Deep Neural Networks

For the UCI Samueli School of Engineering

Saketh Karumuri

UCI Samueli School of Engineering

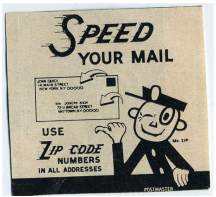
May 6, 2025

Outline

What Are Deep Neural Networks?

How Do You Solve Problems With DNNs

Mr.Zip Learns to Read

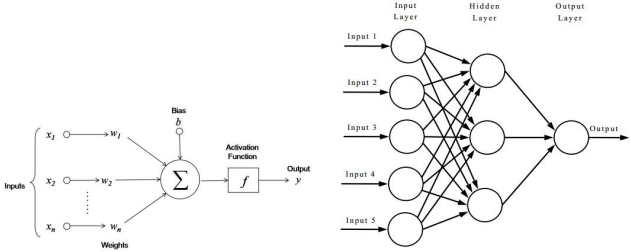


You work for the post office during the dawn of character recognition.
How can you make an algorithm that can go from an image to a zip code?

Mr.Zip Learns to Read

Thankfully, you paid enough attention in bio to learn that our neurons can represent anything we can think of (literally)
Lets make a ”brain” out of these neurons. Now we can ”teach” it by showing it what numbers look like and seeing how the neurons are activated.
We’ve just come up with neural networks!

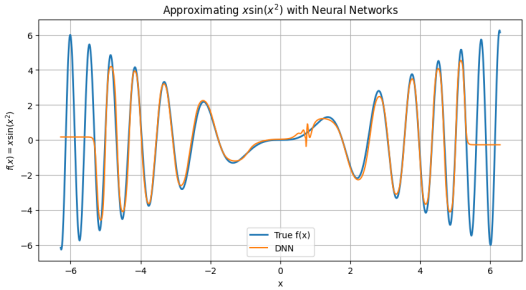
What Are Neural Networks?



Neural Networks

- inspired by biological neurons
- have input stimulus, activation function and output can be used as the networks output or as the input of other neuron(s)
- built up of 1 input layer, a number of hidden layers, and 1 output layer.

Neural Networks are function approximators!

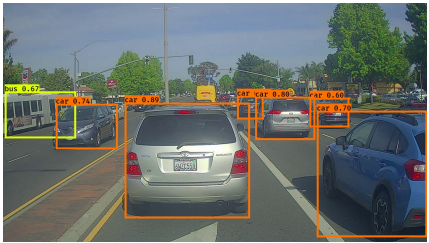


Training is the process of finding which neurons need to be connected to eachother and what level of stimulus they activate at to best approximate the desired function. Neural Networks with one arbitrarily sized hidden layer don't need any more than one hidden layer to represent any function.

Why Approxmiate Functions

Not all functions we work with can be easily represented analytically so for us to do useful things in reasonable time frames we need to represent them. Some examples of functions that would be nice to have analytic solutions to are:

- What will the weather tomorrow be as a function of climate data gathered today
- What number is this image's pixel data trying to represent?
- How should I steer to maximize the likely hood that the car does not crash?

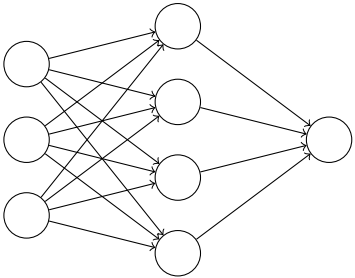


What Are Deep Neural Networks (DNNs)?

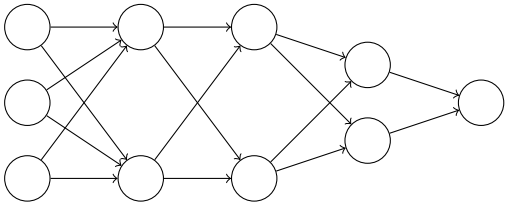
What Are Deep Neural Networks (DNNs)

Deep Neural Networks have more than one Hidden Layer.

Shallow Network

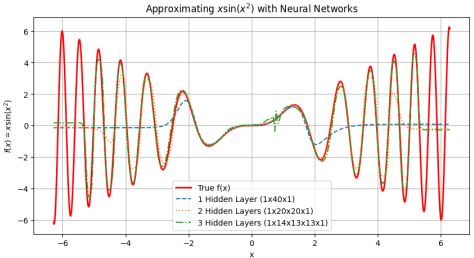


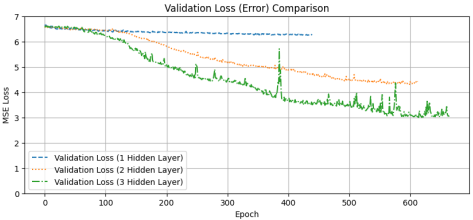
Deep Network



Why Do we Need Depth?

Depth generally improves performance.





But *Why* Do we Need Depth?

For a network with

- m - inputs
- L - layers
- $n > m$ neurons

With ReLU activation, this network can represent a piecewise linear function with a lower bound on the number of segments

$$\Omega\left(\left(\frac{n}{m}\right)^{L-1} n^m\right)$$

https://arxiv.org/abs/1312.6098
http://papers.nips.cc/paper/5422-on-the-number-of-linear-regions-of-deep-neural-networks

We don’t understand how these work

There are so many parameters that it being able to fit to any training data makes sense. But doesn’t make sense is how it’s able to actually learn when given real data if it’s overfitting to random noise?

What does using a DNN to solve a problem look like?

Our Problem

Our Problem

Two robots want to cross a landscape q. q is a string of terrain types in the set of terrain types $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_p\}$. We have Terrain resistances for robot i: $c_i(t)$ where $t \in \mathcal{T}$ Our decision variable is σ the mode indicator.

$$\sigma : \mathcal{T}^N \to \mathcal{M}^N$$

- μ_i - mass of robot i
- $M_i(m)$ - Effective Mass function
- where m is the collaborative mode

Our effective mass functions are defined as:

$$M_1(m) = \begin{cases} \mu_1 & \text{if } m = 0 \\ \mu_1 + \mu_2 & \text{if } m = 1 \\ 0 & \text{if } m = 2 \end{cases}, \quad M_2(m) = \begin{cases} \mu_2 & \text{if } m = 0 \\ 0 & \text{if } m = 1 \\ \mu_1 + \mu_2 & \text{if } m = 2 \end{cases} \tag{1}$$

Our Problem

The energy consumed by each robot for a given assignment string is

$$E_i(\sigma) = \sum_{k=1}^{N} [c_i(q_k)M_i(\sigma_k) + c_{si}\delta(\sigma_{k-1}, \sigma_k)] \tag{2}$$

$$\delta(m_i, m_j) = \begin{cases} 1 & \text{if } m_i \neq m_j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Where $\sigma_0 = 0$,
Finally, we want:

$$\min_{\sigma} \left\| \begin{bmatrix} E_1(\sigma) \\ E_2(\sigma) \end{bmatrix} \right\|_p \tag{4}$$

The Traditional Approach...

Usually we would use something called a Mixed Integer Linear Program which uses decision variables to find the decision variable's values for the minimum (or maximum) value of the function.

$$|\sigma| \propto O(|\mathcal{T}|^N) \tag{5}$$

Since the action chosen with mixed integer program is a decision variable, the number of decision variables we have scales with the length of terrain.

Why Use a DNN here?

Neural Networks can estimate a function that gives us the optimal assignment. Neural networks scale at a much more reasonable rate as we increase the number of layers (however it remains to be seen how much training time increases as a result of this as well). We can effectively design a network to be exactly as big (and computationally expensive) as we want.

Designing our Network

Rules of Thumb

- Generally DNNs use no more than 3 layers
- Bigger hidden layers than input layers
- Aside from this it's a lot of experimenting on your specific problem

We can use a DNN by *embedding* our inputs to get numerical values to give as inputs to the NN
Similary for our output we need to do the opposite to get the network's estimated solution.

Future Work

For this neural network estimation to be useful we need to estimate a system with more robots or use a longer terrain to be crossed.
In both cases this causes a big increase in computing the optimal solution: $|\sigma| \propto O(|\mathcal{T}|^N)$ I have reworked the problem to be something that is a Model Planning Control problem that's solved online. I still want to find at what point it becomes reasonable to use the DNN to approximate the solution.