6.034 Exam 3 Cheat Sheet

Neural Nets

Useful Information

1) Neural Nets are numerical classifiers with binary (`0/1`) output
2) The neuron is a primitive circuit element
3) Forward propagation computes the overall output of a neural net

(Input Layer) -> (Logic Function Layers) -> Output (`0/1`)

A single neuron can draw one line and shade above or below it

Primitive Logic Functions Computable by a Single Neuron

Note: used in the **logic** layer

- `AND(x,y)`

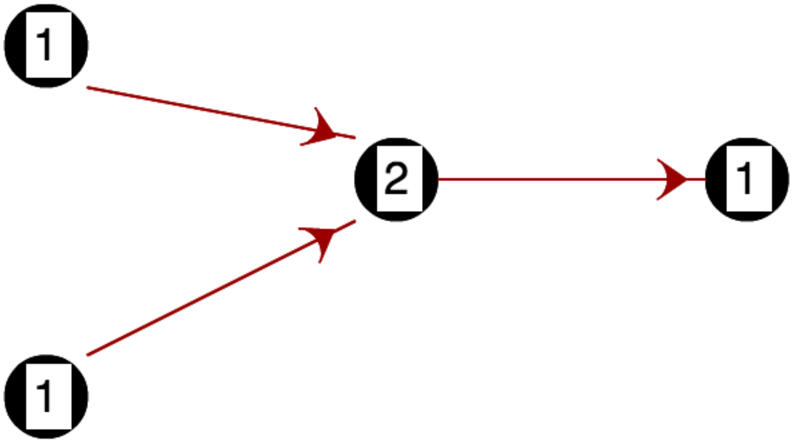


Figure 1: AND

- `OR(x,y)`

- `NOT(x, _)`

* note, the circle on the line means that the weight is `-1`)

- "`MAJORITY(x1, x2, x3, x4, ...)`" (3 input example)

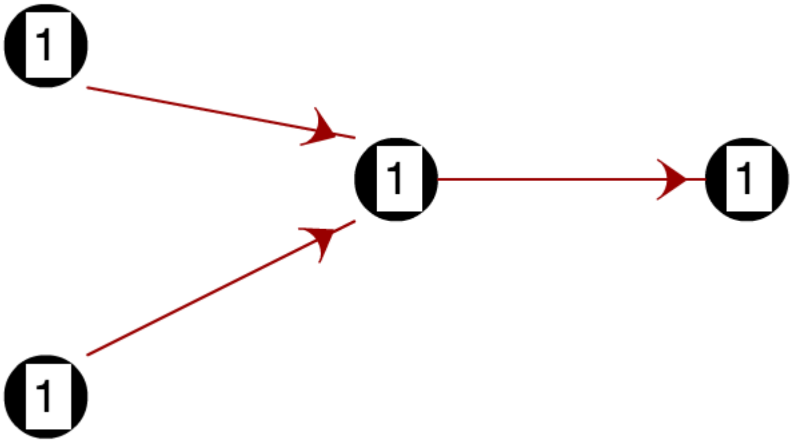


Figure 2: OR

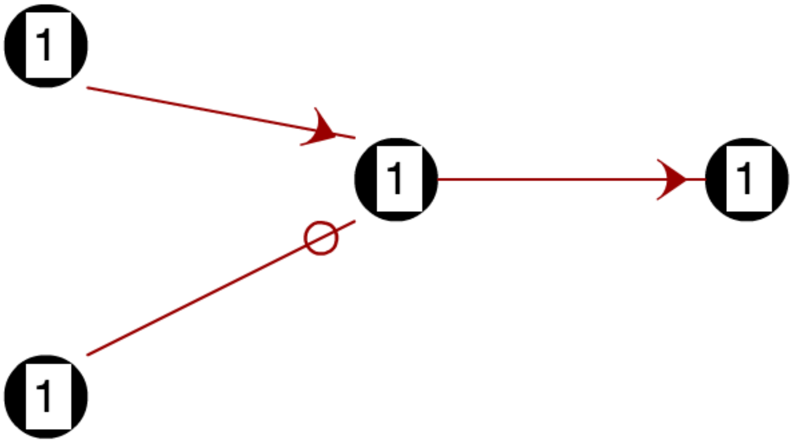
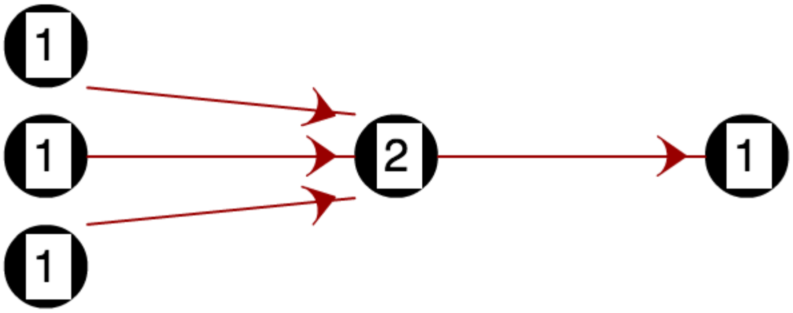


Figure 3: NOT



- note, doubling the weight of the bottom input (`x3` for instance) makes this gate act like `OR(AND(x1, x2), x3)`

Helper Functions

$StairstepT(x) =$

$$\begin{cases} 1 & \text{if } x \geq T \\ 0 & \text{if } x < T \end{cases}$$

$SigmoidS,\ M(x) =$

$$\frac{1}{1+e^{-S(x-m)}}$$

$Performance\ =\ Accuracy(out*, out)\ =\ \frac{1}{2}(out*-out)^2$

- * means **desired** output

Quick Formulas For Backward Propagation

$$W'_{A\to B} = W_{A\to B} + \Delta W_{A\to B}$$
$$\Delta W_{A\to B} = r \cdot out_A \cdot \delta_B$$

$$\delta_B = \begin{cases} out_B(1-out_B)(out*-out) & \text{if neuron B is in final (output) layer} \\ out_B(1-out_B)\sum_{outgoingC_i} W_{B\to C_i}\delta_{C_i} & \text{if neuron B is not in final (output) layer} \end{cases}$$

Backwards Propagation Steps

1. Computing output of each neuron using forward Propagation and $Stairstep_T$ function
2. Compute δ_B for final layer
3. Compute δ_B for earlier layers
4. Compute updates for weights
5. Update all weights

Miscellaneous Notes

- You can never classify all points correctly if you have a + data point and a − data point (contraditory) right on top of each other

Overfitting - too strict with regards to the data it's trying to model

Underfitting - too simple with regards to the data it's trying to model

Support Vector Machines

Useful Information

- like Neural Nets, classifies numerical data into two classes: + and −
- draws the decision boundary line that separates the training data with the widest possible margin

Boundaries

- 1-D - just a point
- 2-D - some sort of line or curve
- 3-D - some sort of plane

How to Draw SVM Boundaries (2D)

1. Draw the *convex hulls* for the + and − training points.

- a convex hull is the shape you get when you wrap a rubber band around the points and let it contract

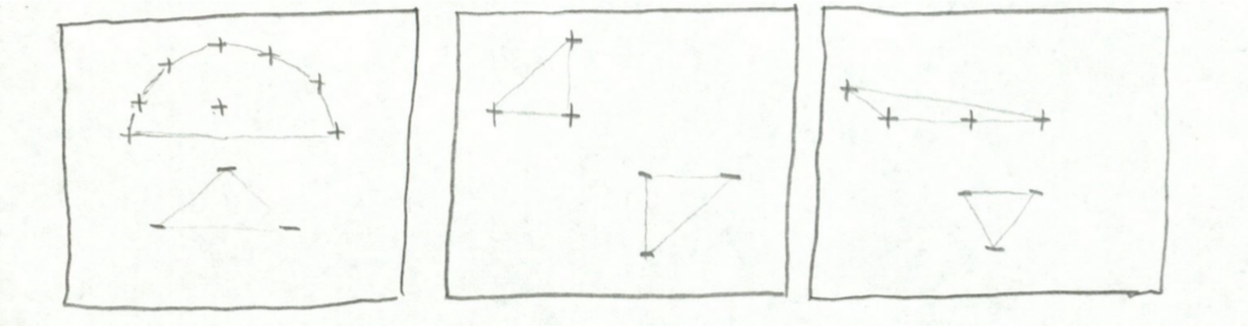
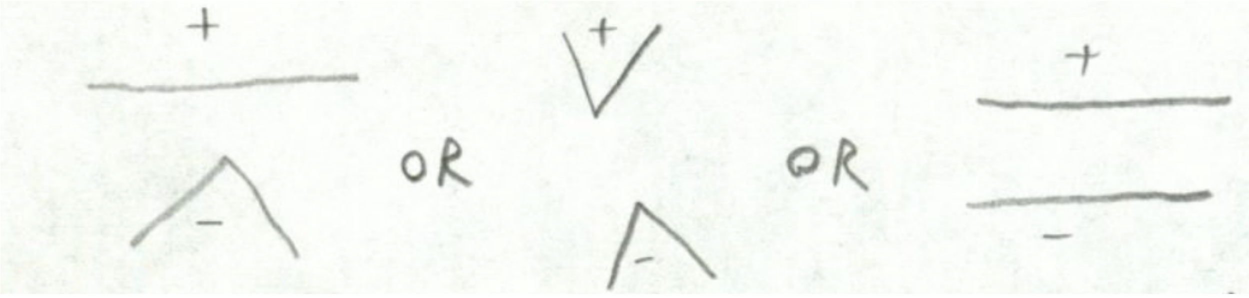


Figure 4: Convex Hull Examples

2. Look at the regions where the convex hulls are closest.
   3 Cases:



3. The corresponding boundaries look like this: