Math 345 - Notes
Neural Networks I
November 19, 2018

# Neural Networks I

Neural networks are used for non-linear classification. We have seen that for data that can be linearly separated into two categories, one can use Logistic regression or the PLA algorithm. In fact, the PLA algorithm simulates the event where the network has one neuron, so it can be thought of as a very special case of a neural network.

Rather than using and function with output in $\{0, 1\}$ or $\{-1, +1\}$, which has a hard threshold, we will work with a sigmoid function which is differentiable. Thus, making small changes in parameters of the network would lead to small changes in the output. The neural network will then be based on sigmoid neurons. Let us investigate some properties of the logistic function $\theta(x)$ :

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

Recall that the output of the sigmoid represents a probability, in this case the probability that the neuron should fire. If the probability is larger than 0.5, we would set the neuron to fire. Thus, the boundary of this sigmoid classification curve occurs when $\theta(x) = 0.5$, that is, at $x = 0$. However, close to $x = 0$, confidence in correct labeling is low. For higher confidence, let us look at the value of $x$ for which
- $\theta(x) = 0.99 \;\Rightarrow\; \frac{1}{1+e^{-x}} = 0.99 \;\Rightarrow\; x = 4.6$
- $\theta(x) = 0.01 \;\Rightarrow\; \frac{1}{1+e^{-x}} = 0.01 \;\Rightarrow\; x = -4.6$

Thus, for a set of weights $\mathbf{w}$ and input vector $\mathbf{x}$, if $\mathbf{w}^T\mathbf{x} > 4.6$ we label the neuron with $+1$ (fire) and if $\mathbf{w}^T\mathbf{x} < -4.6$ we label the neuron with 0 (do not fire). In between, we can label as usual by comparing with the boundary value, but confidence in labeling will be low.

The goal of this set of notes is to describe a neural network that allows us to label the output of an $XOR$, or exclusive or. Recall that $XOR$ can be described by the truth table (1 =true, 0 =false)

| $x_1$ | $x_2$ | $y = XOR(x_1, x_2)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

This set cannot be linearly separated. But we can use the laws of logic to derive a multi-step sequence of logic gates, each being a perceptron. Here are two possible ways to re-write $XOR$

1.

$$\begin{aligned} XOR(x_1, x_2) &= \neg((x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)) = \neg(x_1 \wedge x_2) \wedge (x_1 \vee x_2) \\ &= AND(NAND(x_1, x_2), OR(x_1, x_2)) \end{aligned}$$

2.

$$\begin{aligned} XOR(x_1, x_2) &= (x_1 \wedge \neg x_2) \vee (x_2 \wedge \neg x_1) \\ &= OR(AND(x_1, NOT(x_2)), AND(x_2, NOT(x_1))) \end{aligned}$$

As we will show now, $AND, OR, NAND, NOT$ can be described by perceptrons. Since data for these logic gates is linearly separable, we can find a set of weights $\mathbf{w} = [w_0, w_1, w_2]$ such that $\theta(w_0 + w_1 x_1 + w_2 x_2) \approx$ logic gate$(x_1, x_2)$. One can use the PLA algorithm or logistic regression, but we will guess a set of weights that are large enough to output a value close to 0 or 1, and we will try to keep them symmetric $w_1 = w_2$.

(a) $AND$ gate:

| $x_1$ | $x_2$ | $y = AND(x_1, x_2)$ | $\theta(-30 + 20x_1 + 20x_2)$ |
|---|---|---|---|
| 0 | 0 | 0 | $\theta(-30) = 0$ |
| 0 | 1 | 0 | $\theta(-10) = 0$ |
| 1 | 0 | 0 | $\theta(-10) = 0$ |
| 1 | 1 | 1 | $\theta(+10) = 1$ |

As we have seen in class, the following set of weights work (see output in red)

$AND(x_1, x_2) = \theta(-30 + 20x_1 + 20x_2)$.

(b) $OR$ gate:

| $x_1$ | $x_2$ | $y = OR(x_1, x_2)$ | $\theta(-10 + 20x_1 + 20x_2)$ |
|---|---|---|---|
| 0 | 0 | 0 | $\theta(-10) = 0$ |
| 0 | 1 | 1 | $\theta(+10) = 1$ |
| 1 | 0 | 1 | $\theta(+10) = 1$ |
| 1 | 1 | 1 | $\theta(+30) = 1$ |

$OR(x_1, x_2) = \theta(-10 + 20x_1 + 20x_2)$.

(c) $NAND$ gate:

| $x_1$ | $x_2$ | $y = NAND(x_1, x_2)$ | $\theta(30 - 20x_1 - 20x_2)$ |
|---|---|---|---|
| 0 | 0 | 1 | $\theta(+30) = 1$ |
| 0 | 1 | 1 | $\theta(+10) = 1$ |
| 1 | 0 | 1 | $\theta(+10) = 1$ |
| 1 | 1 | 0 | $\theta(-10) = 0$ |

$NAND(x_1, x_2) = \theta(30 - 20x_1 - 20x_2)$.

(d) $NOT$ gate:

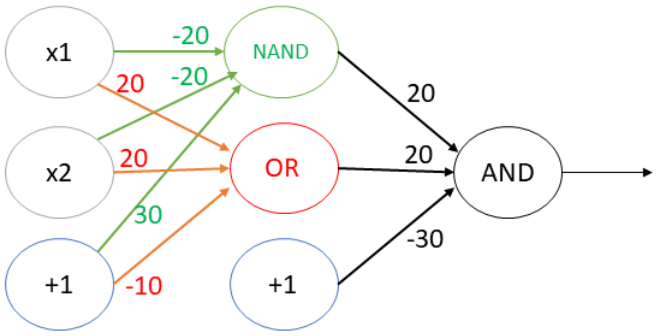
| $x_1$ | $y = NOT(x_1)$ | $\theta(10 - 20x_1)$ |
|---|---|---|
| 0 | 1 | $\theta(+10) = 1$ |
| 1 | 0 | $\theta(-10) = 0$ |

$NOT(x_1) = \theta(10 - 20x_1)$.

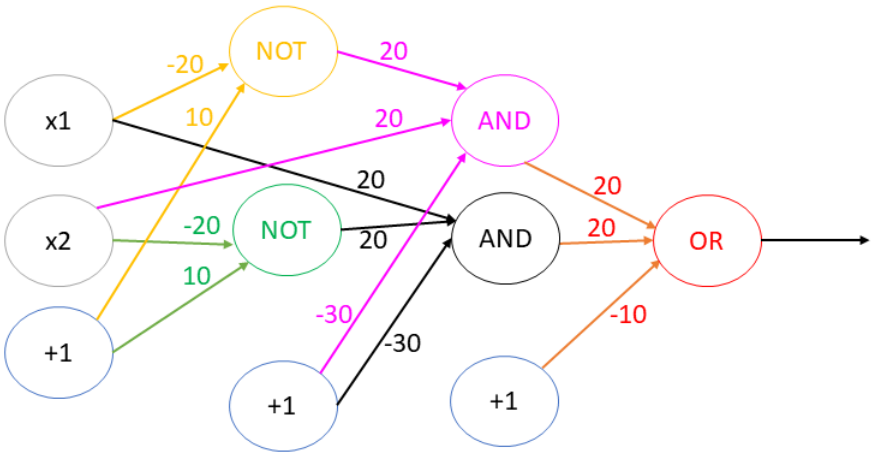
(e) Can you find weights for the $NOR = NOT(OR(x_1, x_2))$ logic gate?

Now we are ready to build the neural network for $XOR$, using the two logic expressions discussed before:

1. $XOR(x_1, x_2) = AND(NAND(x_1, x_2), OR(x_1, x_2))$ uses one hidden layer:




2. $XOR(x_1, x_2) = OR(AND(x_1, NOT(x_2)), AND(x_2, NOT(x_1)))$ uses two hidden layers:




In this second neural network, we can introduce the identity operator, so that only units from the same hidden layer are used.