Welcome Back!

...

Machine Learning Workshop
Round 2: Neural Nets and Stuff

This Week:

Warm Up

- Review and Alternative Algorithms
- Introduction to more complex problems

Main Content

- The problem of classification
- The neural network approach
- Applying machine learning concepts to neural networks
- New optimization methods

Last Week:

Talked about machine learning

- Create models which can solve complex problems
- Using machine learning algorithms to optimize models for us

Pros and Cons

Optimization Methods

- Linear Regression
- Gradient Descent

Review

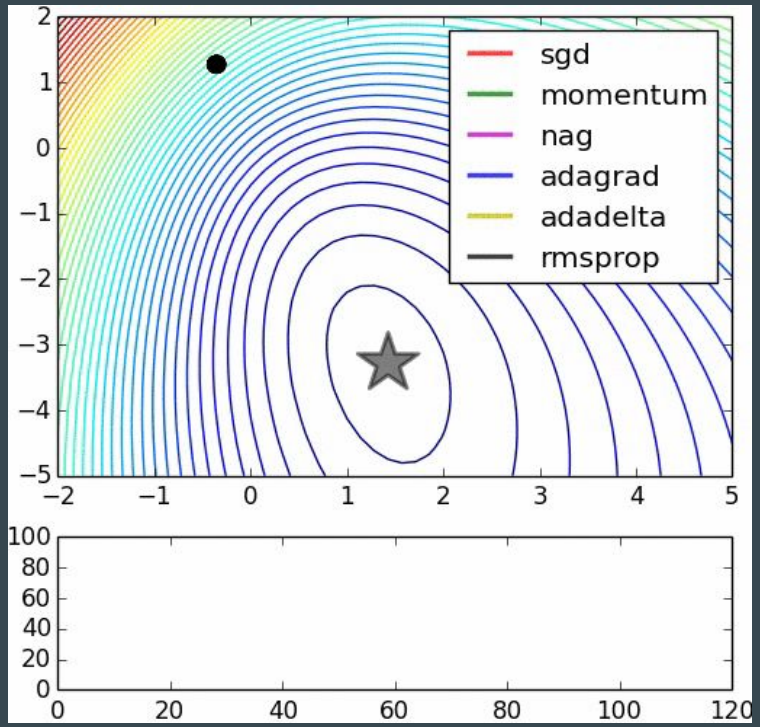
Supervised Learning: Prediction & Classification

1. Create a model whose behaviour depends on their internal parameters.
2. Create an error function (continuous) to define the success/failure of the model.
3. “Teach” the model, give it inputs for which expected output is known. Modify the model in order to reduce error.

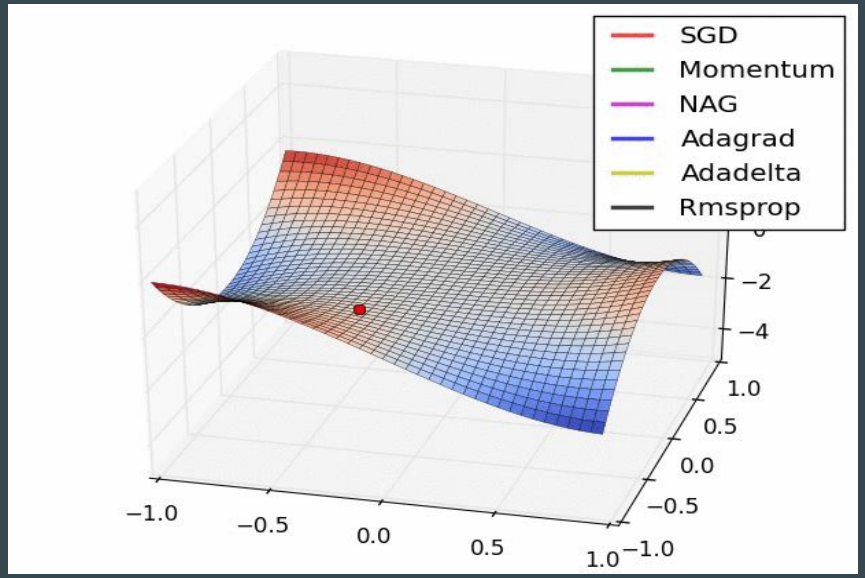
Optimization: The Gradient Descent Strategy

1. Start at an arbitrary point
2. Calculate the gradient at current point
3. Move slightly in the direction opposite the gradient (down the slope)
4. Repeat steps 2 & 3 until satisfied.

Alternative Algorithms

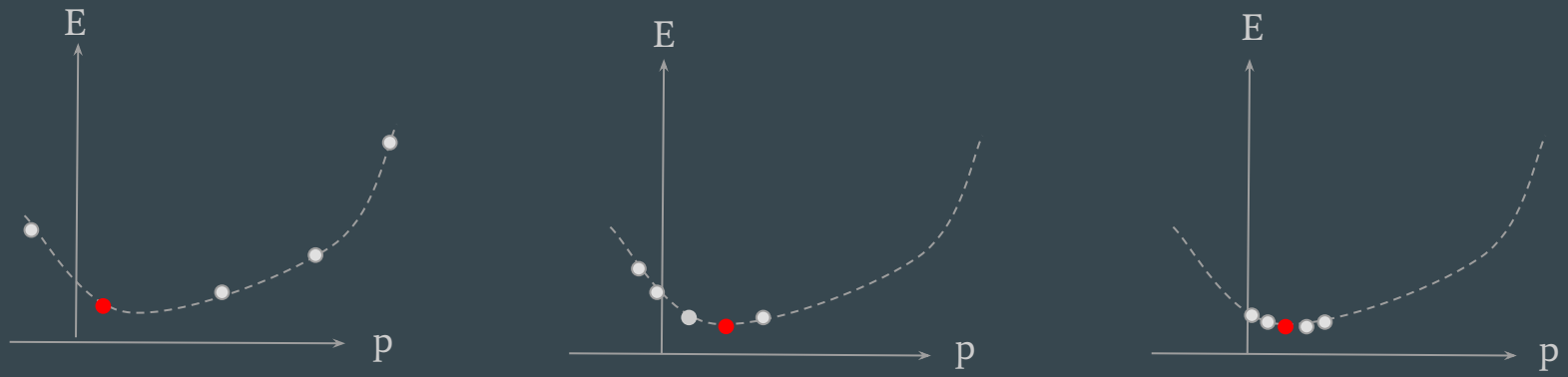


- Many similar alternatives to gradient descent
- Use slightly different kinematics to achieve result

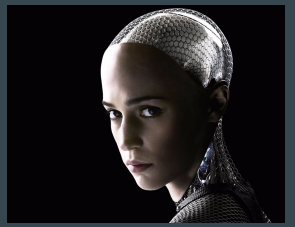


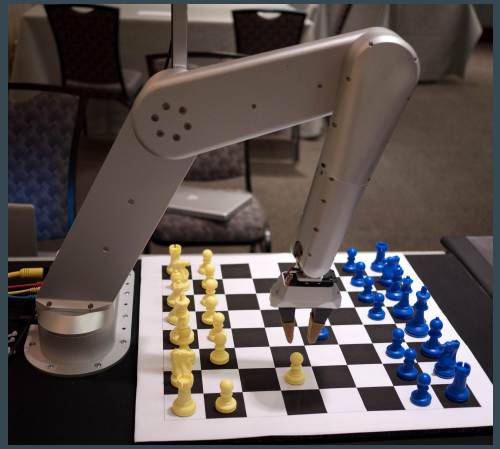
Alternative Algorithms: Successive Approximation

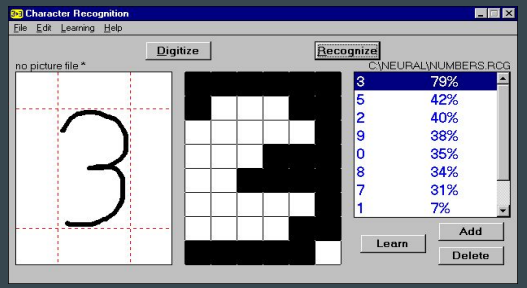
- Slightly different approach than gradient descent
- Often better at finding global minima in a given parameter range

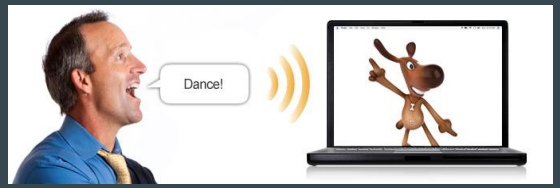


More Complex Problems







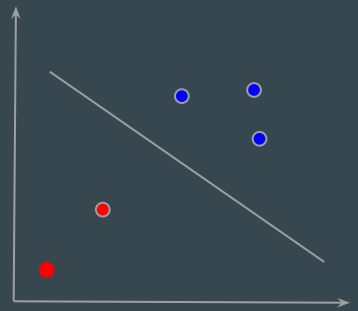


More Complex Models

- Linear/Logistic Regression
- Decision Trees
- SVM
- Artificial Neural Networks
- Bayesian Networks
- Nearest Neighbour

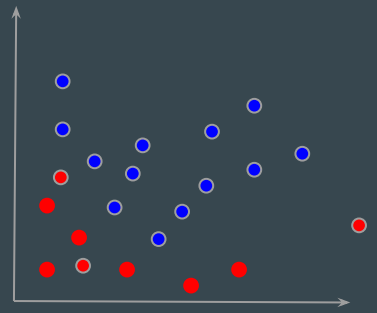
A Closer look at Classification

If we look at a set of plotted points, the problem of classification can be thought of as drawing a line or curve such that all points of one type lie on one side of the line.



A Closer look at Classification

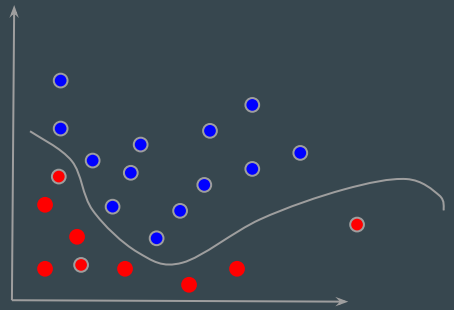
For more complex data, this problem becomes harder



A Closer look at Classification

Recall though, that according to Taylor Theorem, you can represent any function as an infinite sum polynomial.

$Y = A + Bx + Cx^2 + ...$



Which is nice, especially since this holds even for multivariate problems

A Closer look at Classification

Summary:

For any classification problem, we can try to approximate the true value of this function by combining products of the inputs. This approximation gets better and better as we consider higher and higher order products. All we have to do is tune the coefficients.

Problem:

As our number of inputs expands, the number of products we must consider grows exponentially

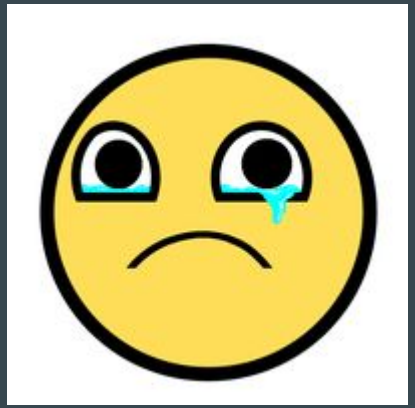
A Closer look at Classification

Example: 3 Inputs: x, y, & z

1st Order	x, y, z	3
2nd Order	xy, xz, yz, xx, yy, zz	6 + 3 = 9
3rd Order	xxx, xxy, xxz, yyy, yyz yyx, zzz, zzx, zzy, xyz	10 + 9 = 19 (I'm missing some? Or nah?)
...	...	...
100th Order	Omg how, so many	Way over 9000

Conventional Methods: The Struggle

As number of parameters go higher and higher, and order of approximation gets higher and higher, conventional methods of computation become too expensive. There are simply too many terms to optimize.

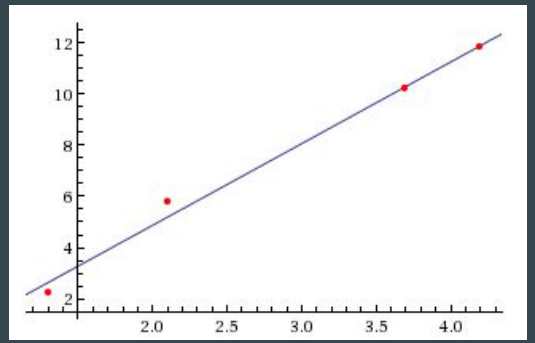


Underfitting vs. Overfitting

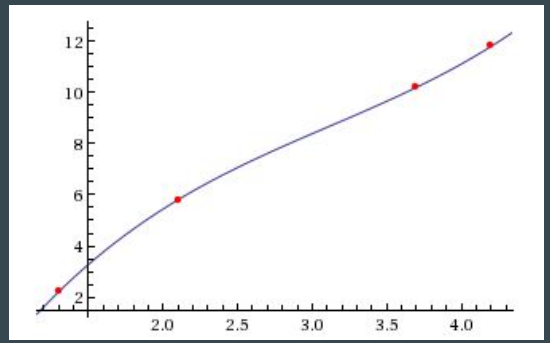
The goal of optimization is to minimize error. However in certain cases our optimization strategy actually finds ways to reduce error that are unfavourable. Specifically, sometimes the model changes in a way that doesn't properly represent the real world.

Example:

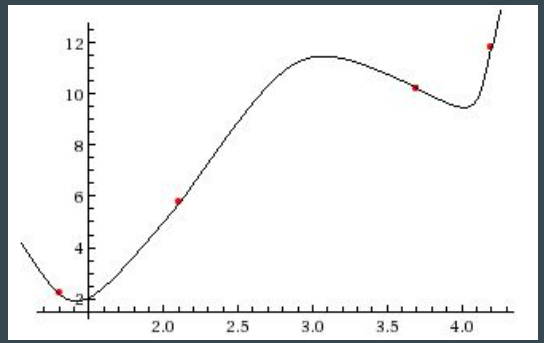
Linear Fit



Cubic Fit



Arbitrary Higher Order Fit



Underfitting vs. Overfitting: The Takeaway

1. Overfitting is when we optimize the model until it no longer represents the real world.
2. In order to prevent overfitting, we need to prevent the optimization from taking advantage of features that are specific to the *training data* and **not** to the *real world*. This can be done in a few ways:

    - We can guess at the nature of the complexity of the model
    - We can withhold some of the training data and use it instead as test data. If the performance of the model differs greatly between the training data and test data, we know that the model is overfitted.
    - ...(not an exhaustive list)

Neural Networks

Background

Neural networks are a sort of clever way of circumventing this problem. Initially they were merely an exercise in modelling a brain using computers. Computational limitations, and lack of optimization strategies made them kind of waste. However, with advances in computation power and “learning” algorithms, neural networks have become one of (if not THE) most popular machine learning method.

How?

Instead of considering every possible term, neural networks (sort of) have a way of figuring out which combinations/products are useful.

Neural Networks

What?

TL,DR; there are many types of neural networks, and there is a lot to know about each of them.

Today we will look at the simplest types of neural networks:

Feed Forward Neural Networks

Feed Forward Neural Networks Mimicking the Brain

We want an AI that has similar properties to the human brain

- Relatively small models can tackle complex problems
- Adaptive
- Capable of completing a wide variety of tasks

Neural Network

The most widely successful ML model in the modern age.

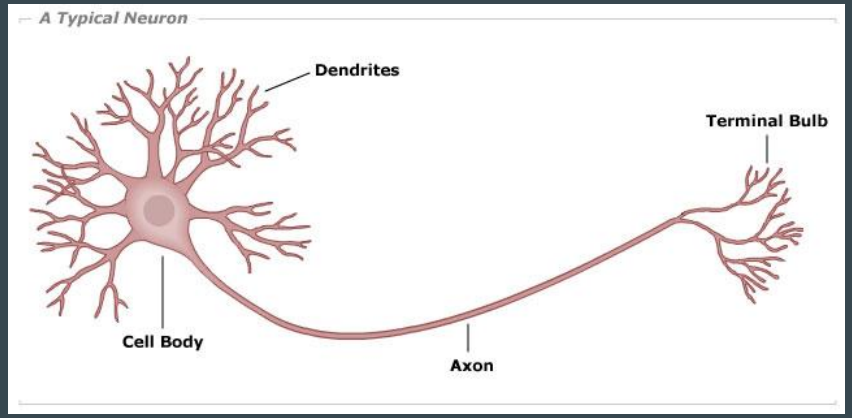
Idea: Create a model that mimics the organizational pattern of a brain

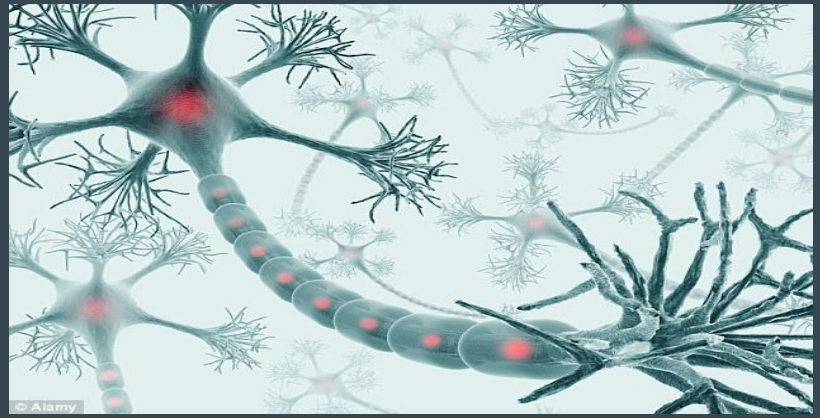
> Group of highly interconnected units (neurons), each capable of performing very basic computation

Upon testing the model, we've shown that this structure can have all of the properties we're looking for

Building Blocks: The Neuron

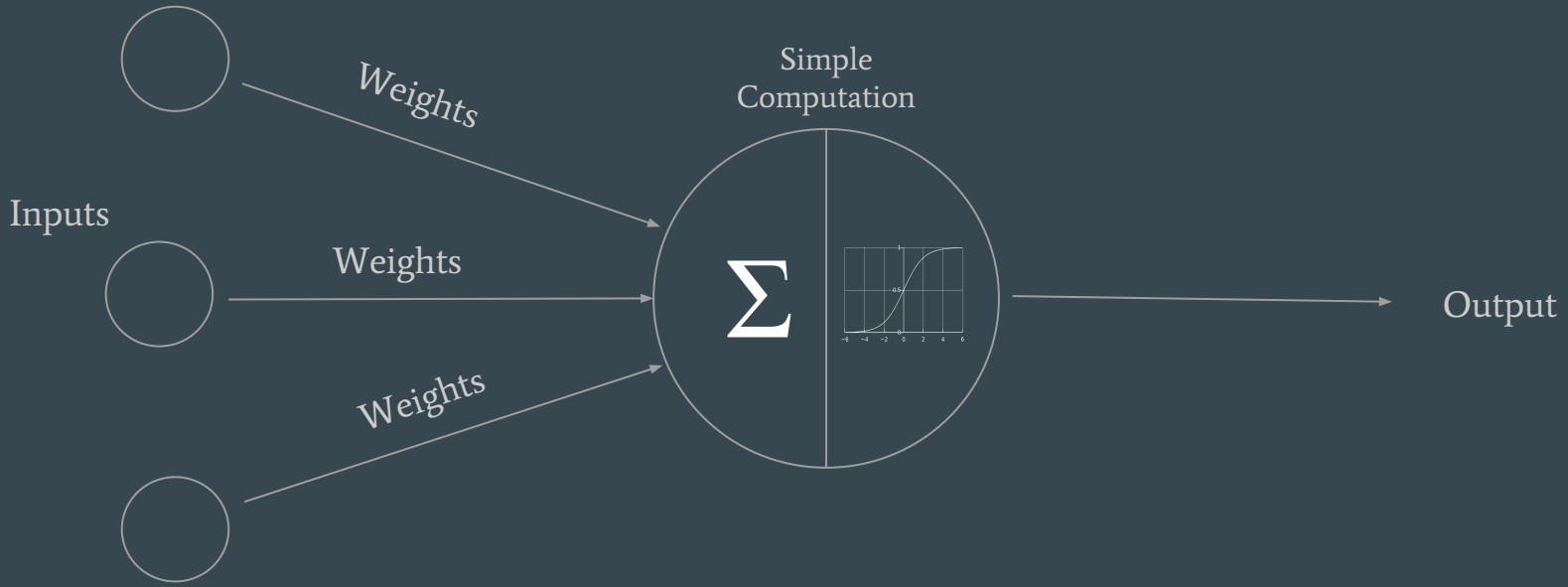
Recall how neurons work in our brains





Dendrite Input -> Calculation -> Axon Spike -> Terminal Bulb -> New Dendrite Input

Building Blocks: The Neuron

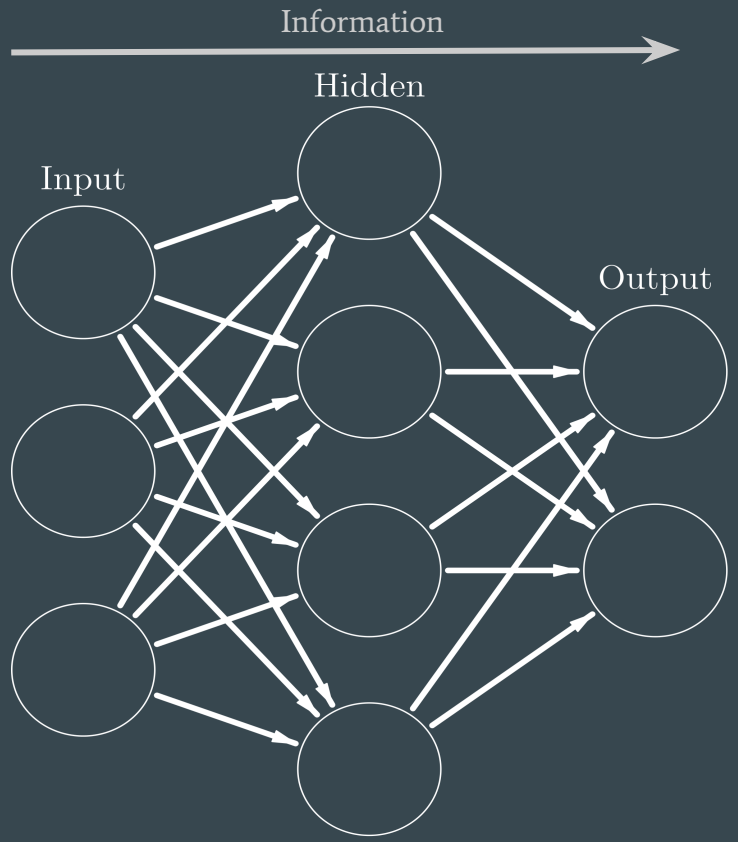


Bringing Them Together

The true power of a neural net comes when you connect these neurons together into networks.

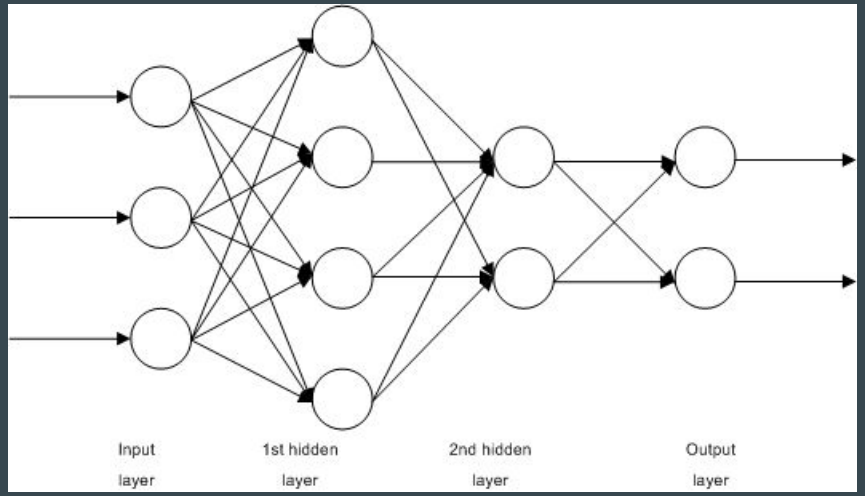
Even very simple networks can perform a variety of tasks that would be difficult for models such as linear regression

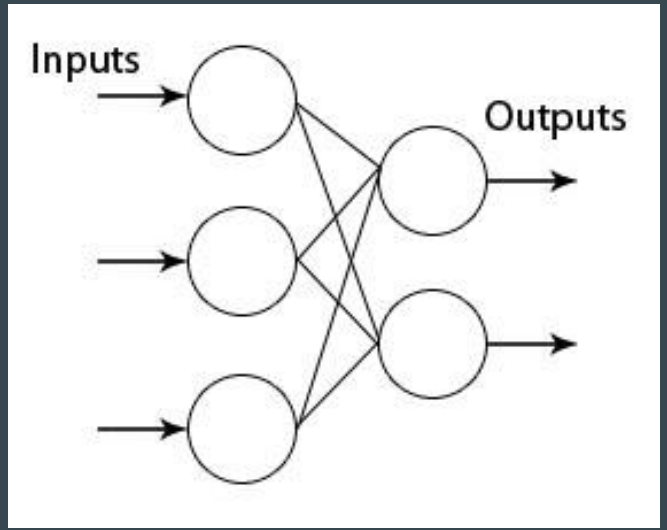
Ex: AND, OR functions



Layers

The types of problems that neural networks can tackle is dependent on the number of layers in the net.



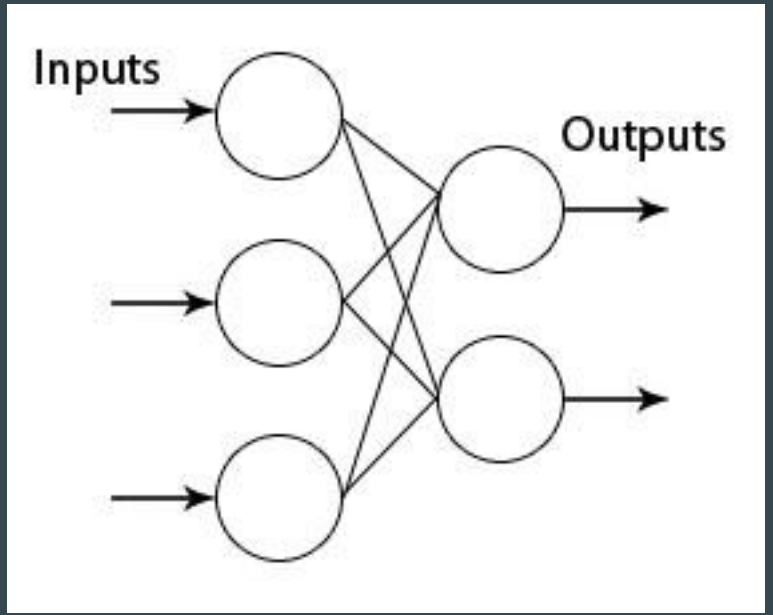


Layers

Single Layer, feed forward neural networks are the simplest type of neural network.

(In counting the layers, we kind of neglect the input layer)

They are limited in that outputs can only be linear combinations of inputs. In mathematical terms, we say that single layer neural nets can only solve linearly separable models.



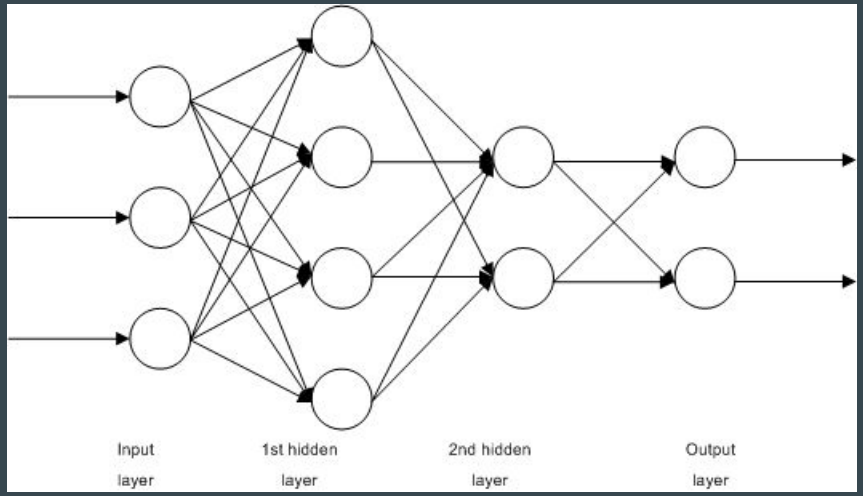
It has been shown that single layer nets cannot compute the XOR function

Layers

Multi-layer, feed forward neural networks are a little more complicated.

They can compute things like the XOR function... And so, so much more

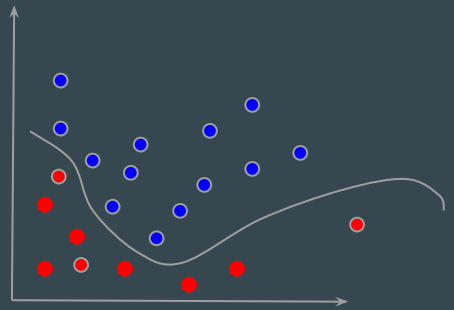
According to the universal approximation theorem, multi layer nets with just one hidden layer can approximate any continuous functions on a compact subset of R^n



Making Connections

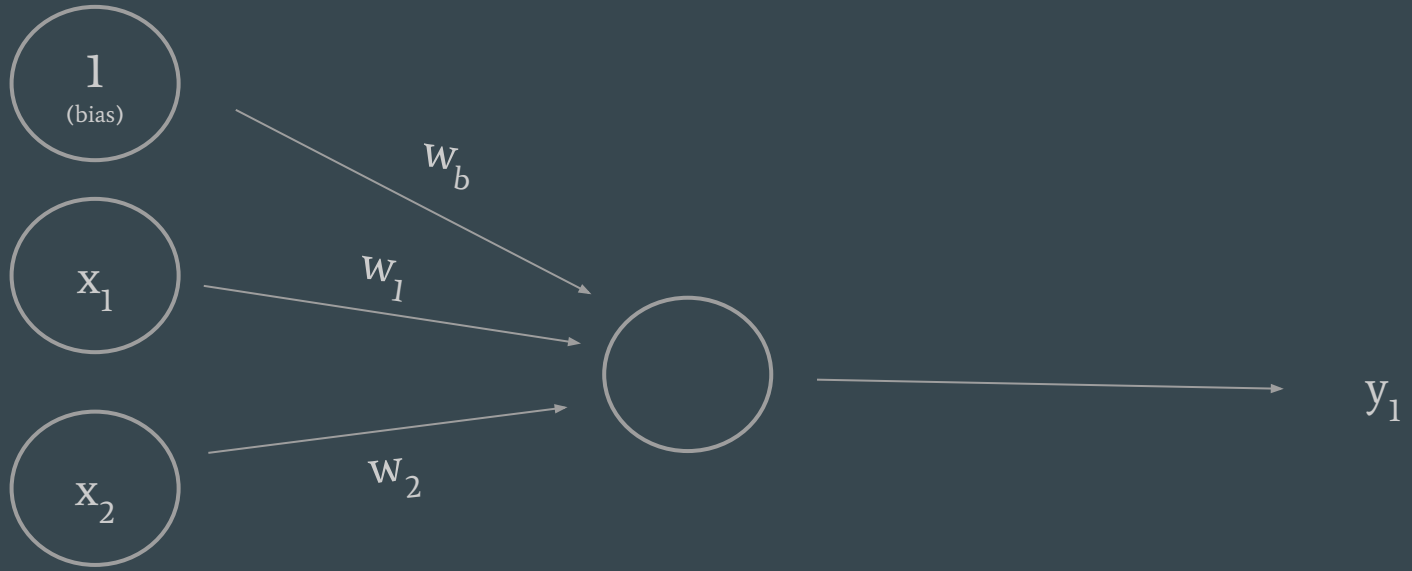
Thinking back to the functions we were trying to approximate before

$Y = A + Bx + Cx^2 + ...$



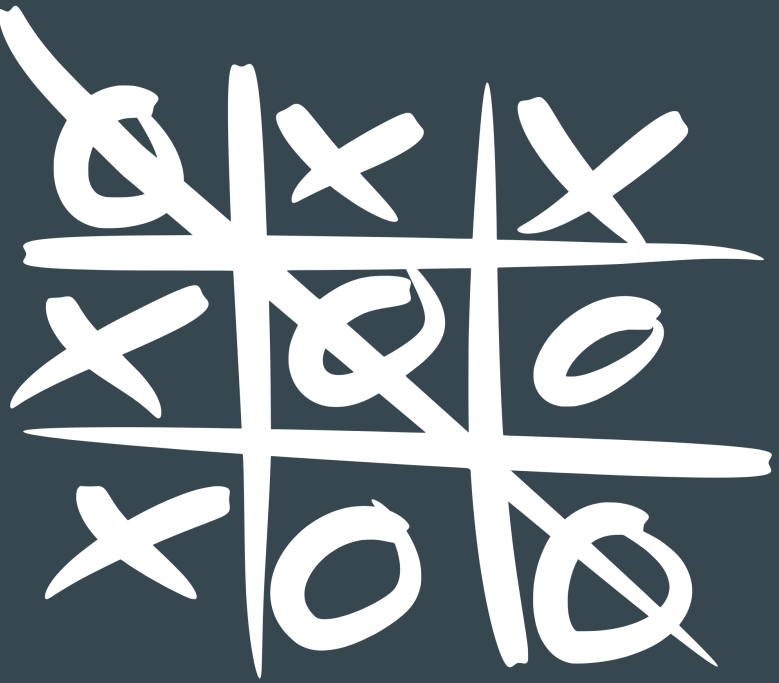
Neural nets just became an attractive option.

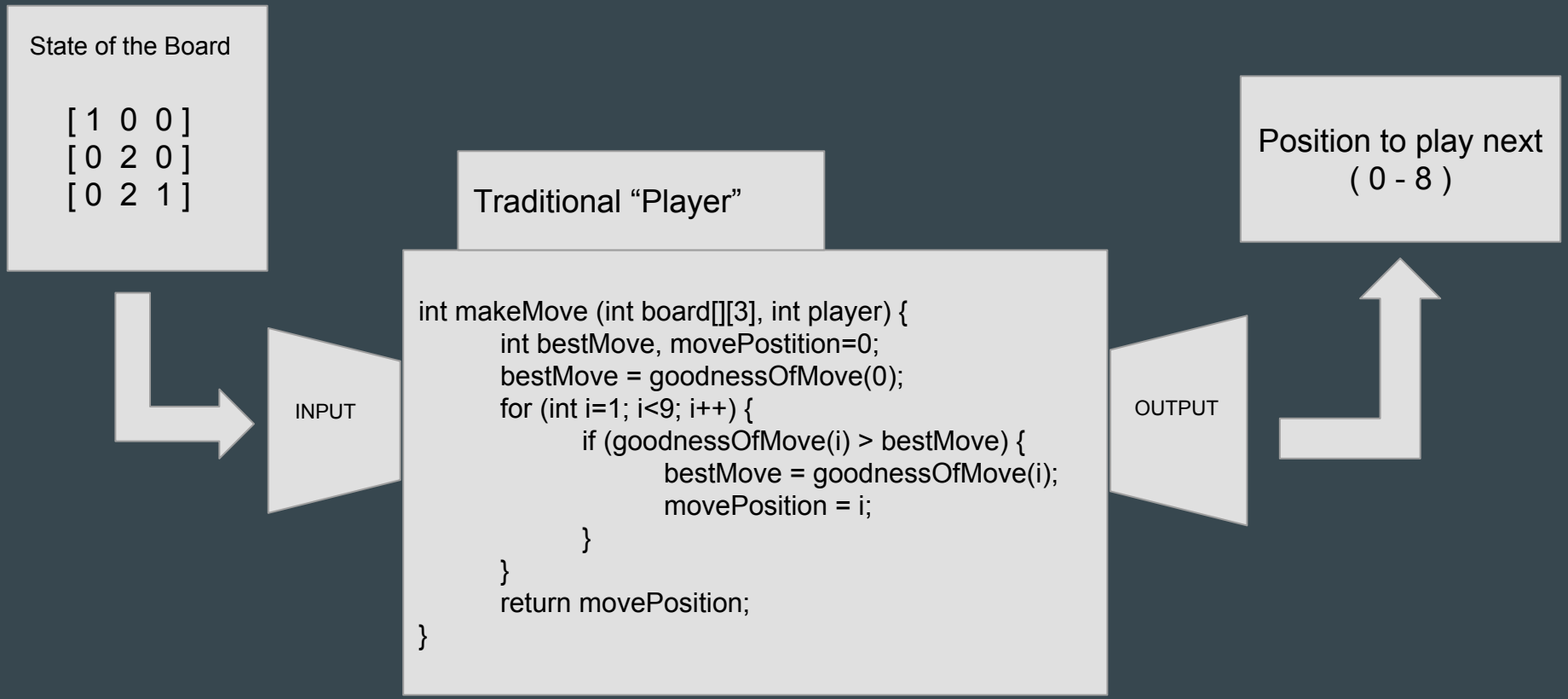
Quick Example



More Complex Example

Tic Tac Toe. (Lol imagine trying to do this with linear regression)




State of the Board
[1 0 0]
[0 2 0]
[0 2 1]
INPUT
Traditional “Player”
int makeMove (int board[][3], int player) {
int bestMove, movePostition=0;
bestMove = goodnessOfMove(0);
for (int i=1; i<9; i++) {
if (goodnessOfMove(i) > bestMove) {
bestMove = goodnessOfMove(i);
movePosition = i;
}
}
return movePosition;
}
OUTPUT
Position to play next
(0 - 8)

State of the Board

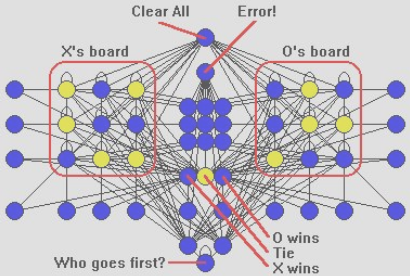
[1 0 0]
[0 2 0]
[0 2 1]

INPUT

Our Player

Neural Network

- takes in the state of the board as input
- returns position to play next as output



OUTPUT

Position to play next
(0 - 8)

Challenges

Can handle problems that are orders of magnitude harder

BUT

How can we optimize our model?

New Optimization Algorithms

Traditional methods like gradient descent can’t be implemented in the same way due to the complexity of our model and its output.

Calculating the “gradient” has become much harder

Luckily, people smarter than us have already worked out the nitty-gritty details.

Optimization (“Learning”) Algorithms

1. Conventional Backpropogation
2. Backpropogation with momentum
3. Adaptive step algorithms
4. Spray-and-approach algorithms
5. Relaxation methods
6. QRProp
7. Linear Oscillators

Backpropagation

ML experts have developed a method of calculating the “gradient” of a neural net.

1. Run a training example
2. Calculate difference between net’s output and accepted output
3. Look at the last layer of the net
4. Calculate how much each neuron contributed to this “error”
5. Define this to be the “error” of the neuron
6. Continue this backwards traversal until the error of each neuron if known

Once these values are known, we can simply plug them into one of our optimization algorithms.

Genetic Algorithms

Genetic Algorithms mimics the process of evolution for the purpose of optimization:

The algorithm will:

- randomly generate a population of hundreds of unique tic-tac-toe players
- have them play against one another.
- rank each player based on its skill.
- Then, delete the worst players and use slightly modified versions of the best players to make up the next generation

Just as in biological evolution, natural selection will take place and after sufficient generations a “smart” player will emerge from the randomness.

Genetic Algorithms:

Steps:

1. Define a model that can accomplish your task (i.e. a net that plays tic tac toe)
2. Create a “mutate” method
    a. Define a way of making a slight random change to your model
3. Create a “mate” method
    a. Define a way of combining the qualities of two model instances
4. Plug your model into our genetic algorithms library
5. Watch as the initially random models evolve and become increasingly better at the task at hand

Pros and Cons

Works for ANY models that can be ranked by ability

- All game playing AIs
- Any model with a cost function
- Pretty much every supervised learning application

Less systematic than alternatives

- Must define sensible methods for mutation and mating
- Many tweakable parameters
- Hard to “get right”

Summary, Discussion & Questions

& cool stories?