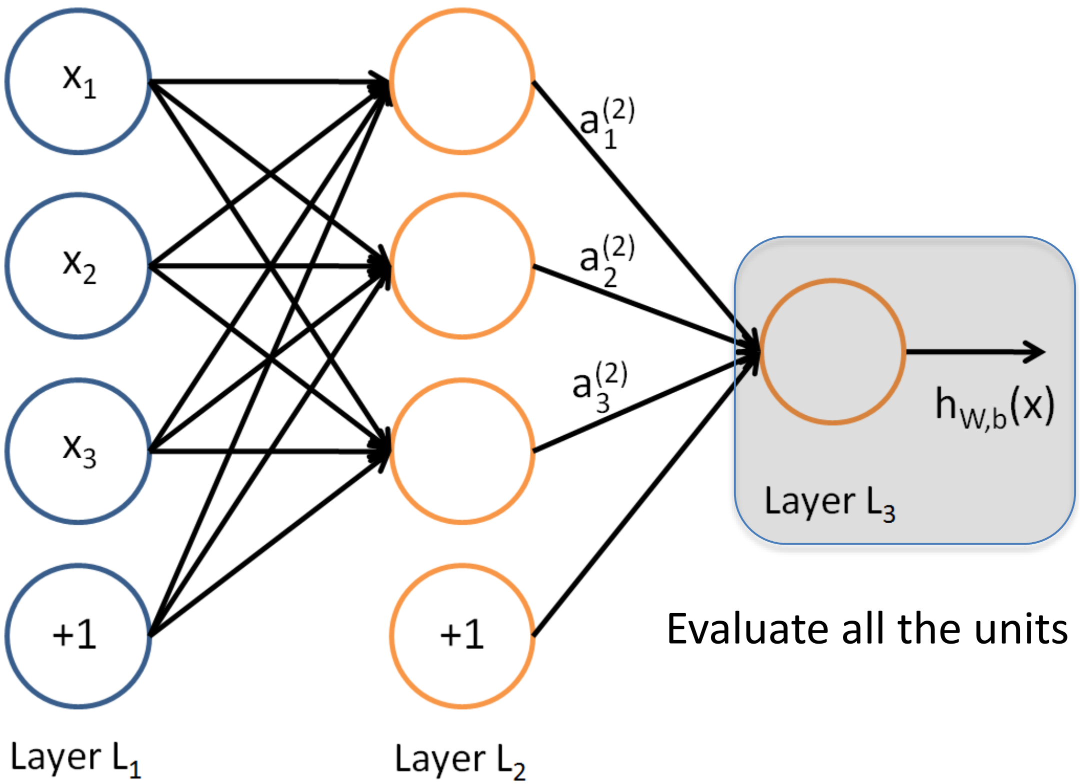
$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

Is a weighted average of all the errors related to this node

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

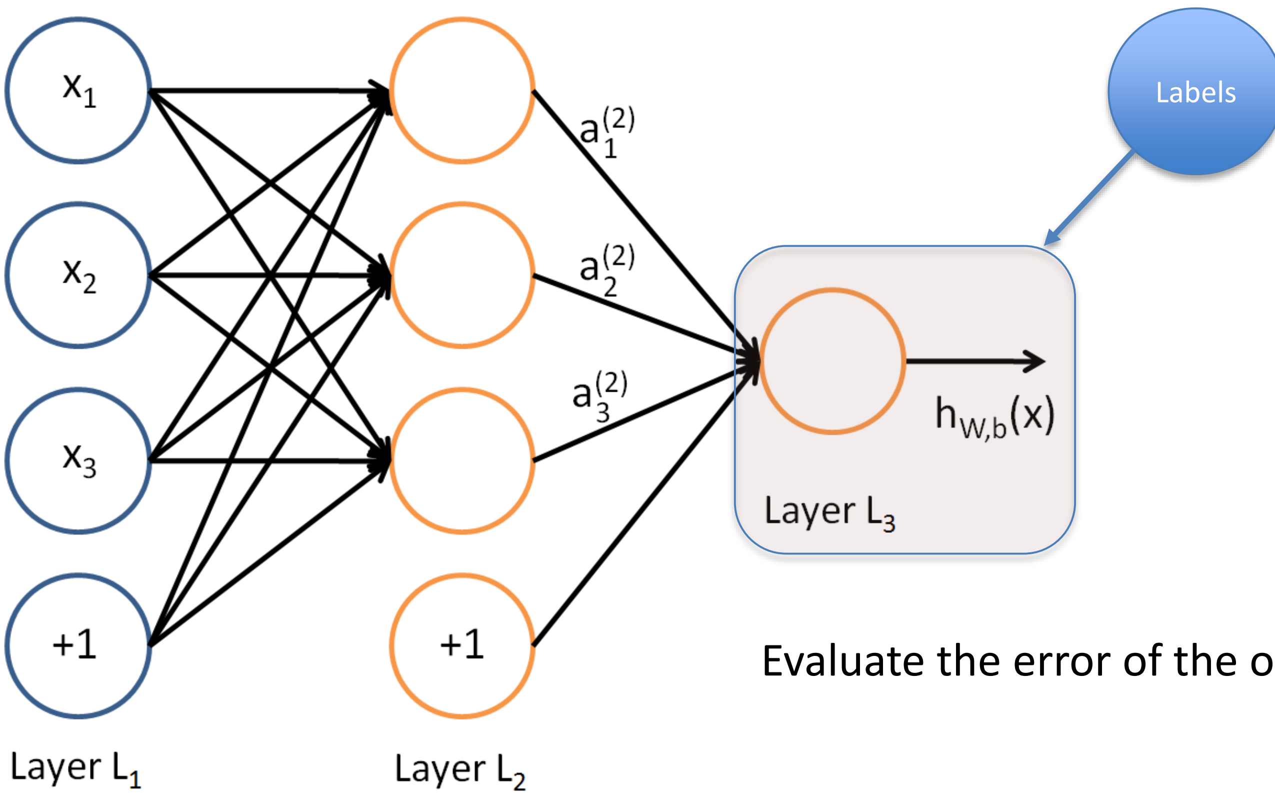
$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$

Feedforward-Backpropagation



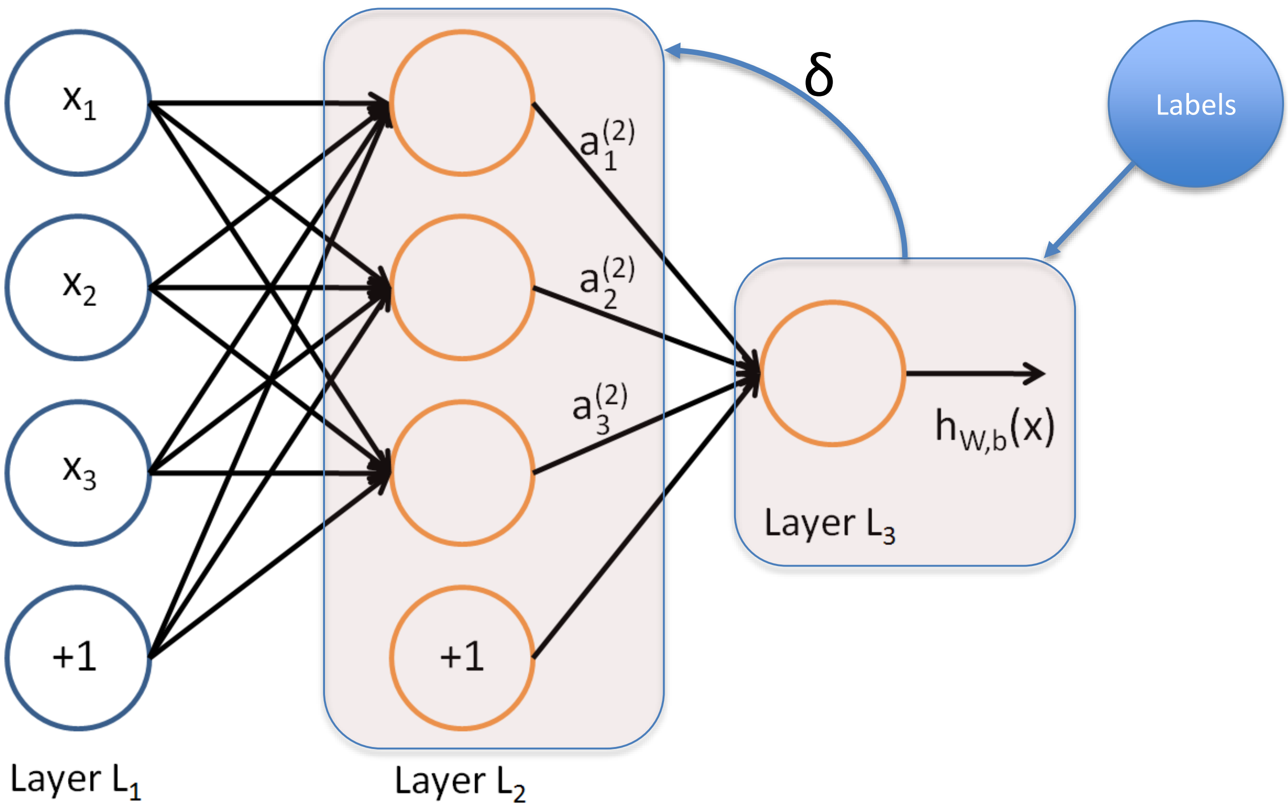
Evaluate all the units in Layer L3

Feedforward-Backpropagation



Evaluate the error of the output node (δ)

Feedforward-Backpropagation



Evaluate the errors of the middle layer nodes (δ)

How to initialize the weights

- In general is a bad idea to just use a range between 0 and 1.
  - Since there are many parameters, it can take a long time if we use a random initialization.
- For training, a good initialization range is:

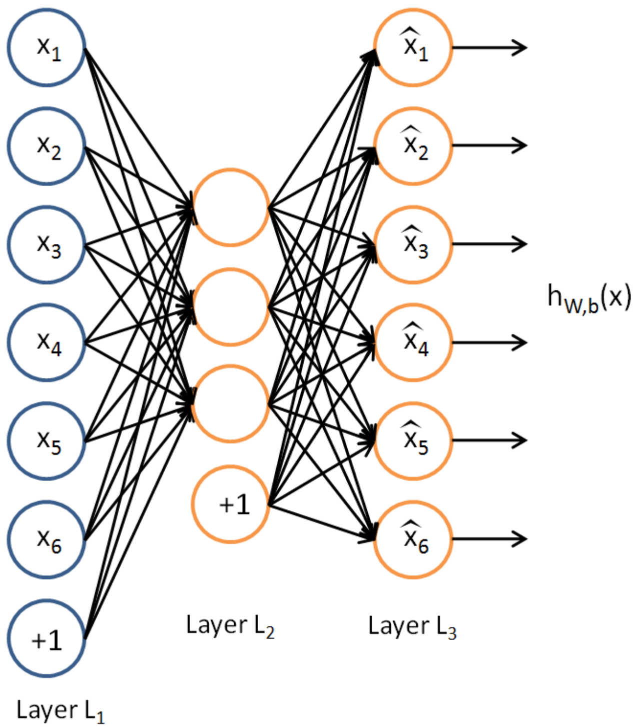
$$\left[-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}} + 1}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}} + 1}}\right]$$

The Autoencoder

- The autoencoder is one of many architectures of NNs.
- In the autoencoder we do not use the labels in the dataset.
- Is an unsupervised learning algorithm.
- We do not run things like Cross Validation or testing and training datasets.

Autoencoders

- An autoencoder is a NN where the output and the input are the same.



MNIST Dataset

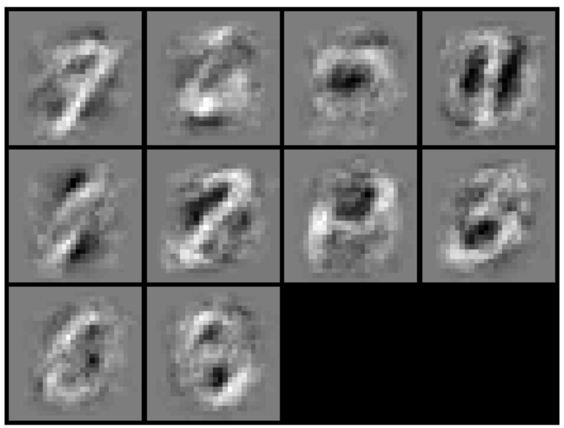
- Dataset of handwritten digits
- Has a training set of 60,000 examples, and a test set of 10,000 examples.
- Each digit is an 28x28 image (784 pixels)
- Each digit has a label that identifies which digit it represents. (9 labels)

MNIST Dataset

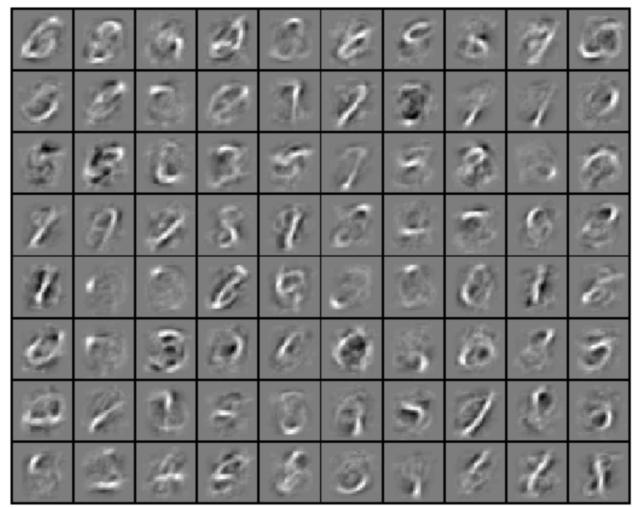
- http://yann.lecun.com/exdb/mnist/
- Is very good for learning:
  - All of the images have the same size (8x8)
  - It’s black and white, which means we do need to modify activation functions.
  - All the numbers have the same orientation.
- http://yann.lecun.com/exdb/lenet/scale.html

Autoencoder

- Why would I want both the input and the output to be the same.
- MNIST dataset as an example (28x28 input images)

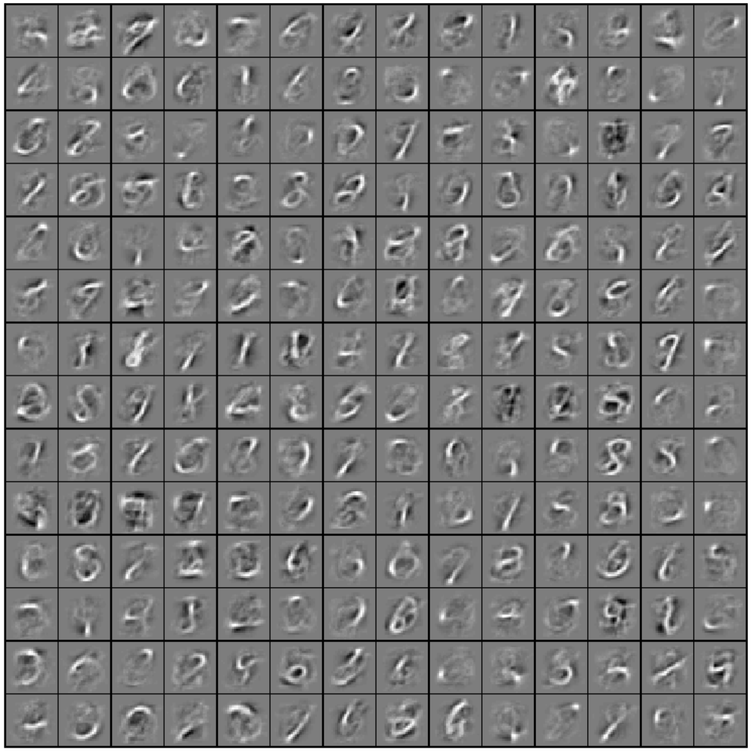


10 hidden units in Autoencoder

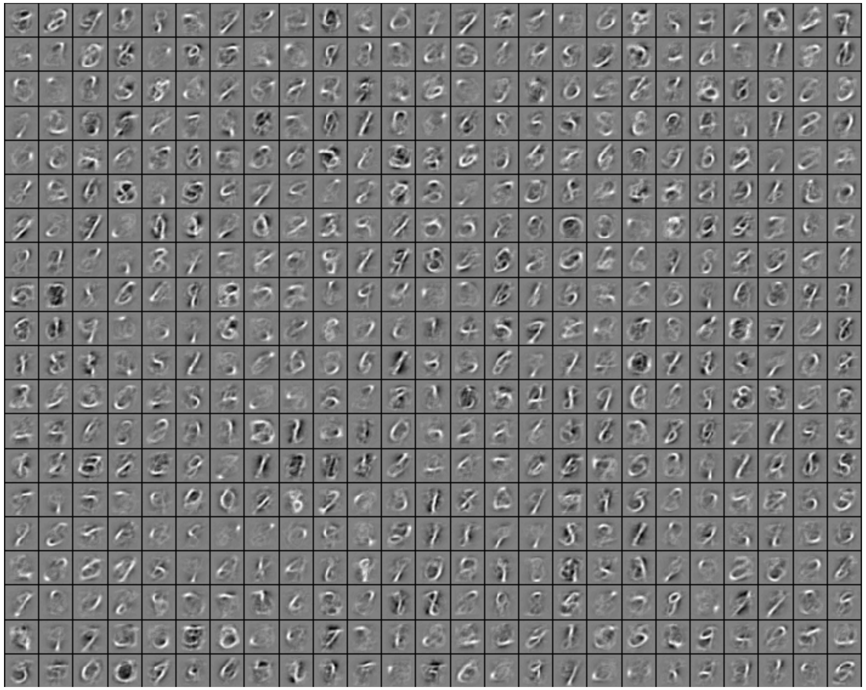


80 hidden units in Autoencoder

Autoencoder



196 hidden units in Autoencoder



500 hidden units in Autoencoder

Pseudocode

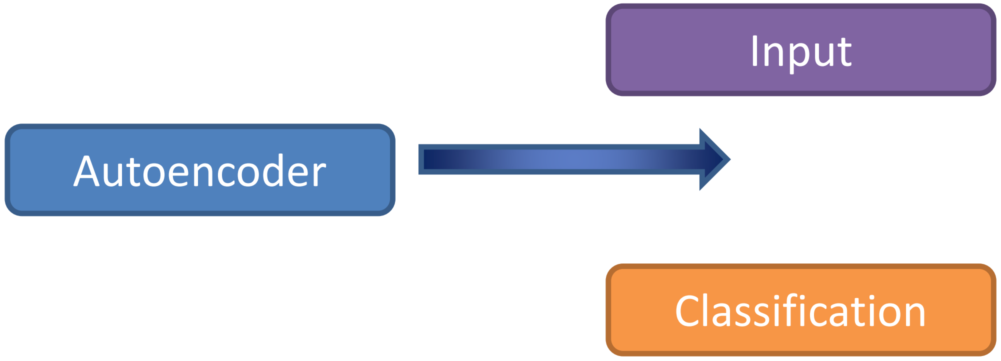
- For the MNSIT dataset:
  – Input Layer: 784 (28x28)
  – Hidden Layer: around 200 is a good number
  – Output layer: 784
  – In this case the label for each input is the same input.
    - Run Forward – Backpropagation, and Gradient Descent

Autoencoders & Deep Nets

- Remember we mentioned that initializing Neural Networks is tricky.
- The Vanishing gradient makes it hard to train large NNs.
- Autoencoders (and RBMs) are a solution to this problem.
- We use the autoencoder as an initializing step.

Autoencoder & Deep Nets

- If we train an autoencoder, and plug it in a NN then train. Things just work

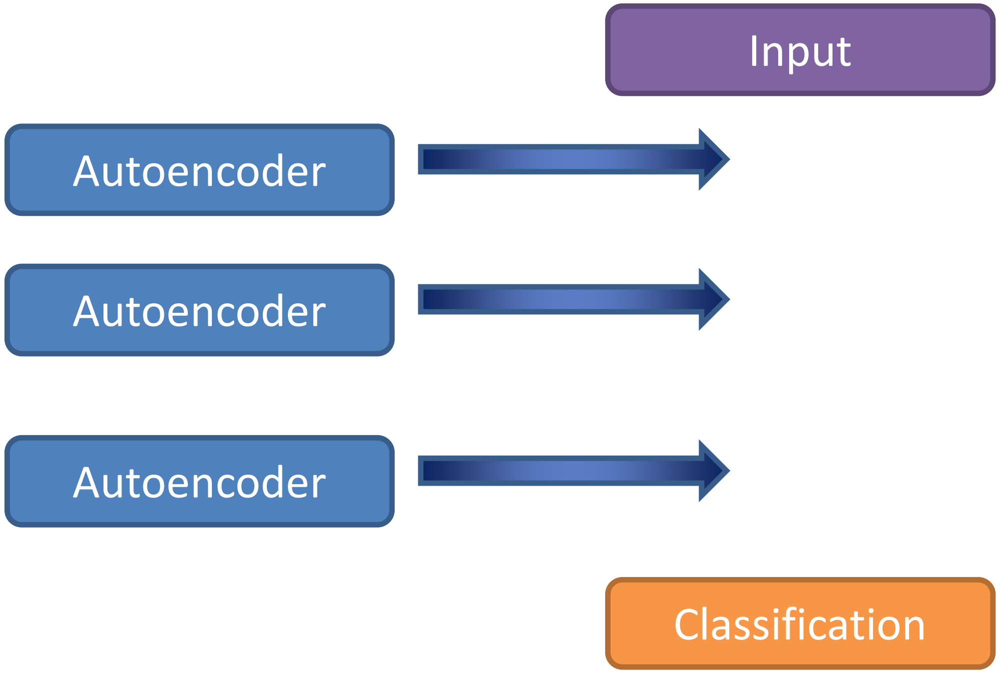


Deep Nets

- Train Autoencoder using a subset of MNSIT (10,000)
- Plug the autoencoder as the hidden layer of the NN.
- Do what we call a “fine tuning”, which is just a fast training to get the labeled part. (supervised)
- Now you can break ReCaptcha

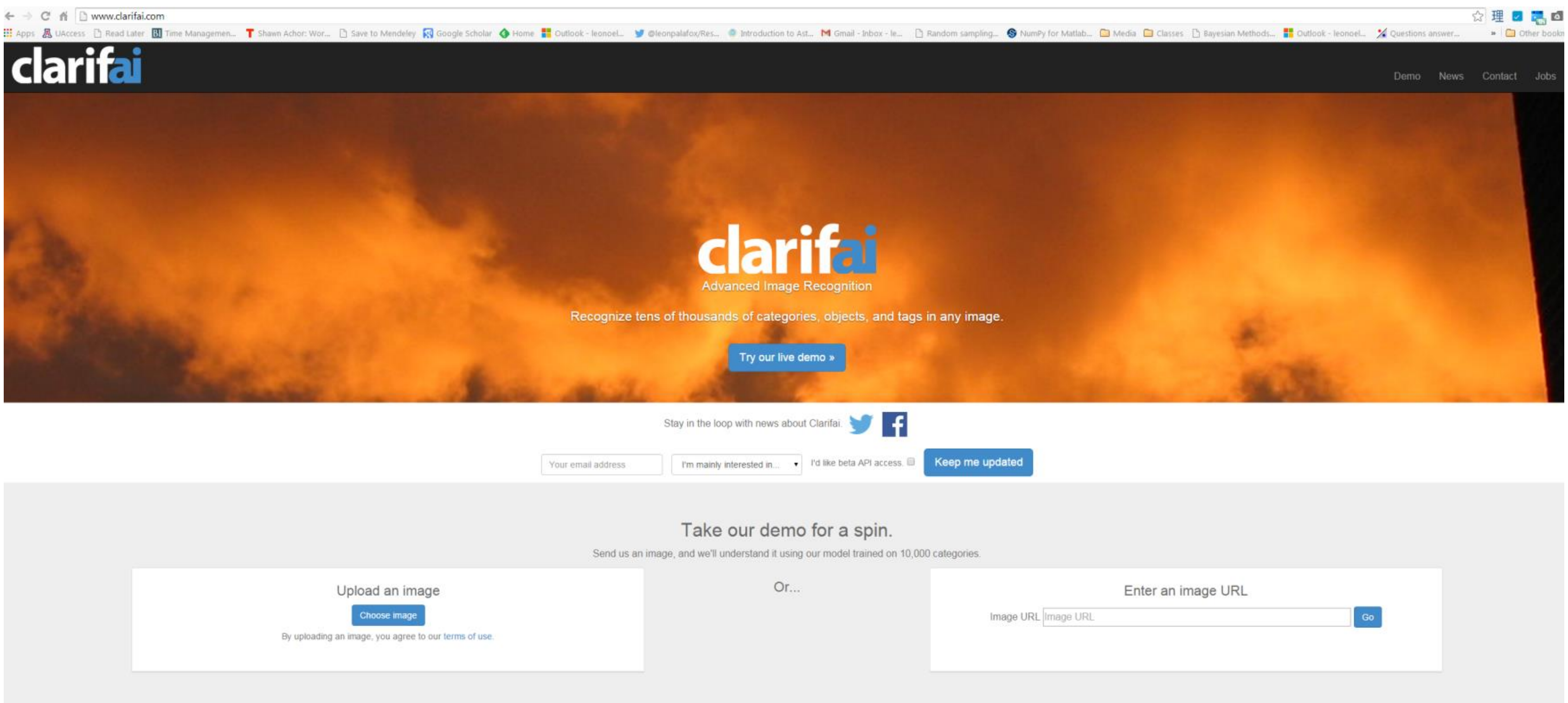
Deep Nets

- They are supposed to be deep so:



Demo

- http://www.clarifai.com/

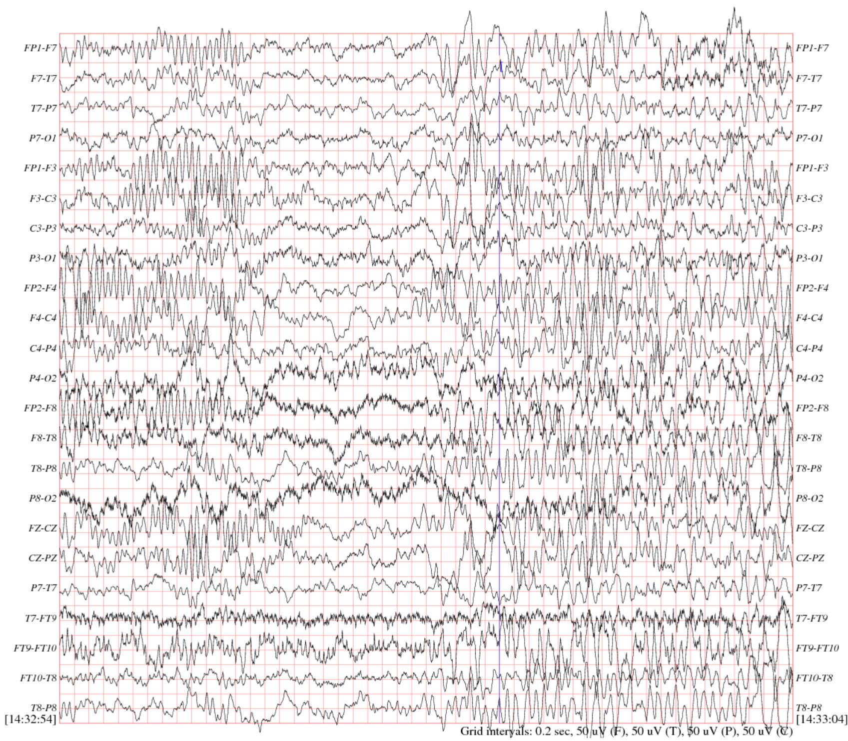


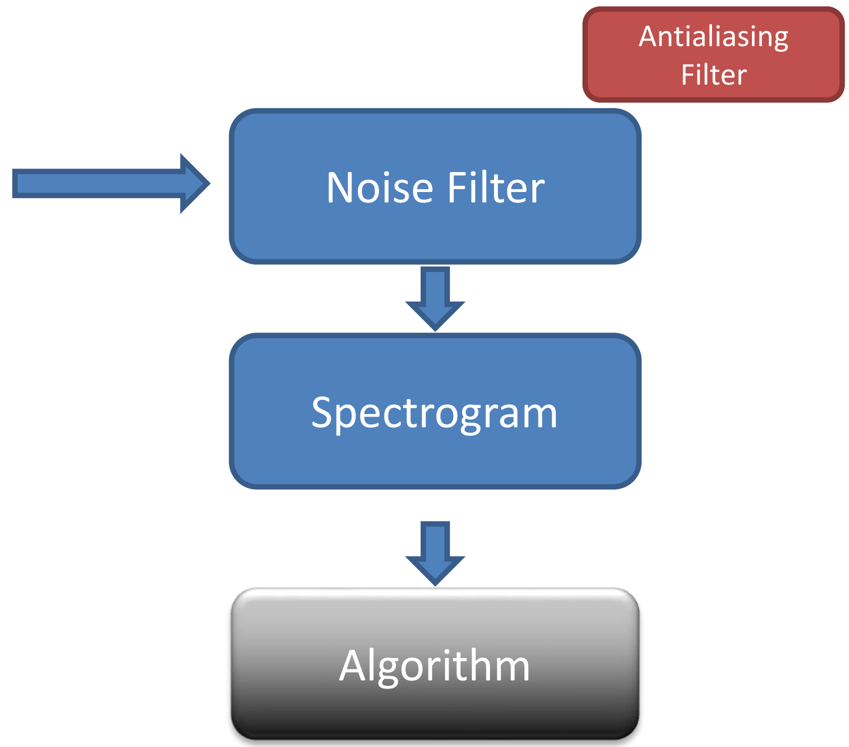
Important notes

- We are still not entirely sure why it works:
  - Some people say is because using this as a random start saves us much hassle.
  - Some say that this artificially moves us to a better search space.
- Using the autoencoder as a preprocessing step, has been proven to help us save steps when it comes to preprocessing algorithms.
- The autoencoder can find circles, edges, etc by itself.

Preprocessing Data

- It’s a pain, but is needed





Representation Learning

- Deep Nets are undergoing through a rebranding (again).
- One of the main problems in many fields, is that you have to go through many steps before you can use any algorithm.
- This is called preprocessing, and there are many sophisticated techniques to go about it.
- Deep Nets, via the autoencoder learn all of this transformations.
- This new area is being called representation learning.