Day 7: The Modeling Problem and More

The Modeling Problem

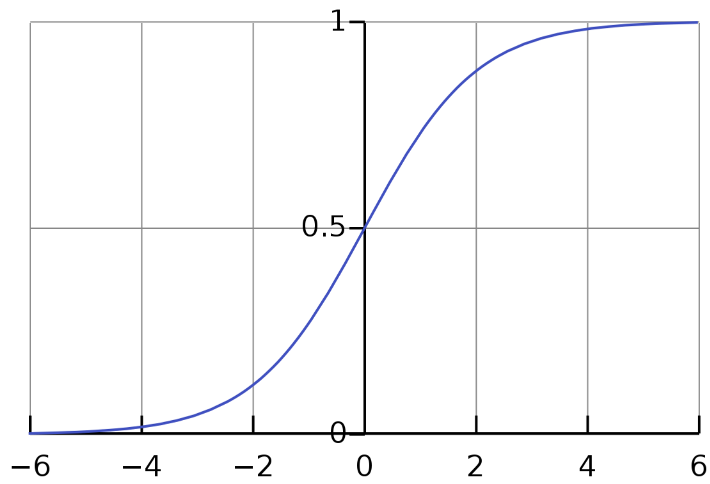
At a baseline level, we fed observed data into a mathematical model to yeild predictions and/or decisions. Ultimately, we want some function F that is designed to take in some input and give an “intelligent” output.

- Traditionally requires domain expertise
- Often unclear how to improve on a baseline model
- Deep learning takes simple building blocks that allow us to learn a model through an optimization procedure
    - Doesn’t require nearly as much domain expertise to develop a model
    - Need data—usually lots of it
    - **key** ~ neural networks are a way of solving the modeling problem using a common set of building blocks

The Neuron

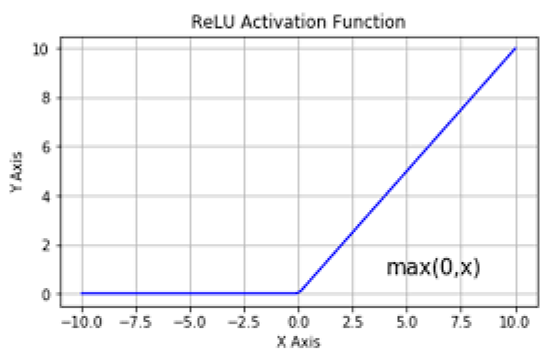
- A mathematical function - “activation function”
    - Nonlinear
    - **Monotonic** ~ the output of a monotonic function *either* never increases *or* never decreases as input
    - Can be thought of as having “no” or “off” states
    - Example: sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



- Example #2: relu (rectified linear unit)

$$relu(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$



- Parameterization
    - Adding “knobs” to control the shape of the activations (e.g. $F(w, b; x) = \phi(wx + b)$ where ϕ is a generic activation function, w are the weights, b is the bias, and x are/is the inputs)
        - Can be converted into $v\phi(wx + b)$, where v determines a component of steepness to the activation function.

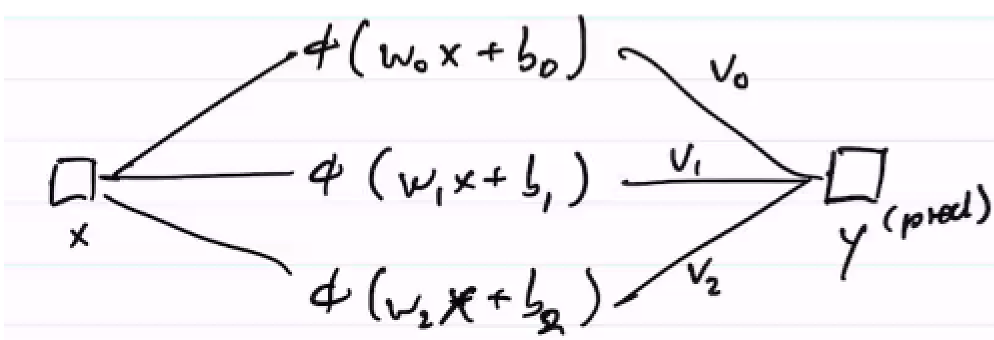
Universal Function Approximation Theorem

(one of them)

- Suppose we want to approximate a potentially unknown/complex function $f(x)$ on the domain $x \in [x_{min}, x_{max}]$ such that the approximating function $F(x)$ is within some error ϵ of $f(x)$ in the same domain.

$$|f(x) - F(x)| < \epsilon$$

- **UFA**: using $F((u_i)_{i=0}^{N-1}, (w_i)_{i=0}^{N-1}, (b_i)_{i=0}^{N-1}; x) = \sum_{i=0}^{N-1} v_i \phi(w; x + b;)$ with sufficiently large N, we can always find parameters such that $|f(x) - F(x)| < \epsilon$.
  - Why does ϕ have to be nonlinear?
    - If ϕ was linear, then UFA could not be reached because the summation of multiple linear components is another linear function, which would not fulfill the criteria.
  - Why v?
    - If ϕ is bounded, then v "unbounds" the function by letting it scale beyond it normal range
    - Final weighting of the neuron in the final outcome (prediction function)
- **Single Layer Neural Network (Perceptron)** ~ a summation of several (scaled) activation function to reach an approximating function to as closely match the data as possible



  - Hyperparameters for layers of a Perceptron (or any neural network)
    - N ~ number of neurons in the layer
    - ϕ the choice of the activation function
- Example: approximate $f(x) = cos(x); x \in [-2\pi, 2\pi]$

- Procedure
    1. Pick M numbers from $[-2\pi, 2\pi] \rightarrow x_{batch}$
    2. $F(x_{batch}) \rightarrow y^{(pred)}$ or $L(F, f) = \frac{1}{M}\sum_{i=0}^{M-1}((y_i^{(pred)}) - cos(x_i))^2$
    3. Gradient descent on L to find w_i, b_i, v_i such that L is minimized

$$x : (M, 1)$$
$$w : (1, N)$$
$$v : (N, 1); v = \begin{pmatrix} v_0 \\ v_1 \\ ... \\ v_{N-1} \end{pmatrix}$$
$$b : (N, 1); b = [b_0, b_1, b_2, ..., b_{N_1}]$$

Linear Algebra Review

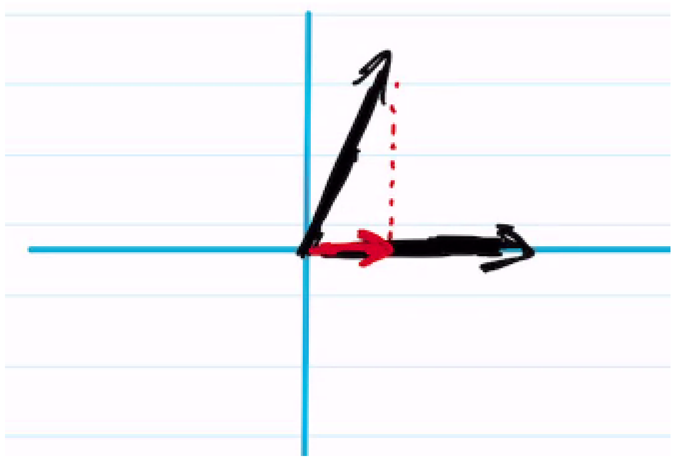
Dot Product

- What is the dot product actually doing?
    - It is a measure of similarity between 2 vectors, x and y.
        - Parallel vectors maximize similarity
        - Antiparallel vectors
        - Perpendicular Vectors

$$\vec{x} \cdot \vec{y} = x_o y_o + x_1 y_1 + ... + x_{n-1} y_{n-1} = \sum_{i=0}^{N-1} x_i y_i = ||\vec{x}|| \times ||\vec{y}|| cos(\theta)$$

Dot Products as Projection

- Dot products can be seen visually as the horizontal distance that two vector components share:



- Due to dot products being used as a measure of similarity, the adjusted weights of a neural network are inherently determining how similar the input is to the weight and, therefore, outputting values based on that similarity

$$y = \sum_{i=0}^{N-1} \vec{v}_i \phi(\vec{x}_i \cdot \vec{w}_i + b_i)$$

  - Where $\vec{x}_i \cdot \vec{w}_i$ determines the similarity between the input and the (constantly adjusted) weights, b_i determines a similarity threshold (due to this factor shifting the function horizontally) required to activate a neuron using ϕ, and v_i scales the output