



Introduction to Artificial Intelligence

Week 13



Artificial Neural Networks

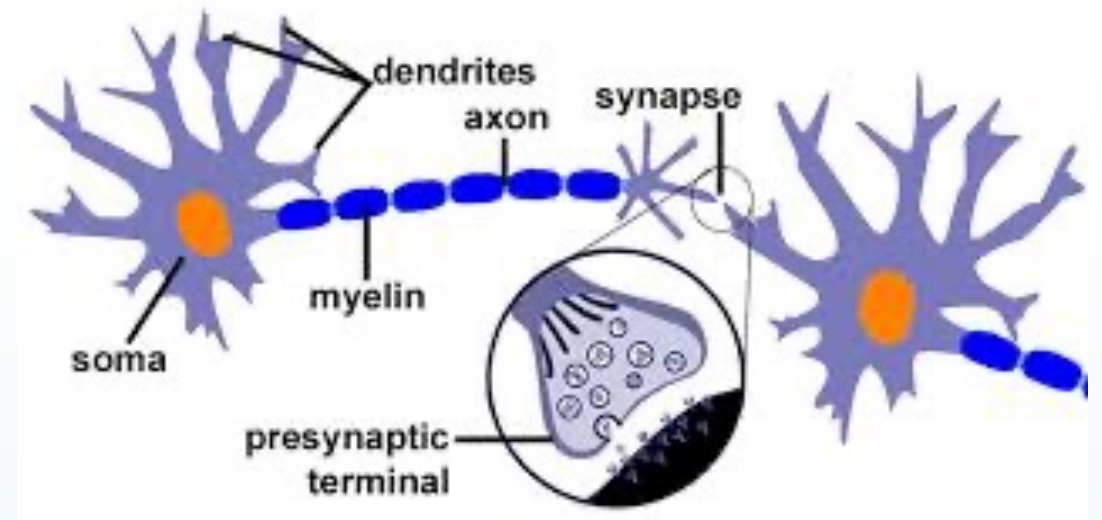


Artificial Neural Networks

- Again – biological inspiration
 - Neurons in the brain
 - First models from 1957
 - F. Rosenblatt on contract to US Navy
 - Intended to be hardware machines
 - Implemented in software to prototype
 - Now seen in both software and hardware forms
 - “the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence” – NY Times
- 1969 saw the death of many of these connectiveness models (Minsky-*Perceptrons*)
 - XOR could not be developed via such a model
 - Research funder saw the dream of AI to be a farce and pulled back money
 - AI Winter
- Funding now returning – we are in an “AI Spring”?

Biology

- Neuron Cell
- Information
 - Processing
 - Message Passing
 - Input from the dendrites
 - Output from the synapse
- Electrical message passing model
 - Spike models from 1907 have been investigated





ANN

- Specified by:
 - Architecture
 - How are the connections made
 - What Neurons are connected to others
 - Feedforward
 - Recurrent
 - Neuron Model
 - What functions can the neurons take as inputs
 - Can restrict the learning algorithm (backpropagation requires the functions to be differentiable)
 - Learning Algorithm
 - Connected Neurons have weights in their connections
 - How do we make the updates to weights?
 - Backpropagation of errors
 - Evolutionary methods
 - Supervised or Unsupervised




(Un)Supervised Learning

- Supervised
- Tasks with outcomes
- Tasks
 - Classification

- Unsupervised
- Task inputs with no expected output
- Tasks
 - Clustering and Data understanding

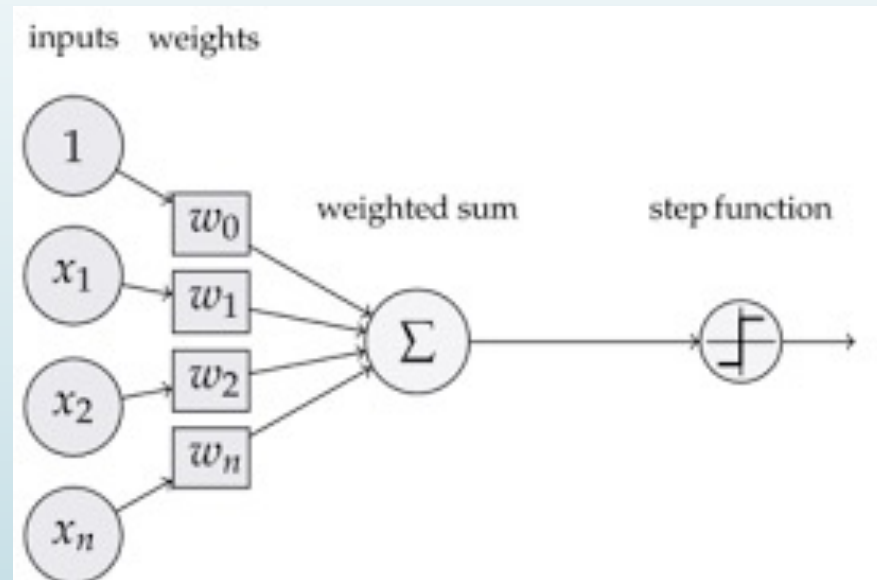


ANN Usage Fields

- 
- Speech recognition
 - Computer vision
 - Music, art generation
 - Classification
 - Non-linear tasks
 - etc.

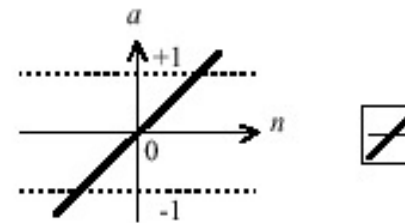
Perceptron

- Single layer feedforward networks
- Simplest version of a neural model
- $output = step_0(\sum w_i x_i)$ where
 $step_0(n) = 1$ if $n > 0$
and 0 otherwise



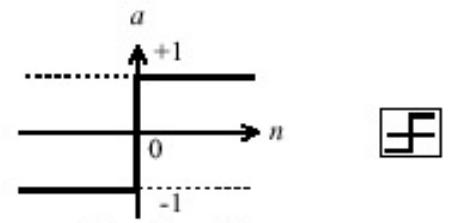
Activation Functions

- Generally non-linear
 - Otherwise output is input
- Required to be differentiable in backpropagation training
 - Uses a calculation of error
 - Needs to get the integration of the differentiation to find error



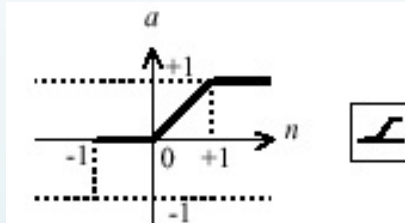
$$a = \text{purelin}(n)$$

Linear Transfer Function



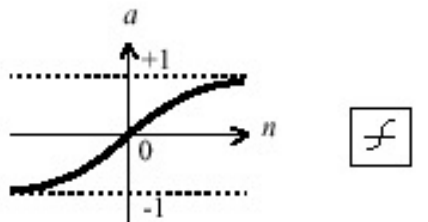
$$a = \text{hardlims}(n)$$

Symmetric Hard Limit Trans. Funct.



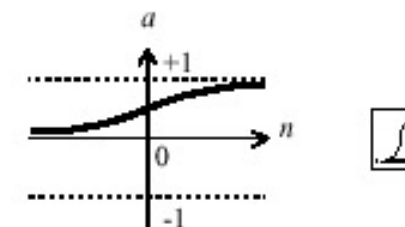
$$a = \text{satlin}(n)$$

Satlin Transfer Function



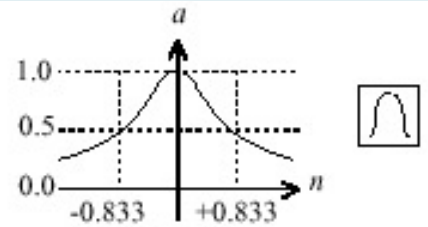
$$a = \text{tansig}(n)$$

Tan-Sigmoid Transfer Function



$$a = \text{logsig}(n)$$

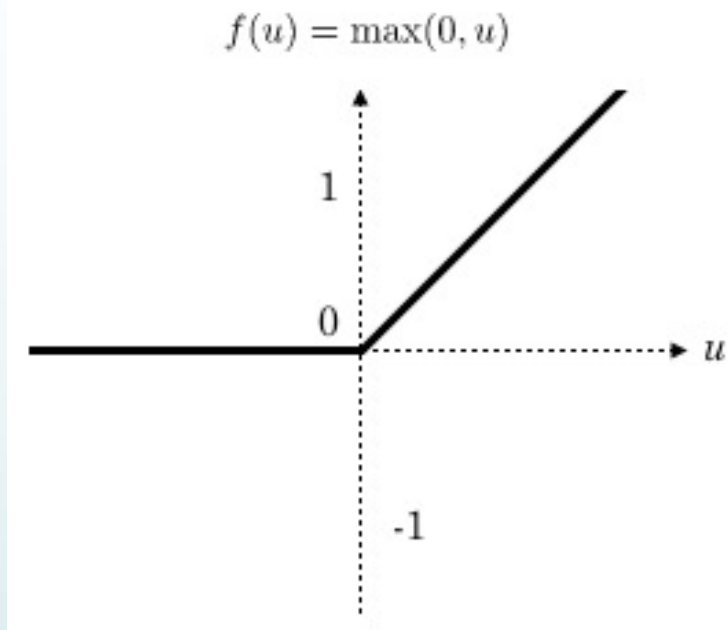
Log-Sigmoid Transfer Function



$$a = \text{radbas}(n)$$

Radial Basis Function

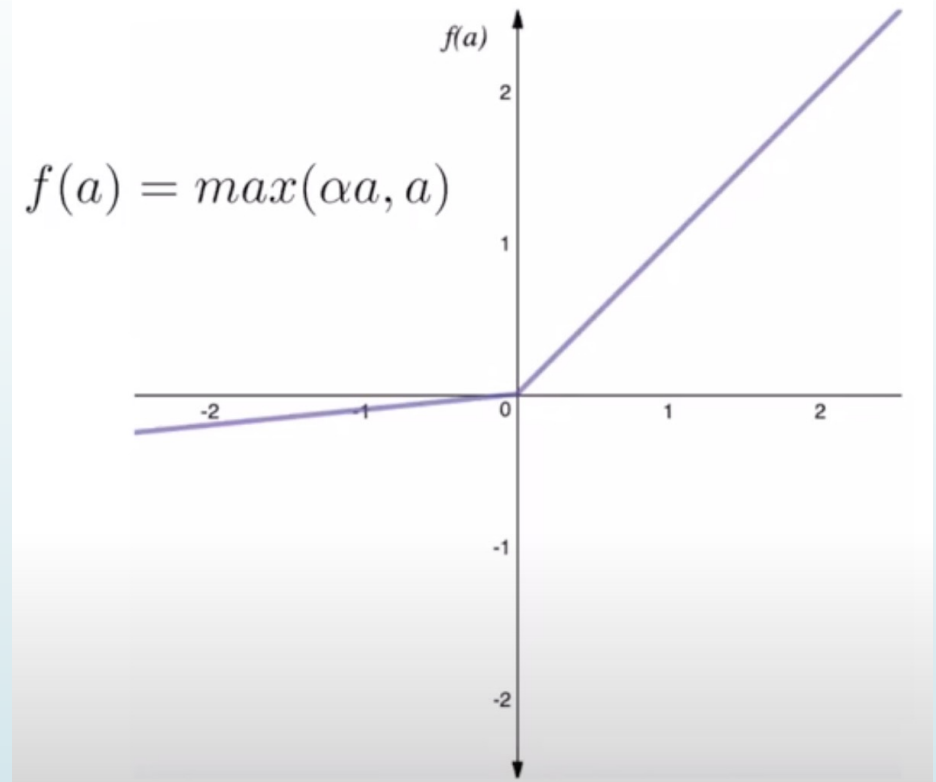
Rectified Linear Unit (ReLU)



- Where u is an input to neuron
- One of the most popular activation functions

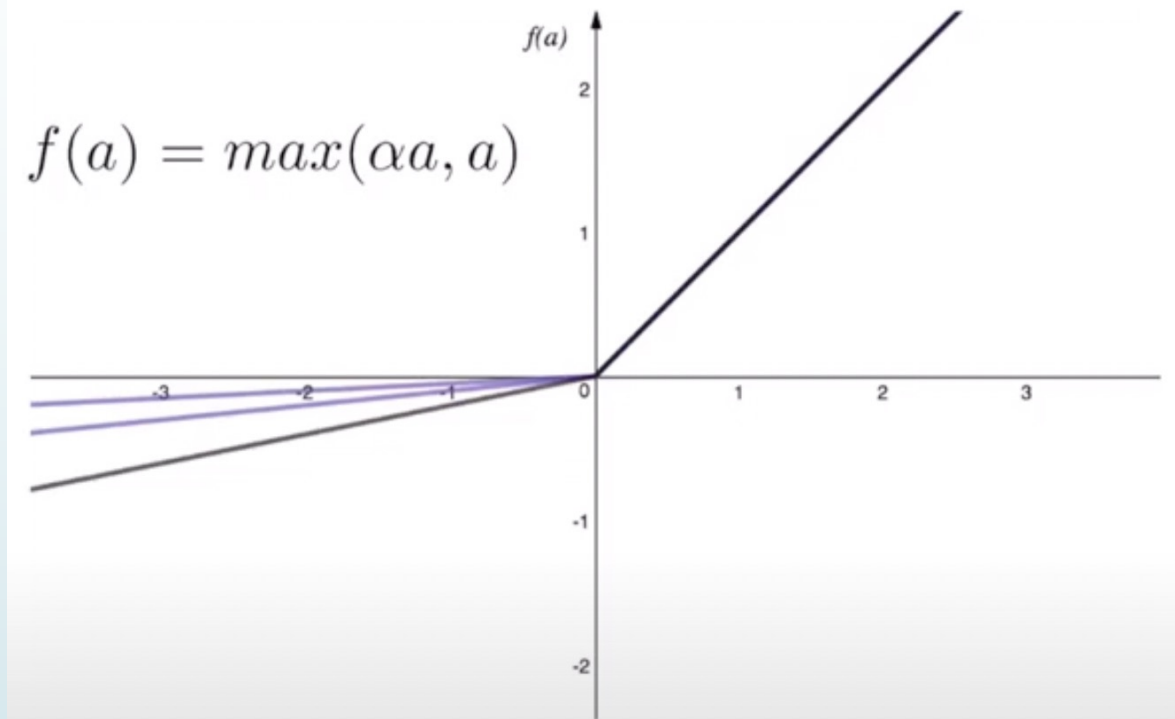
Leaky ReLU

- More advanced ReLU
- α is a hyperparameter chosen and fixed initially
 - Usually, small value

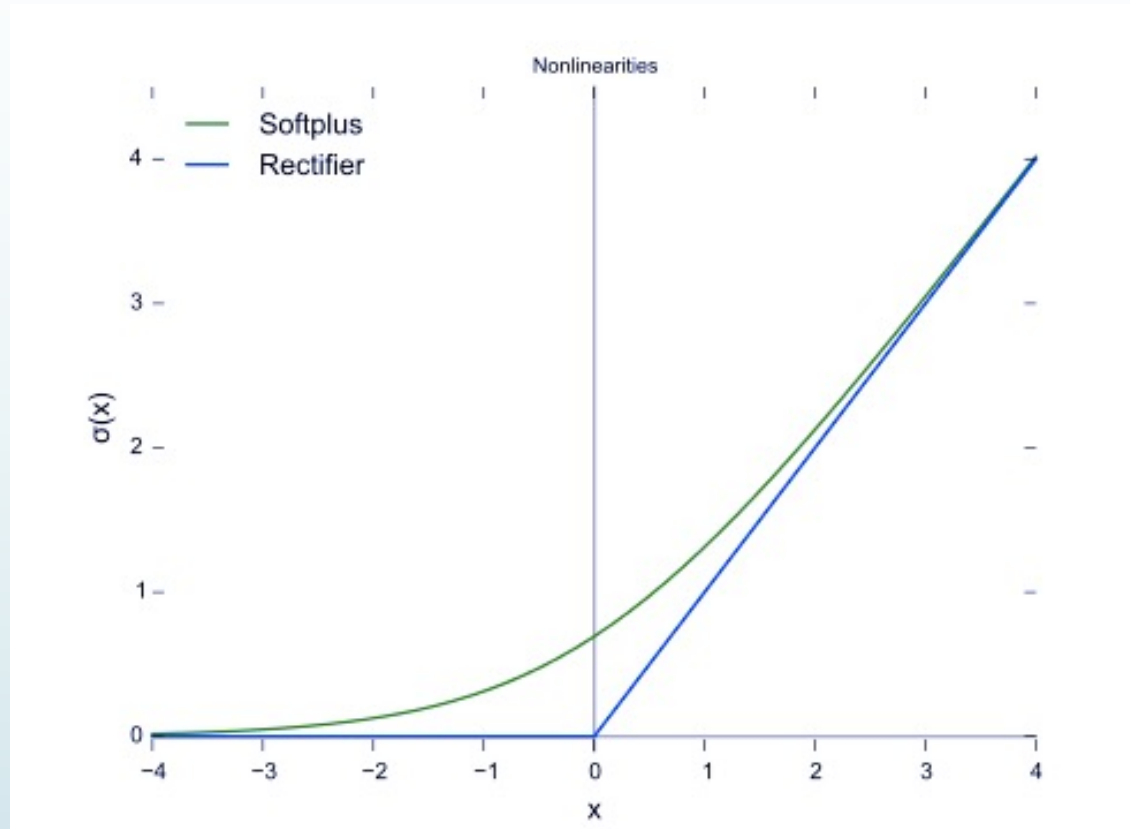


Parametric ReLU

- Advanced Leaky ReLU
- α parameter can be trained

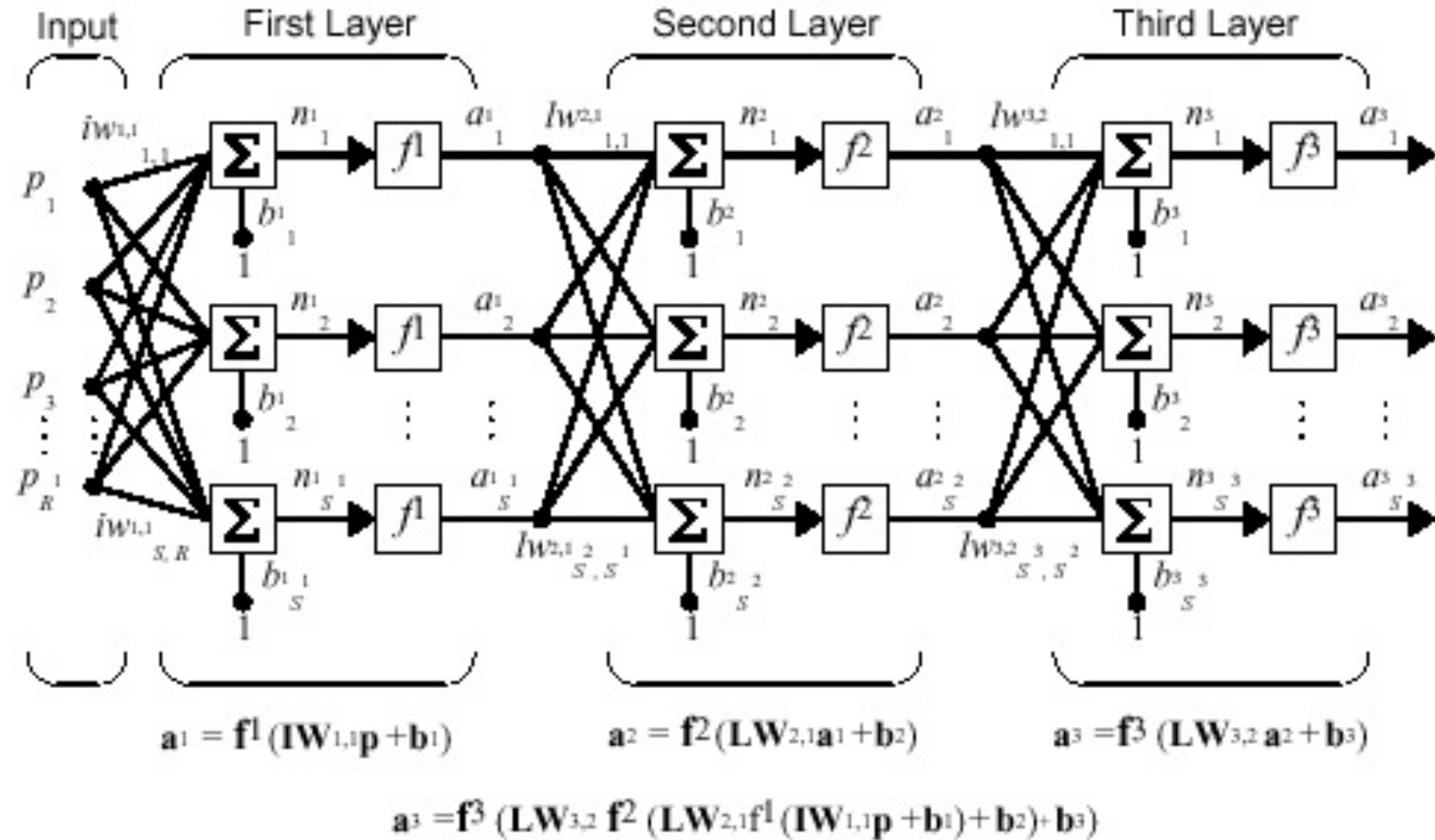


Softplus



- Softplus is a approximation to a ReLU which is $f(u) = \ln(1 + e^u)$

Feedforward Network





Backpropagation on a Network

- Single layer can only deal with linearly separable problems
- To deal with multiple dimensions the network needs to expand in the number of layers
 - Increases the dimensionality; everything in a high dimensional space BECOMES linearly separable
- As this expands larger and larger the type of problems which can be faced expands
 - Deep Learning
 - Beware of Overtraining

Basic Training Algorithm

- Initialize the weights in the network (often randomly)
- **repeat**
 - **for** each example e in the training set **do**
 - $O = \text{neural-net-output}(\text{network}, e)$; // forward pass
 - $T = \text{teacher output for } e$;
 - Calculate error $(T - O)$ at the output units;
 - Compute $\mathbf{w}_j = \mathbf{w}_j + \alpha * \text{Err} * I_j$ for all weights from hidden layer to output layer; //backward pass
 - Compute $\mathbf{w}_j = \mathbf{w}_j + \alpha * \text{Err} * I_j$ for all weights from input layer to hidden layer;
 - Update the weights in the network to the new weights;
 - **End for**
 - **until** all examples classified correctly or stopping criterion met
 - **return**(network)

Calculating Error in Backpropagation

- Each node is considered to be part of the error in the overall function based on its weight in the input

$$\text{error of hidden node } j = \sum_{i \in \text{outputs}} w_{ij} \delta_i$$

where δ_i is the error at output node i .



Issues with Backpropagation Networks

- Black Box (all ANN)
 - How did you get the answer – can we form any rules?
 - I had the program run and this thing was made
 - Have to change it into a lookup table and figure out why it has those weights
- Large training samples needed for backpropagation to work effectively



Other Trainings and Connection Methods

- Neural Evolutionary Models
 - Have a GA train the network – even allow the GA to assemble the network
 - NeuroEvolution of Augmenting Topologies (NEAT)
- Hopfield Networks
 - Train much like perceptron
 - Not connected in a linear manner



Neural Evolution

- Treat the weights as being the elements of the chromosome
- The fitness function is the amount of error
- This allows for the use of non-differentiable functions in the elements
 - We do not need to rationalize the error caused by each node
 - This allows for problems without clear gradients to be attacked with neural methods
- Can also evolve the structure and connections within the network and build new infrastructures
 - Information propagation



Deep Learning

- Not a new topic – the biggest developments were done in the 1970s
 - Issue – AI Winter
 - Issue – Lack of High Performance Computing (HPC) infrastructures
- Removal of these problems has lead to a boom in Deep Learning Research
 - AI Winter – funding in AI is returning and expanding
 - HPC is now affordable to even start-ups for reasonable costs
 - Cloud Computing
 - Funding from government interests
 - Moves by Google/Amazon etc.



Attacking the Unassailable

► GO

- If chess is a hard problem, then GO is an improbable problem
- Chess n-foction of tree bounded by at most all moves that can be made by 16 pieces – each piece between 3 (pawn on non-home square) and 4×7 moves allowed to it (queen able to visit all other squares in a rank, file, or two diagonals), worst case upper bound is $16 \times 4 \times 7$ open moves on a turn (note very generous overage assuming all pawns are now queens)
- In GO, upper bound of moves is to place a stone in one of a 64 by 64 board, 64×64 nodes to visit
- How?



Research into GO

- Lots of work has been done on the problem of GO due to its complexity
- Full special sessions and Journal articles have been dedicated to advancements in play
- Many of these look at higher level structures rather than individual plays
 - Eyes (places you cannot put an enemy stone as it would be captured)
 - Board edges (Again building from the uncapturable nature of these squares)



Why is GO so important – it is only a game!

- ▶ Games provide interesting surrogate problems for real world issues
- ▶ GO is a game of structures and a game with huge number of potential moves
 - ▶ We are very interested in this problem as developments in this game demonstrate perhaps methods we could use to investigate other complex structures and look at other huge move problems
 - ▶ Cancer cells
 - ▶ Protein arrangements
 - ▶ Molecules



Alpha GO

- Developed in X by Google
- World leading method for the playing of GO
- Uses deep learning (large many multiple layer Neural networks trained by hours and hours worth of Go playing) in order to develop patterns in the play, and learn the correct structures
- MCTS is then used to produce the plays with the deep network as the evaluation method



Adversarial Networks



- Say we are not interested in classification, but in creation of objects which are close in style to an artist
 - Images of Cats
 - Music
- We know how to make good classifiers for a problem
 - We can say it is a cat or not a cat
 - We can say it is music or noise
- The generator's job is to become so good that our classifier says its output is in the right class



Forgers and Detectives

- The detective begins his work, looking at a number of real notes coming from the treasury department, he studies them. He also studies a group of randomly coloured sheets of paper
 - Once the detective can distinguish between banknotes (YES) and scraps of coloured paper (NO) to a sufficient extent he is given a badge and goes on the hunt
- The forger now begins its work, taking random scraps of paper as input (random strings) the forger outputs some fake currency
- The detective now looks at the work of the forger
 - If the forger fools the detective then they can spend the fake currency and are reward a point
 - If the forged bill is detected then the forger gets nothing



Forgers and Detectives (cont.)

- After seeing where the forger was correct or not, the forger trains based on this outcome, attempting to ensure that all these forgeries are not detected
 - The forger over time will become better and better at fooling the detective
- Once the forger is fooling the detective enough then the detective goes back to the crime lab and studies the current set of forgeries which are passing them and a new set of examples of real currency
 - The detective over time will become better then at not being fooled
- This process continues of training each to defeat the other in a zero sum game
 - The better the forger is, the worse the detective has become and vice versa



How Does this Now Generate Stuff

- We can remove the detective and then look at the forgeries
- As they are training against one and other and the network is sufficiently large, i.e. can build very large equations and add in new terms without disrupting other structures, it is likely that forgers become good enough to fool not just the detective but humans
- There is a risk of overtraining the network though and just producing the input reality sets
 - This is fixed by rotating and modifying the training samples and having enough representative samples



Libraries



TensorFlow



Keras

 PyTorch