Machine Learning applied to Planetary Sciences

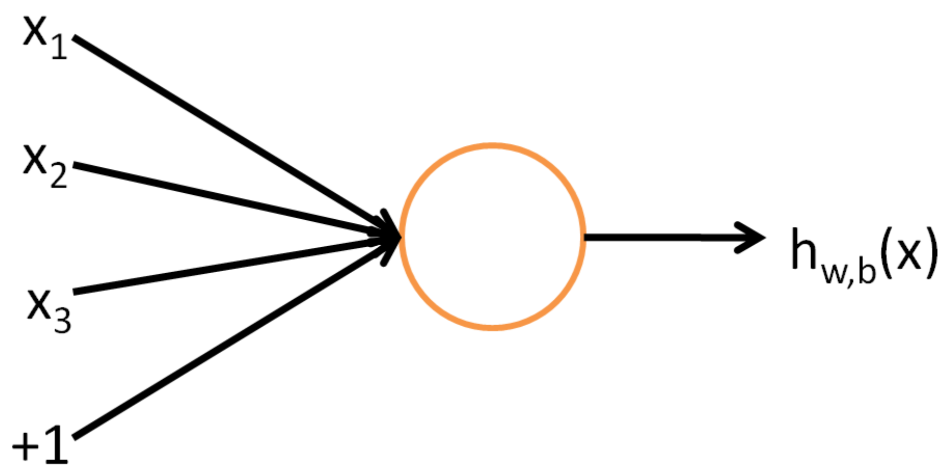
PTYS 595B/495B

Leon Palafox

https://leonpalafox.github.io/MLClass/

Perceptron

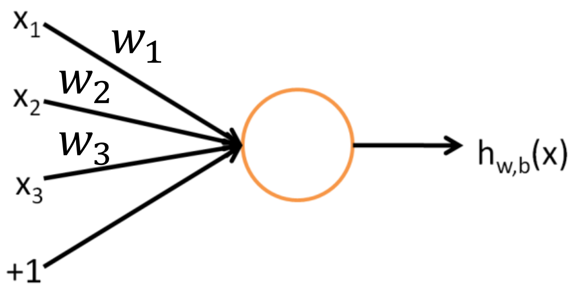
- Tries to mimic a real NN, since it has a nucleus that processes some inputs and give an output.



- $h_{w,b}(x)$ is a function of all the inputs, and is composed of two terms.

Perceptron

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



f is called the activation function, and it works as a way to discretize the outputs of the perceptron.

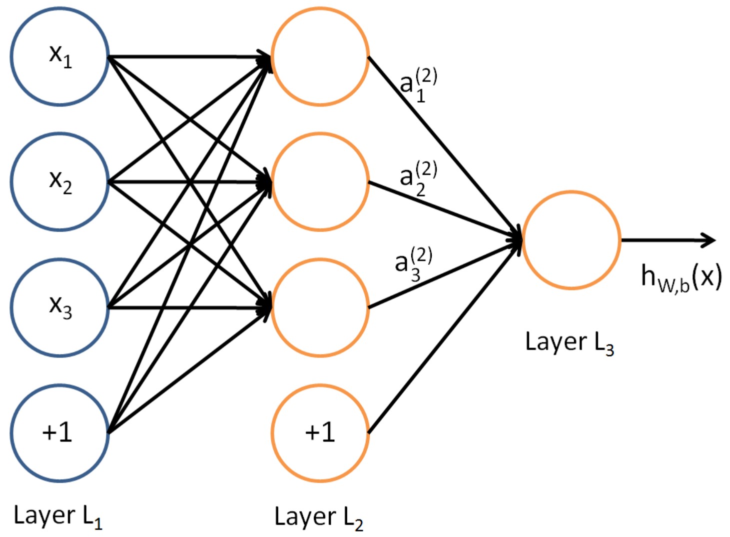
One of the most common activations functions is the sigmoid function:

$$f(z) = \frac{1}{1 + \exp(z)}$$

This looks very familiar

Neural Network

- Naturally, a NN is going to be a set of perceptrons interconnected within each other.

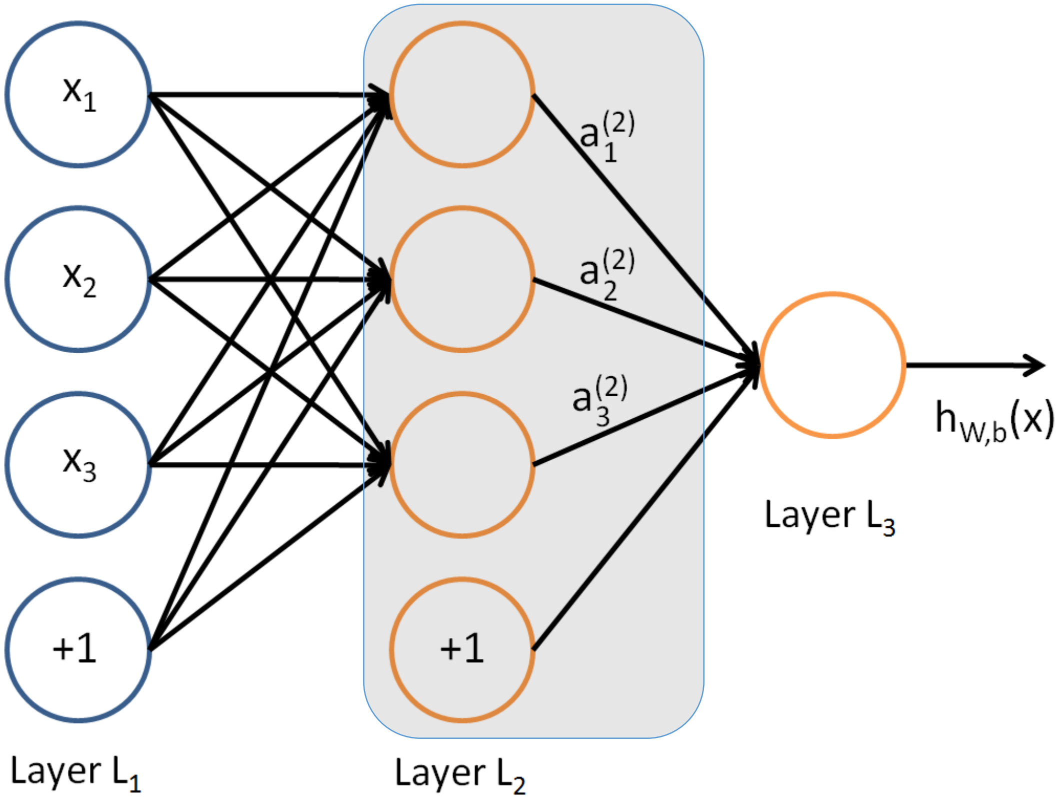


$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)}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

Training of Neural Networks

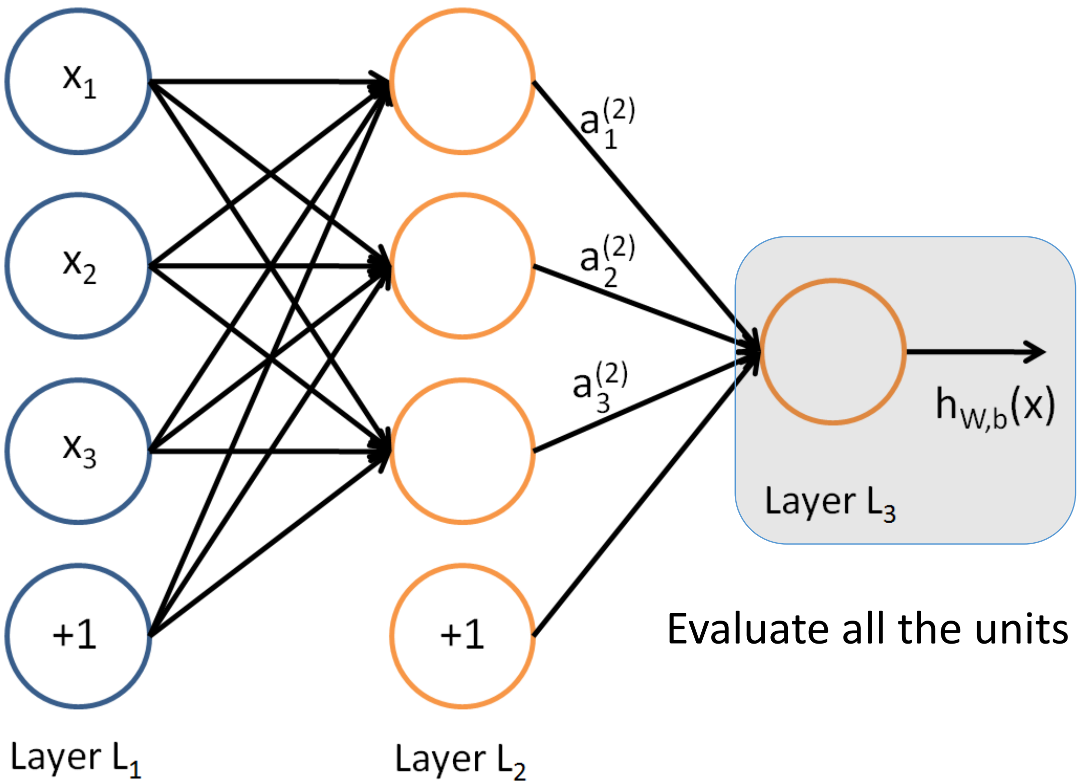
- The basic principle to train Neural Networks is Error Backpropagation.
  - We can find an error for a given input using the equations in the slide 11.
  - Next we go backwards in the network, and find the “share” of error each individual neuron has.
  - We calculate the derivative of this error to use Gradient Based techniques, like Gradient Descent.

Feedforward-Backpropagation



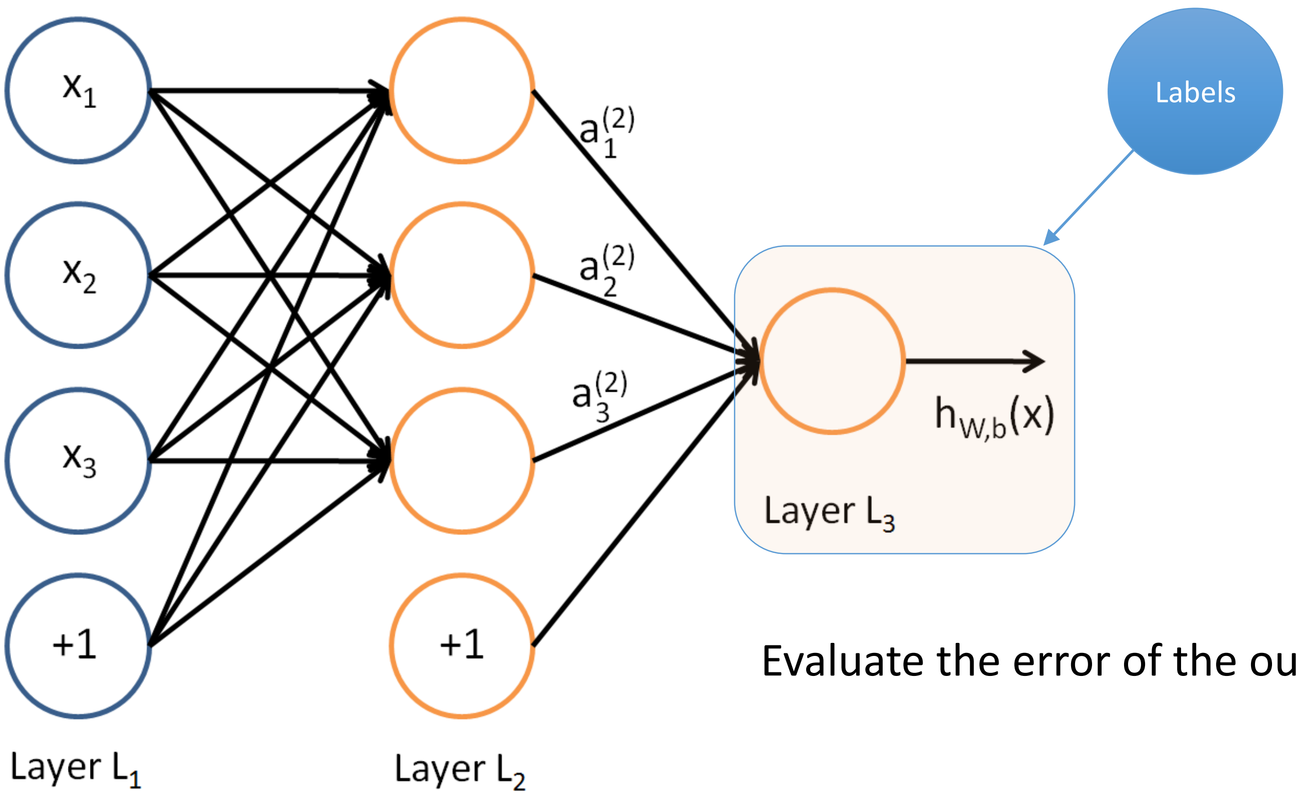
Evaluate all the units in Layer L2

Feedforward-Backpropagation



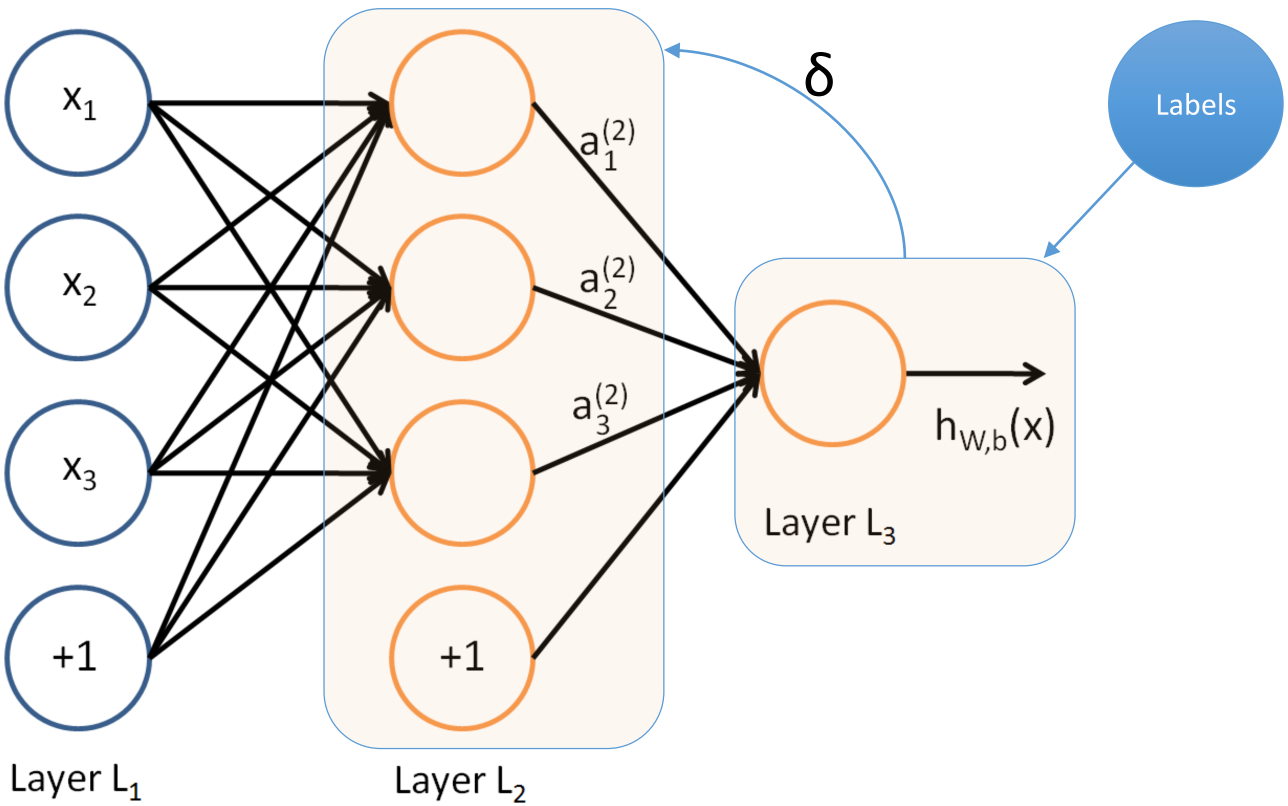
Evaluate all the units in Layer L3

Feedforward-Backpropagation



Evaluate the error of the output node (δ)

Feedforward-Backpropagation



Evaluate the errors of the middle layer nodes (δ)

Gradient Descent

- Given a cost function

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- m is the number of examples and h some function of parameter θ.
- Gradient descent updates the parameter in steps:

$$\theta_j = \theta_j - \alpha\frac{\delta}{\delta\theta_j}J(\theta)$$

Gradient Descent for NNs

- The cost function for the overall network is:

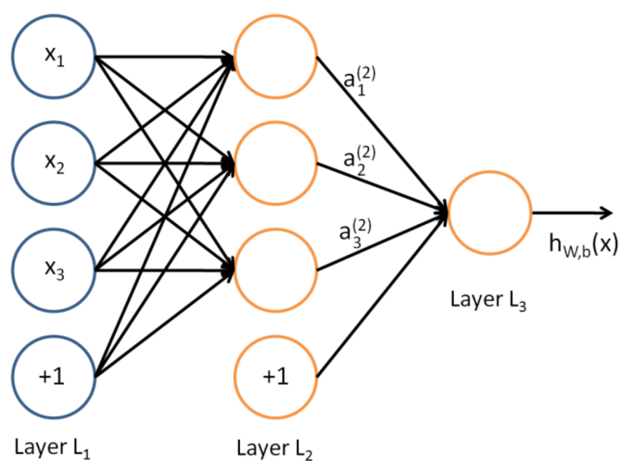
$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

- 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

- Gradient Descent: (Per Layer)

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

With:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W,b;x^{(i)},y^{(i)})$$

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

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

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

Rule of thumb

- 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:

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

Problems of NNs

- We need to answer two questions:
  - How many layers are enough to solve a problem?
  - How many hidden units should we use per layer?
- As you can imagine, training complexity increases as we increase hidden units.
  - This can be reduced by avoiding a full interconnection.
- The elephant in the room is called “Vanishing Gradient”

Vanishing Gradient

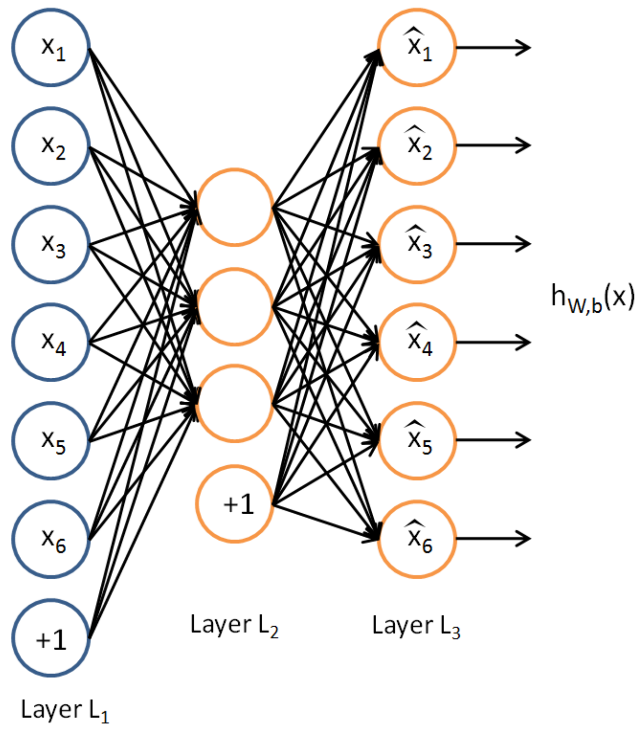
- A problem of NNs, is that our small δs, will become even smaller as we go back in our layers.
- If we have many layers, we are going to end up with really small gradients.
- This will show as negligible updates in the gradient descent equations.
- For many years, before 2006, this was the main reason few people used classic NNs.
  - What is the point of having the power that comes from many layers, if we cannot train it properly anyways?
  - The solution: Pre-training of the individual layers.

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 testing and training datasets.

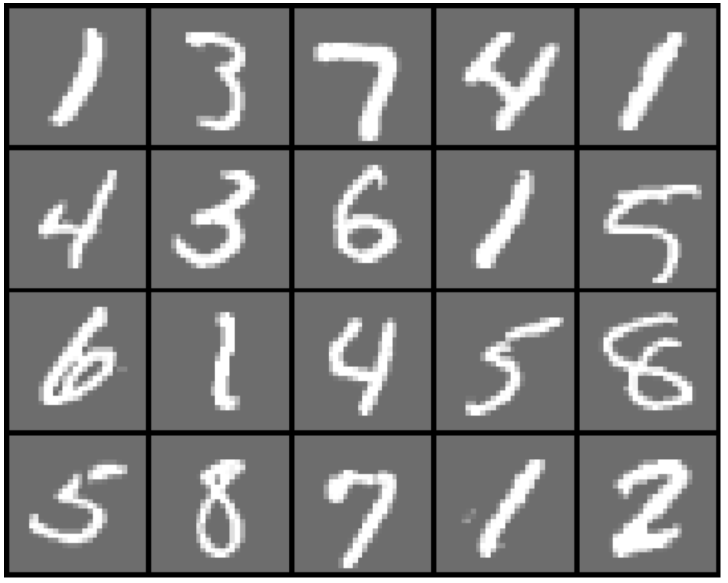
Autoencoders

- 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)

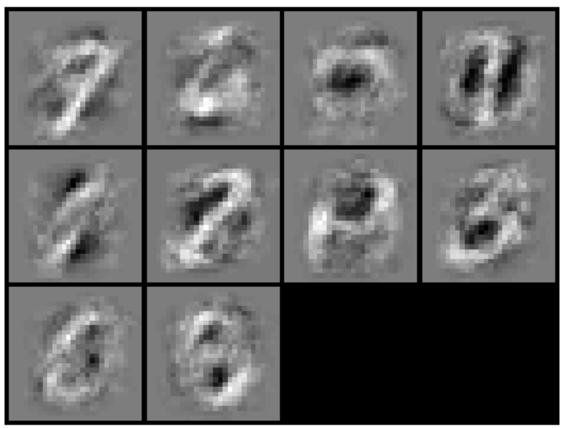


MNIST Dataset

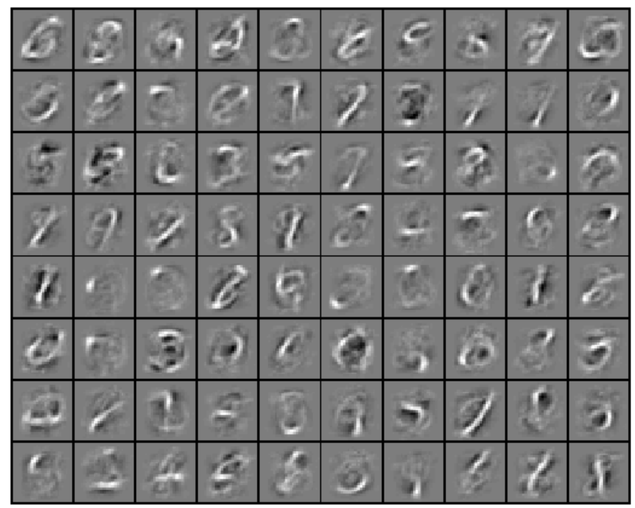
- http://yann.lecun.com/exdb/mnist/
- Is very good for learning:
  - All of the images have the same size (8x8)
  - It’s black and white, which means we do need to modify activation functions.
  - All the numbers have the same orientation.
- http://yann.lecun.com/exdb/lenet/scale.html

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



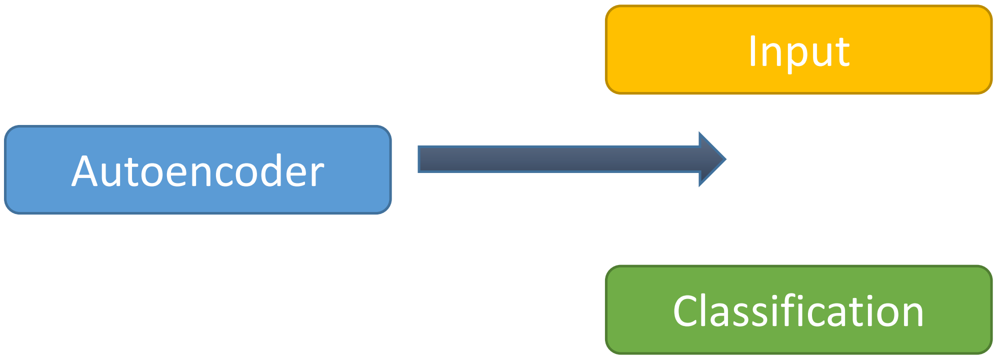
80 hidden units in Autoencoder

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: