CS 486 — Lecture 8: Artificial Neural Networks, Part 1

1 Neural Networks — Introduction

- Machine Learning uses statistics to learn.
- Deep Learning is a branch within ML, by mimicking the human brain.
- ImageNet and the cat experiment were projects regarding image recognition using Deep Learning that helped popularize it.
- The human brain can learn very complex relationships between inputs/outputs — how can we replicate this?
- The brain is a set of densely connected *neurons*.
- The dendrites receive inputs from other neurons.
- The soma controls how strong the output signal is.
- The axon sends outputs to other neurons.
- The synapse links between neurons.

2 Neurons

- A simple mathematical model of a neuron is to use a linear classifier — this fires when a linear combination of its inputs exceeds some threshold.
- A neuron j computes a weighted sum of its input signals.
- Then, the neuron applies an activation function, g, to the weighted sum to derive the output.

$$a_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} a_i)$$

- $a_0 = 1$ is known as the dummy input, and $w_{0,j}$ is our bias weight. This allows us to have a constant term in our linear function.
- So how do we choose our activation function?
  - It should be non-linear. Most complex things in the world are non-linear. A possibility of using a linear function is that we end up stacking many linear functions together, and no matter how many times we stack linear functions, we end up with a linear function.
  - It should mimic the behaviour of real neurons. If the weighted sum of input signals is large enough, then the neuron fires.
  - It should be differentiable almost everywhere. This is mostly for learning; we will use algorithms like GD to learn, and that requires the derivative to decide which direction we go.
- One common activation function is the step function, $g(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases}$. This isn't that good though as while it's simple, it's not differentiable. Also not used much in practice.
- Another option is the sigmoid function, $g(x) = \frac{1}{1+e^{-kx}}$.

  - This also has some problems.
  - It approximates the step function as k increases.
  - It gives clear and bounded predictions.
  - Differentiable!
  - However... it causes a problem called vanishing gradients. If the value of x is very large or small, the value of $g(x)$ changes very little (flat curve).
  - This means that $g(x)$ responds very little in that area.
  - So, the gradient is very small.
  - So, the network learns slowly or not at all in that area!
  - Another problem is that it is computationally expensive.
- The rectified linear unit, or ReLu, is a function like $g(x) = \max(0, x)$. This is efficient, differentiable, non-linear. But again, while it fixes the vanishing gradient problem on the positive side, it doesn't solve it on the negative side (dying ReLu problem).
- The leaky ReLu is $g(x) = \max(0.1 \times x, x)$. This results in a small positive slope in the negative side rather than just a flat line at 0. This enables learning for negative input values!

3 Perceptrons

- Feed-forward network
  - No loops — a directed acyclic graph with no loops.
  - One could say an output is a function of its inputs.
- Recurrent network
  - Has a loop — this allows a node to have some memory (a la circuit design).
  - This adds complexity; an output is now a function of its current inputs *and* potentially its current state.
  - RNN makes more sense when comparing to real life brains, but it also adds complexity to both training and interpretation of what happens to it.
  - We mostly focus on FFNN.
- The perceptron is a single-layer FFNN.
- The inputs are connected directly to the outputs.
- We can use a perceptron to replicate a logical function (ie: AND, XOR, etc.).
- People thought that AI would be solved if we could achieve simple logical reasoning...
- Unfortunately, perceptrons have limitations.
- XOR cannot be represented using just perceptrons. We would need a deeper network, which people couldn't train at the time (only knew how to train single-layer, which perceptron was)!
- This showed that perceptrons couldn't solve everything; this led to AI development to freeze for a while.
- So why can't perceptrons represent XOR? It's because XOR is not linearly separable!
- We could represent XOR with a 3-layer NN (input layer, hidden layer, output layer).