Introduction to Machine Learning (by Implementation)

Lecture 7: Neural Networks

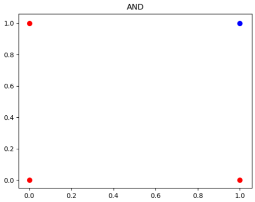
Ian J. Watson

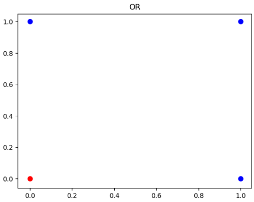
University of Seoul

University of Seoul Graduate Course 2019

KOREA RESEARCH FELLOWSHIP
해외 우수신진연구자 유치사업

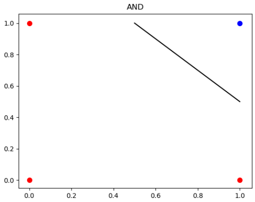
Some very simple examples for simple logistic regression

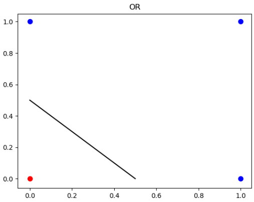




- Let's think about using logistic regression to approximate some simple binary functions
- OR and AND gates
  - OR is 0 (red) if both input are 0, 1 (blue) otherwise
  - AND is 1 if both inputs are 1, 0 otherwise
- Can we find logistic function approximations for this?
  - That is, $f(x_1, x_2)$ returns approximately 1 or 0 at the indicated points

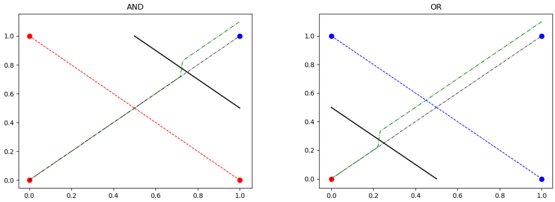
Some very simple examples for simple logistic regression





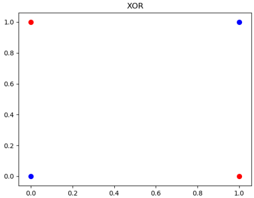
- Let's think about using logistic regression to approximate some simple binary functions
- OR and AND gates
  - OR is 0 (red) if both input are 0, 1 (blue) otherwise
  - AND is 1 if both inputs are 1, 0 otherwise
- Can we find logistic function approximations for this?
  - That is, $f(x_1, x_2)$ returns approximately 1 or 0 at the indicated points
- Yes! Take the projection perpendicular to the line

Some very simple examples for simple logistic regression



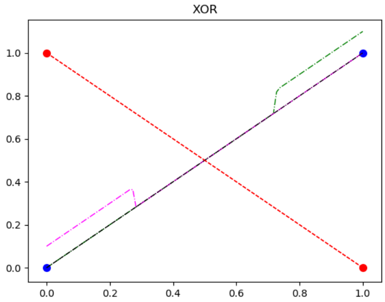
- Let's think about using logistic regression to approximate some simple binary functions
- OR and AND gates
  - OR is 0 (red) if both input are 0, 1 (blue) otherwise
  - AND is 1 if both inputs are 1, 0 otherwise
- Can we find logistic function approximations for this?
  - That is, $f(x_1, x_2)$ returns approximately 1 or 0 at the indicated points
- Yes! Take the projection perpendicular to the line
- and have the logistic turn on at the line
  - e.g. $f(x_1, x_2) = \sigma(2x_1 + 2x_2 - 1)$ for OR,
    $f(x_1, x_2) = \sigma(2x_1 + 2x_2 - 3)$ for AND [σ is our logistic function]

Very simple example with issues for Logistic Regression



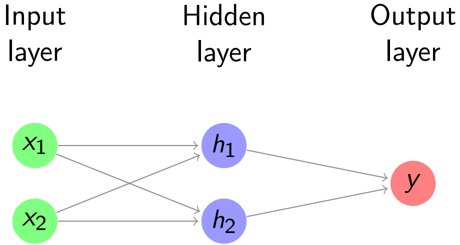
- Now consider the XOR gate: 1 if both inputs are the same, 0 otherwise
- The XOR gate can't be generated with a logistic function!
- Try it: no matter what line you draw, can't draw a logistic function that turns on only the blue!

How to Fix: more logistic curves!



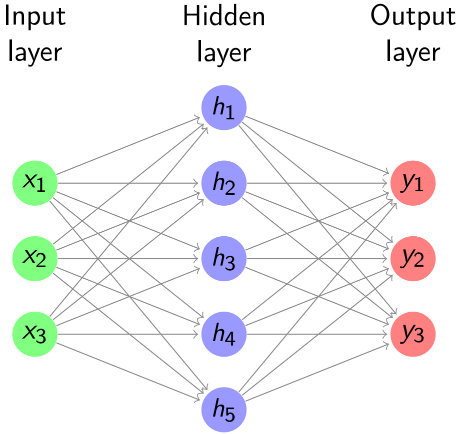
- Can fix by having 2 turn-on curves, one turning on either of the blue points, then summing the result
- $f(x_1, x_2) = \sigma(2x_1 + 2x_2 - 1) + \sigma(-2x_1 - 2x_2 + 1)$

The Feed-Forward Neural Network



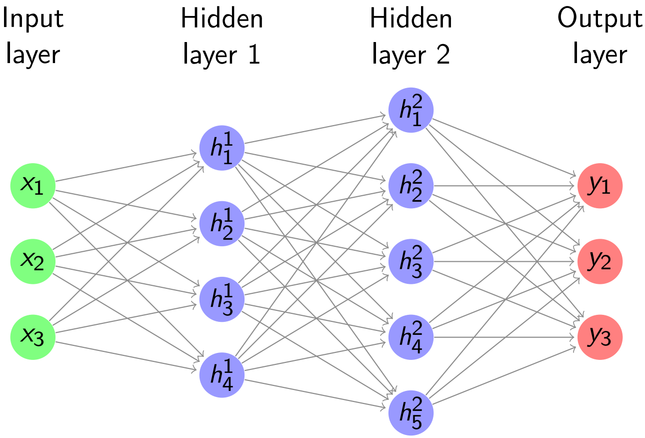
- Consider the structure of what we just made
  - $y = f(x_1, x_2) = \sigma(-1 + 2x_1 + 2x_2) + \sigma(1 - 2x_1 - 2x_2)$
- Decompose the function into:
  - the *input layer* of $\hat{x}$,
  - the *hidden layer* which calculates $h_i = \beta_i \cdot x$ then passes if through the *activation function* σ, (called "sigmoid" in NN terms)
    - as in logistic, there is an extra β_0, called the *bias*, which controls how big the input into the node must be to activate
  - the *output layer* which sums the results of the hidden layer and gives y
    - $y = 0 + 1 \cdot \sigma(h_1) + 1 \cdot \sigma(h_2)$

Feed-Forward Neural Network



- In general, we could have several input variables, and output variables
- In the case of classification, we would usually have a final *softmax* applied to $\hat{y}$, but could use any *activation* φ here also
  - *softmax* generalization of our multinomial logistic regression

Feed-Forward Neural Network



- We can even have several hidden layers
  - The previous layer acts the same as an *input layer* to the next layer
- We call each node in the network a *neuron*

Universal Approximation Thereom

Let $\varphi : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded, and continuous function. Let I_m denote the m-dimensional unit hypercube $[0, 1]^m$. The space of real-valued continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, \ldots, N$ such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi \left(w_i^T x + b_i \right)$$

as an approximate realization of the function f; that is,

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

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$. This still holds when replacing I_m with any compact subset of $\mathbb{R}^m$.

- In brief: with a hidden layer (of enough nodes), any (sensible) function $f : \mathbb{R}^m \to \mathbb{R}$ can be approximated by a feed-forward NN
    - Any (sensible) activation φ can work, not just σ

Exercises

- Today, we will just try to come up with a format for neural networks, and implement the XOR network by hand
- A neural network will be a list (of layers) of lists (nodes) of lists (node weights)
  - `[ [[1,2,2], [1,-2,-2]], [[0,1,1]] ]` is a one hidden layer network of 2 hidden nodes, there are two inputs, the first node calculates $-1 + 2x_1 + 2_x 2$, the second calculates $1 - 2x_1 - 2x_2$, the the output layer gives $0 + h_1 + h_2$
- We will need some neuron calculators
  - `inner_neuron(weight, input)` which calculates what we called `inner` last time, where `weights` are `beta` and `input` is `x` (with an extra bias weight) (you can just copy this function)
  - `sigmoid_neuron(weights, input)` does `inner_neuron` and then passes it through `sigmoid` (`logistic_fn` from last time)

Exercises

- `feedforward_(network, input_vector, hidden_neuron=sigmoid_neuron, output_neuron=inner_neuron)`
    - split the network into hidden layers, and the output layer
    - pass the `input_vector` through the hidden layers applying the `hidden_neuron` (nb could be more than one), storing the results
    - pass the result through the `output_layer` applying `output_neuron`
    - return a list of lists of the hidden layer values, and the output layer value
- `feedforward(network, input_vector, hidden_neuron=sigmoid_neuron, output_neuron=inner_neuron)`
    - Run `feedforward_` but drop the hidden layer values
- What I showed earlier was actually the XNOR gate (0,0) and (1,1) are 1 and (0,1), (1,0) are 0. Construct the XOR logic gate as a neural network where (0,1), (1,0) are 1 and (0,0) and (1,1) are 0
    - See the `neural.py` for the implementation of XNOR, note that I've multiplied by large numbers to force the sigmoid hidden layers to return nicer values