Day 20: Neural Networks

The last section of this ran overtime and so was pushed to Day 21, but it's included in these notes.

Connectionism

- What is *Connectionism*?
  - Neurally-inspired approach to cognitive modeling
  - Composed of simple, cheap computational units (like neurons)
  - Connections are like synapses, and controlled by weights
  - Input function combines output from other units
  - Output function is a function of input and state
  - Learning function adjusts the weights over time
  - *Connectionist Aesthetic*
    - Units and connections are neither dynamically created nor destroyed
    - It should only take 100 cycles to accomplish interesting tasks
      - This was inferred from how long a real neuron takes to fire, and how fast humans can do this.
    - This enables parallel computing and biological plausibility
  - Connectionism has a long and bumpy history
    - Some work in the 40s-60s.
    - AI Winter in the 70s
    - Justifications of focusing on connectionism over symbolic AI has varied over time:

        - Parallel processing, neuro-realism, deep learning, it's very effective, tec.
        - Styles of NNs change over time
- What is your mind?
    - Eludes definition, metaphors are used to try and define it
    - One possibility: your brain is a computer and your mind is the program
- Why does Professor Pollack think Connectionism is popular?
    - Looking for an explanation of cognition that isn't purely logical, which is unrealistic
    - As new types of NNs are invented, old ones are thrown out
- Why study non-symbolic neural-like cognition?
    - 97% of brain mass/evolution was pre-symbolism, and that's enough for humans to survive
- Minsky & Papert's argument against perceptrons (1969)

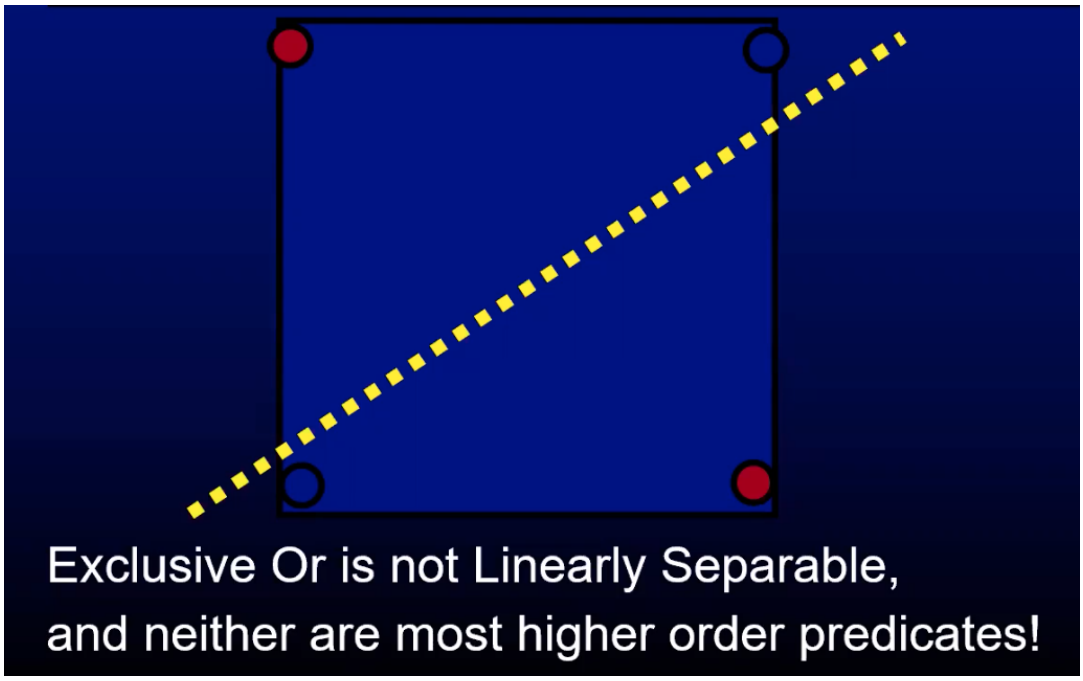
> No machine can learn to recognize X unless it possesses, at least potentially, some scheme for representing X.
>
> Perceptrons have no way to represent knowledge.

- More interest in NNs/Connectionism in the 80s, 90s, and 00s. Big leap was the invention of backpropegation/deep learning in 2012.

Neural Networks

- Neural Network Framework:
    - Set of simple, cheap, processing units, each with some state (a number)
    - Weighted mono-directional connections between units
    - Simple combination function for the inputs

        - Simple computation function for the output
        - Nobody knows how this would produce consciousness, but also nobody knows how the brain makes consciousness.
- Two categories of neural networks:
    - Recurrent/dynamic systems
        - Has state (usually a number, sometimes a vector)
        - Moves from state to state, driven either by input or by time
        - No good learning algorithms yet, often configured manually
    - Feedforward associationist/classification systems (focus for the rest of the lecture)
        - Stateless, purely transforming inputs to outputs
        - Many learning algorithms with different characteristics
- McClelland & Rumelhart (1981): hand-made static network for recognizing words from line segments (layers: line segments → letters → words)
- In practice, NNs are often represented with their weights as a matrix and their state as a vector, with a nonlinear filter somewhere
- Perceptron: threshold logic that learns: first neuron
    - Compares a linear combination of inputs to a threshold, outputs a 1 or a 0 accordingly
    - Can't do multi-layer perceptron networks, because training one layer requires completely re-training all subsequent layers.
    - Can do most boolean logic (except 2/16 of the two input binary operators)
        - Could only do things that were linearly separable:



- Backpropegation: algorithm for training neural nets
  - Replaced binary thresholding with a continuous, differentiable, sigmoid function $\frac{1}{e^{-x}}$ (range: $[-10, 10]$).
  - Makes learning stable over multiple layers
  - Basic idea of backpropegation: you can calculate the error of the whole network by comparing the output to the expected output. You can also come up with an (algebraic) definition of any node's output as a function if its inputs and weights.
    - Ergo, you can calculate the error of each output nodes as a function of its inputs and weights.
    - You can plug that into the formula for calculating the output of the penultimate layer as a function of its weights and inputs.
    - Repeat this till you get to the error of the input layers as a function of their input and all the weights
    - Do gradient descent in weightspace
  - Process of backpropegation:

    - Forward pass: run the network for each case
    - Backwards pass: using the process above, calculate the error as a function of the weights (sum errors for each case)
    - Do gradient descent on that function in weight-space, and update weights accordingly
    - Eventually, the error should asymptotically converge towards 0
- Implementation details (most of these are black arts of how to tune them):
    - Two learning parameters: μ and α (controls velocity and acceleration of changes)
        - More complex problems need lower rates
        - If your learning rate is too high, it can get (falsely) stuck on one path
    - Bias nodes: every hidden and every output node needs a connection to a node with a fixed value of 1
        - This is equivalent to a perceptron's threshold
    - Epoch vs. immediate training: updates weights after training all cases (more common) or after each one, respectively
        - Modern Deep Learning uses *Stochastic Gradient Descent*, where the network is trained on one smaller random subset at a time
- Not super clear why this works so well
- Many applications: curve fitting, logical function inference, inductive classification, etc.
- Improved lots over time
- What can backpropagation do?
    - Logical functions: all 16 binary functions (requires 3 layers)
    - Data Compression: auto-associator: self-supervised model
        - Idea: input and output layers are the same size, while a hidden layer is smaller. If properly trained, you can put in data and get the

same data out, but the hidden layer will be a compressed representation

    - Other, more sophisticated NN compression techniques exist too
- Classification
    - Each input node is a feature, usually [-10, 10] or {0, 1}
    - Output can be either one bit, a 1-in-n code, or a set of "likeness values" (one output node per category)
    - BP can learn weights that perform classification
        - Slower than decision tables, but allows some ambiguity