(Artificial) Neural Networks

Details and Examples

Ashkan Yousefpour

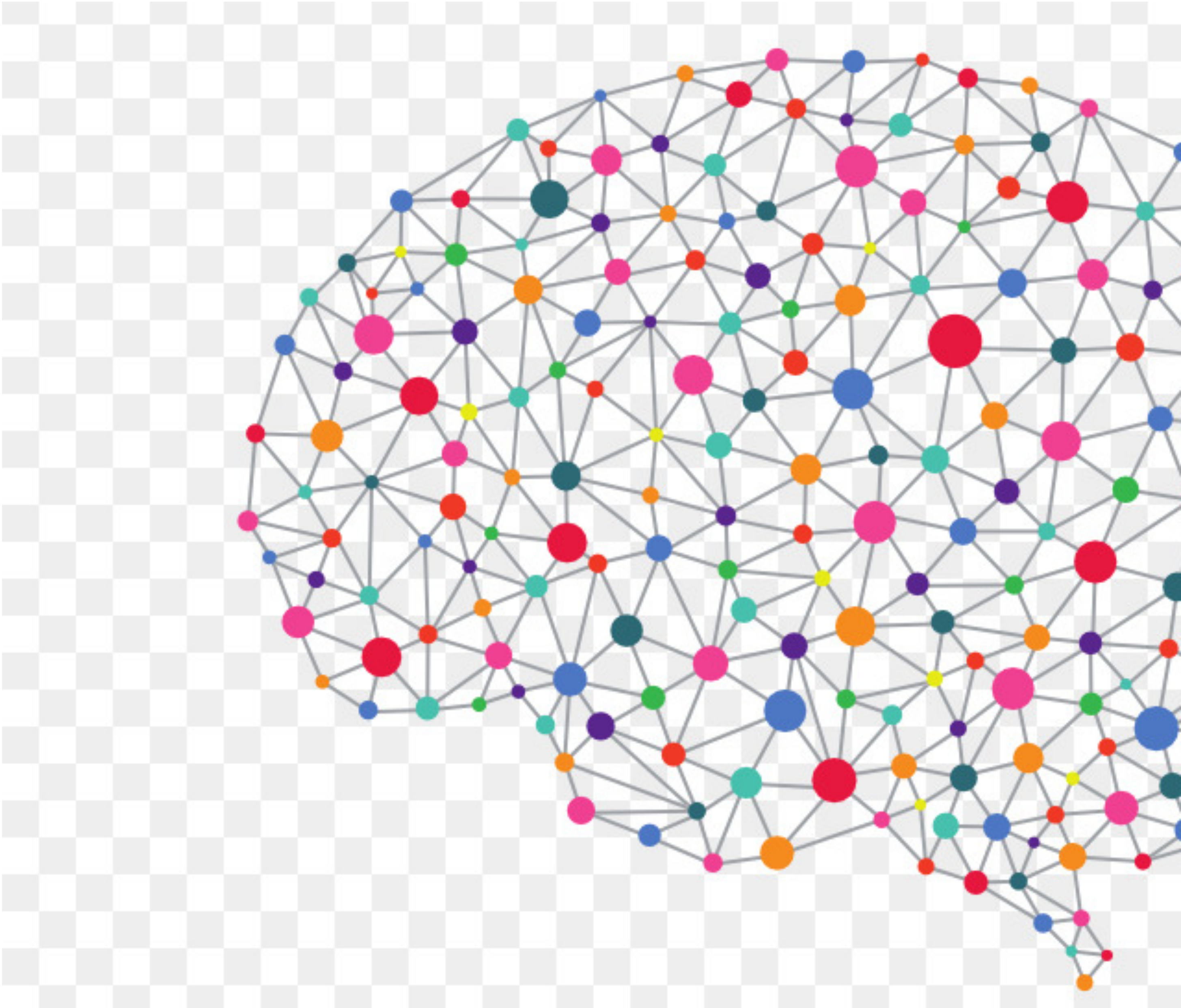
Computer Science
University of Texas at Dallas

CS7301-003 Fall 2018

September, 2018

Outline

- Introduction
- Perceptron
    - Activation Functions
    - Exercise
    - Training Rule
    - Gradient Descent
        - Exercise
- Artificial Neural networks
    - Different Types
    - Exercises
- Back propagation
    - Exercise



Introduction

- Artificial Neural Networks (ANNs) provide interesting alternatives of solving variety of problems in different fields of science and engineering
- Human brain
  - Ultimate goal of a computer scientist is to create a computer that could mimic human brain (e.g. biological neural network)
  - ANNs are simplifications of Biological Neural Networks
- ANNs have proven their applicability and importance by solving complex problems (e.g. emergence of deep neural networks, “deep learning”)

Motivation for this Lecture

By the end of this lecture,
we will be able to solve some concrete exercises like this one

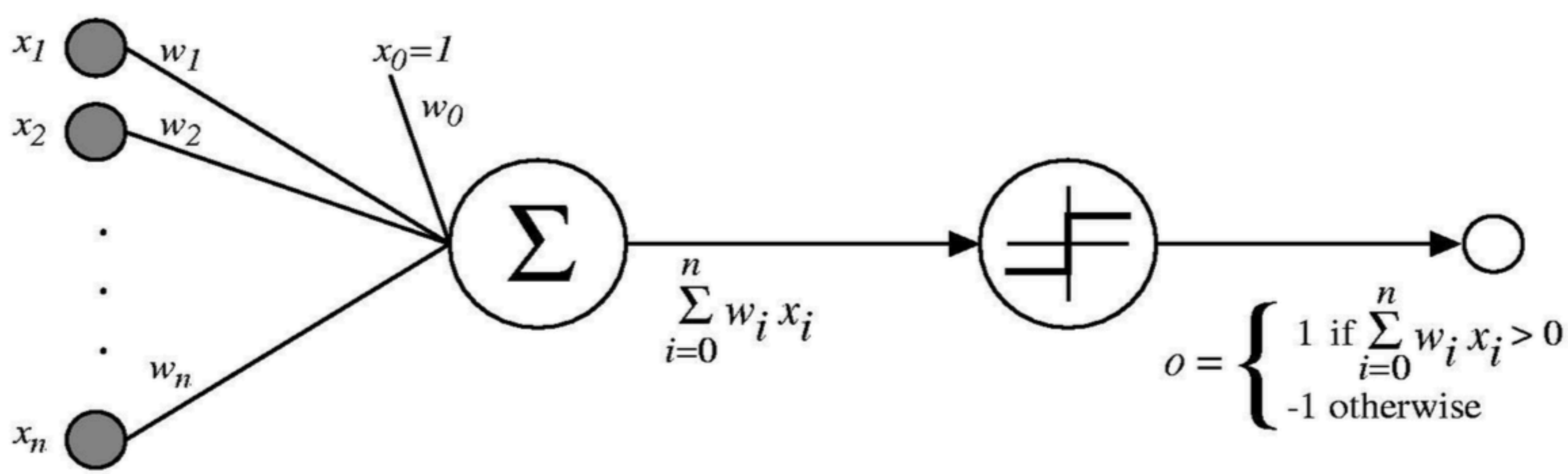
Exercise 1

Draw a neural network that represents the function $f(x, y)$ defined below:

x	y	$f(x, y)$
0	0	10
0	1	-5
1	0	-5
1	1	10

ANN Building Block

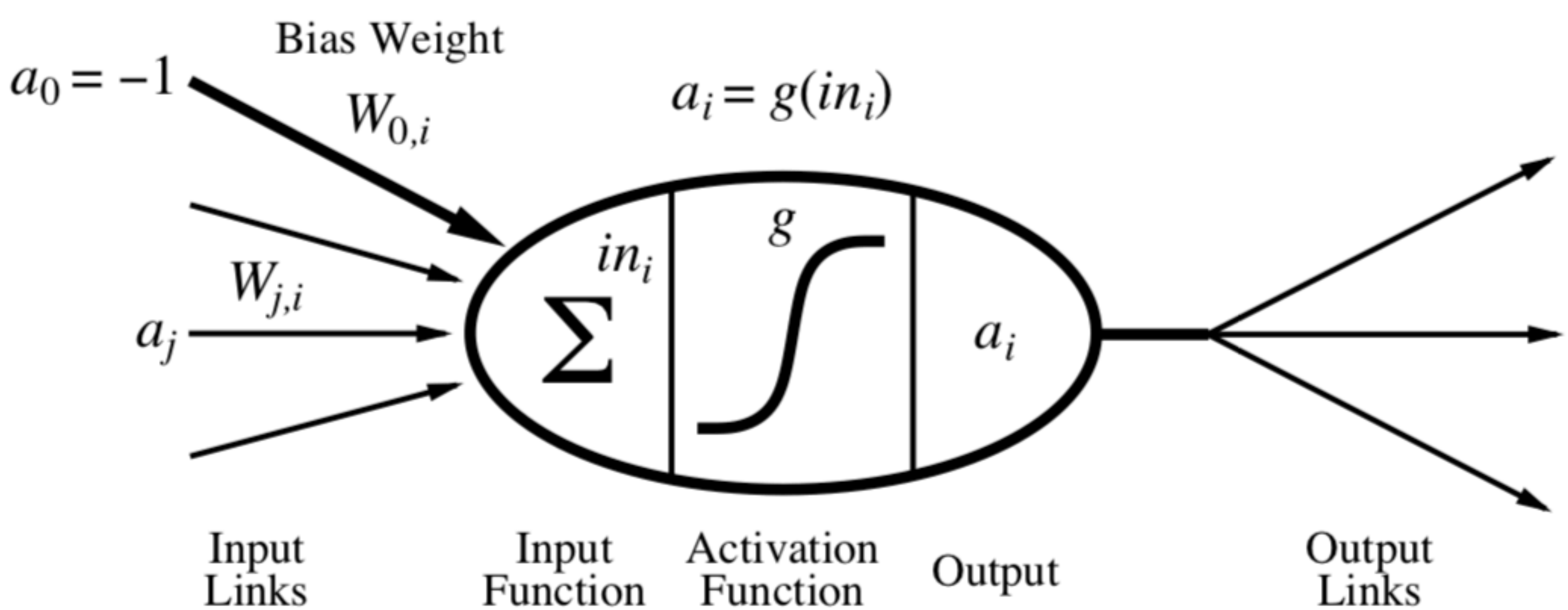
- The main component of ANN is ***perceptron***
- ANN is a combination of many perceptrons, connected in a bigger network
- Perceptron with **step** activation function



Picture borrowed from https://www.hlt.utdallas.edu/~vgogate/ml/2018s/lectures/Perceptrons.pdf

Perceptron

- Usually in ANN, the linear unit (sum) and activation unit are shown in one circle



Picture borrowed from http://aima.eecs.berkeley.edu/slides-pdf/chapter20b.pdf

Perceptron Example

- Spam Detection
- 3 features (frequency of words “money”, “lottery”, and bias)
- spam is “positive” class
- Current weights $(w_0, w_1, w_2) = (-3,4,2)$
  - Email is “win lottery money” -> spam

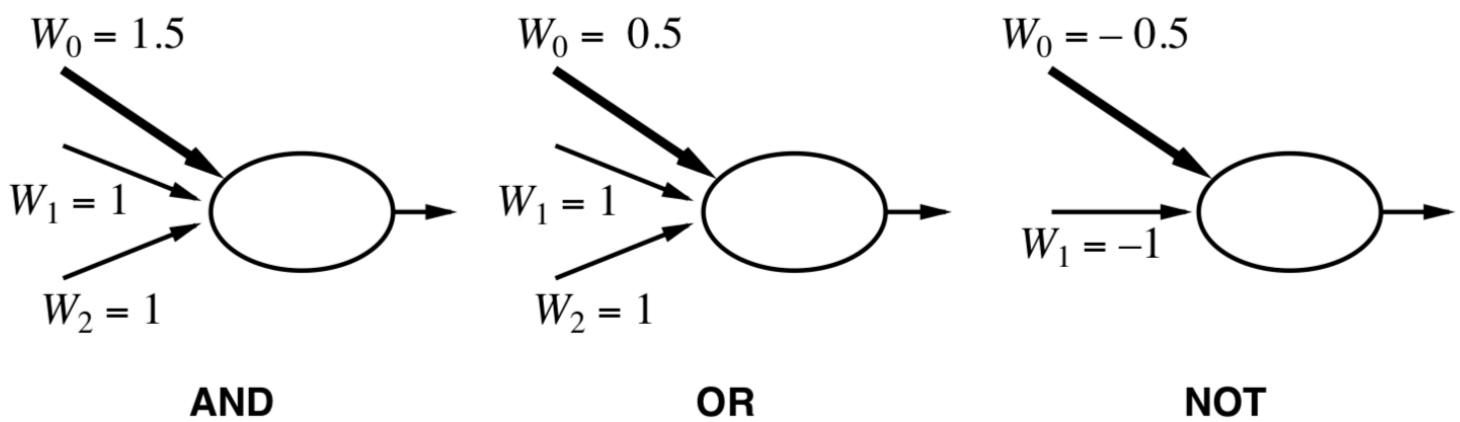
$$W.X = (1)(-3) + (1)(6) + (1)(2) = 5 > 0$$ **Spam!**

Perceptron Activation Functions

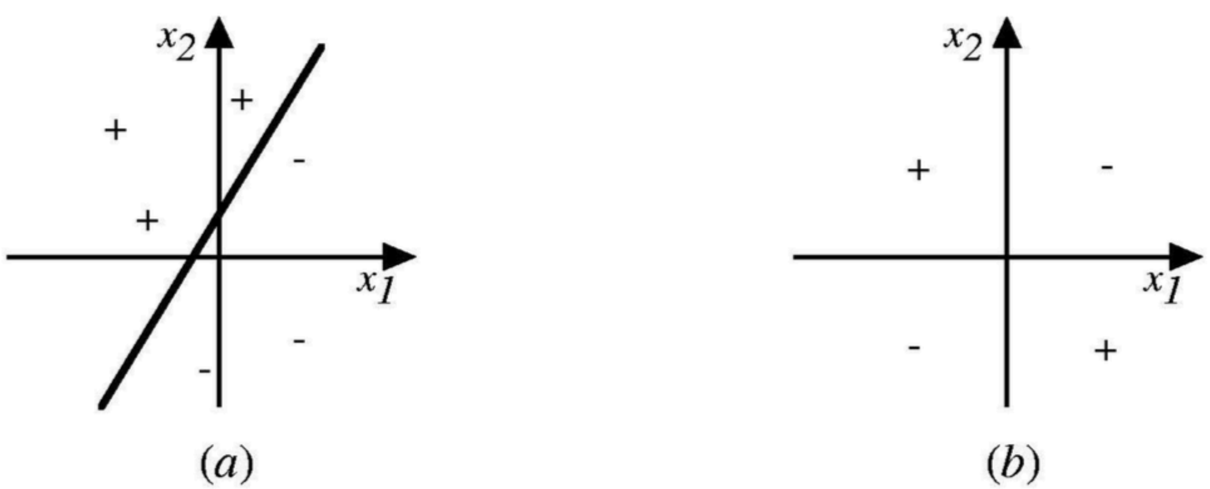
- Activation functions:
    - Identity function
    - Step function
    - Sigmoid function (aka “logistic”)
    - ReLU function
    - See https://en.wikipedia.org/wiki/Activation_function

Perceptron implementable Functions

- **Exercise**: Implement NOT, AND, and OR using perceptron
- Linear functions can be implemented with perceptron (e.g. AND)



- Decision surface of perceptron is hyperplane (line in 2D)



Perceptron Training

- We found a perceptron for AND, OR, NOT
- How about bigger examples, e.g. optical network reconfiguration plan given 200 features?
- How can computer find the weights automatically?

Perceptron Training Rule

- Training rule:

$$\Delta w_i = \eta(t - o)x_i$$

$$w_i \leftarrow w_i + \Delta w_i$$

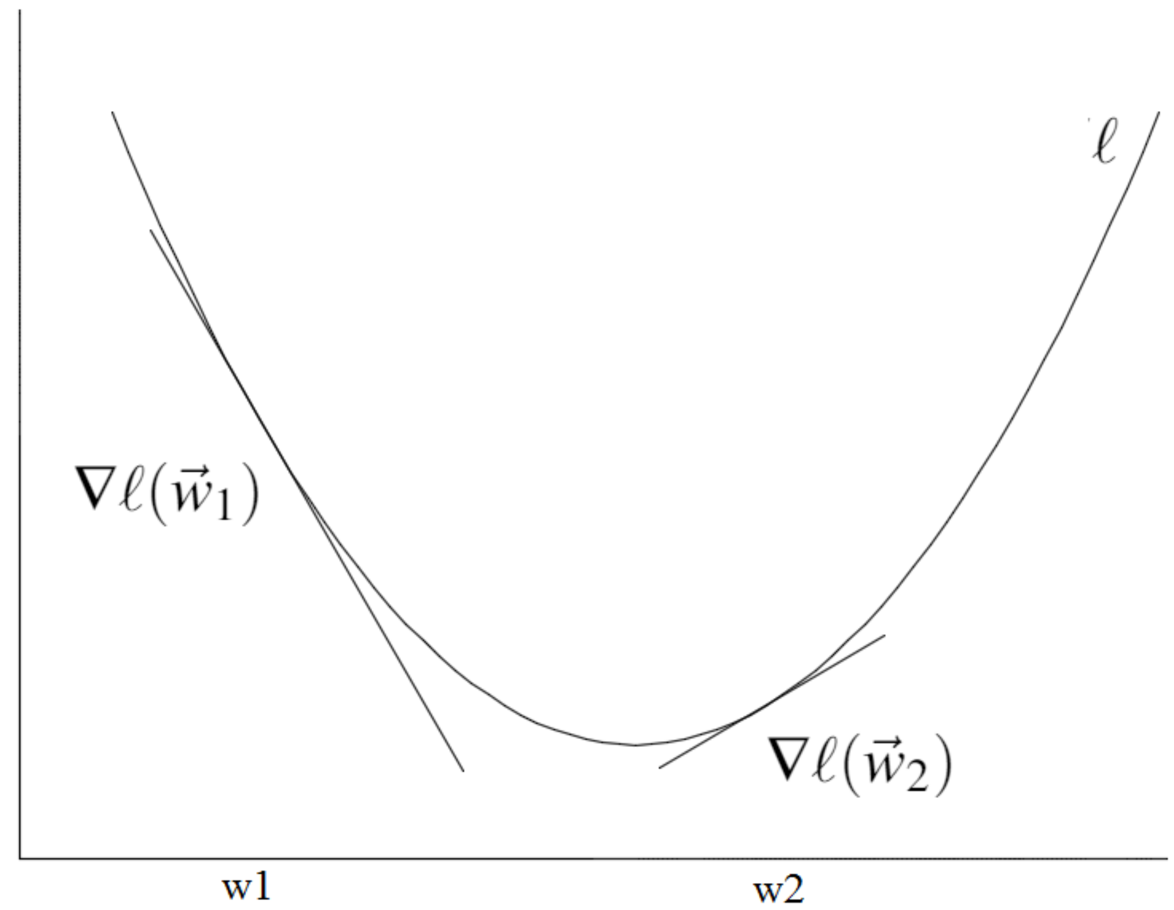
- η is learning rate (constant, e.g. 0.1)
- o is the output of perceptron, including activation function
- t is target value (desired)

Perceptron Training Rule

- Perceptron training rule is great
- However, what happens if data is **not** linearly separable
    - Goes back and forth
    - Will **not** converge!
- Need another training rule
    - Gradient descent or gradient ascent

Gradient Descent

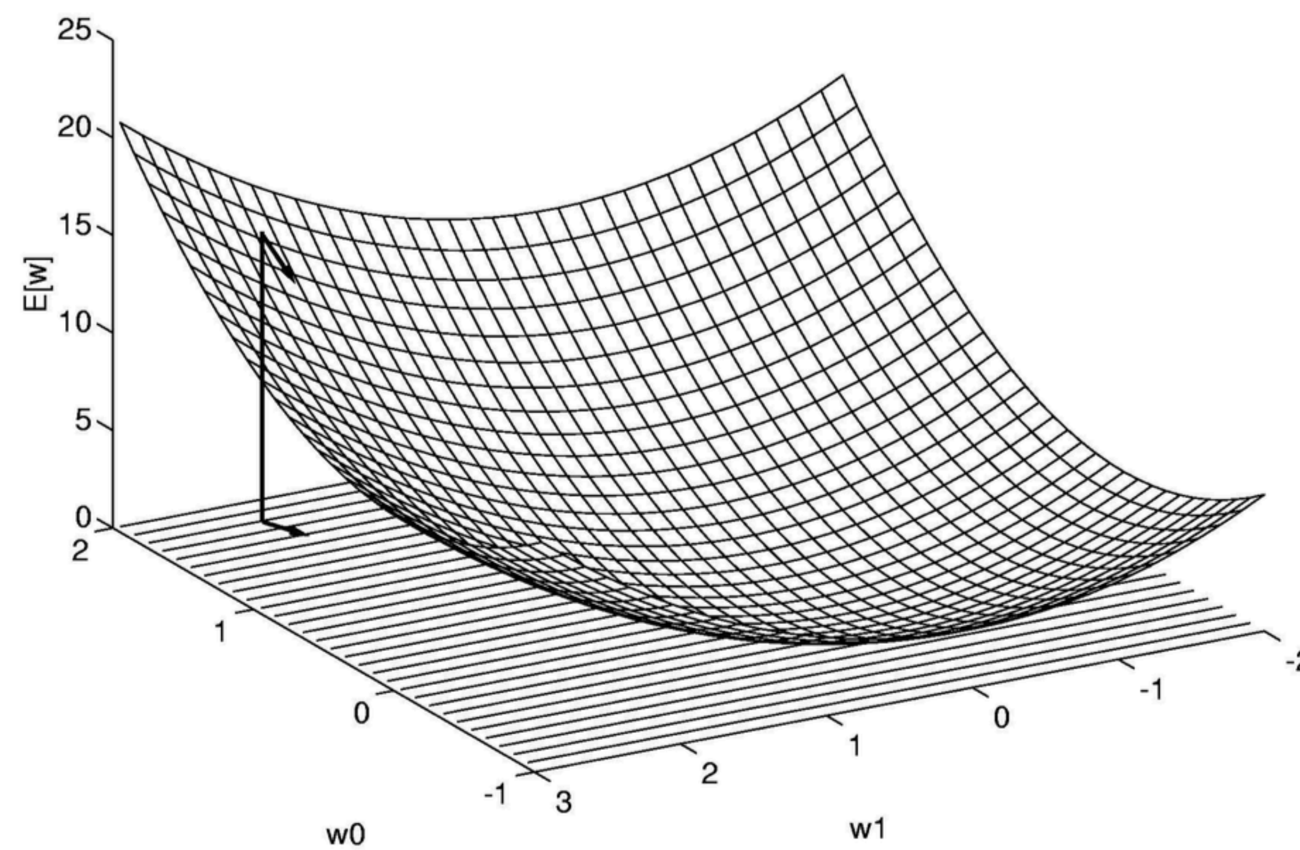
- **Gradient descent**
  - Let's think about error (or loss) function $l(W)$
  - Can we somehow get to the minima?
  - Yes. Using gradient $\nabla\, l[W]$



$$l[W] = E[W] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Gradient Descent

- **Gradient descent**
  - Error function E(W)
  - Start randomly from somewhere (in the E(W) surface)
  - Move downwards using gradient (will see soon)
  - Hopefully you get to global minima
    - Why not always?



Picture borrowed from https://www.hlt.utdallas.edu/~vgogate/ml/2018s/lectures/Perceptrons.pdf

Perceptron Gradient Descent

- Error function E(W)

$$E[W] = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

- D is set of examples (i.e. data)

- Gradient

$$\nabla E[W] = [\frac{dE}{dw_0}, \frac{dE}{dw_1}, \ldots, \frac{dE}{dw_n}]$$

- Training rule

$$\Delta w_i = -\eta \frac{dE}{dw_i}$$

- Gradient descent

Perceptron Gradient Descent

- **Exercise**: Derive gradient descents for
  - Activation: identity

$$\frac{dE}{dw_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

  - Activation: sigmoid

$$\frac{dE}{dw_i} = \sum_d (t_d - o_d)o_d(1 - o_d)(-x_{i,d})$$

Perceptron Gradient Descent

1. Initialize each w_i to some small random value
2. Until convergence do
    1. Initialize each Δw_i to zero
    2. for each example in training data do
        1. input the example x and compute output o
        2. for each linear unit weight w_i do

$$\Delta w_i \leftarrow \Delta w_i + \eta \frac{dE}{dw_i} \qquad \begin{cases} \Delta w_i \leftarrow \Delta w_i + \eta(t-o)x_i & \text{or} \\ \Delta w_i \leftarrow \Delta w_i + \eta(t-o)o(1-o)x_i \end{cases}$$

    3. for each linear unit weight w_i do

$$w_i \leftarrow w_i + \Delta w_i$$

Neural Networks

Neural Network: Connect perceptron (neurons) and make bigger structures

1. Feed-forward NN (ANN)

2. Recurrent Neural Network (RNN)

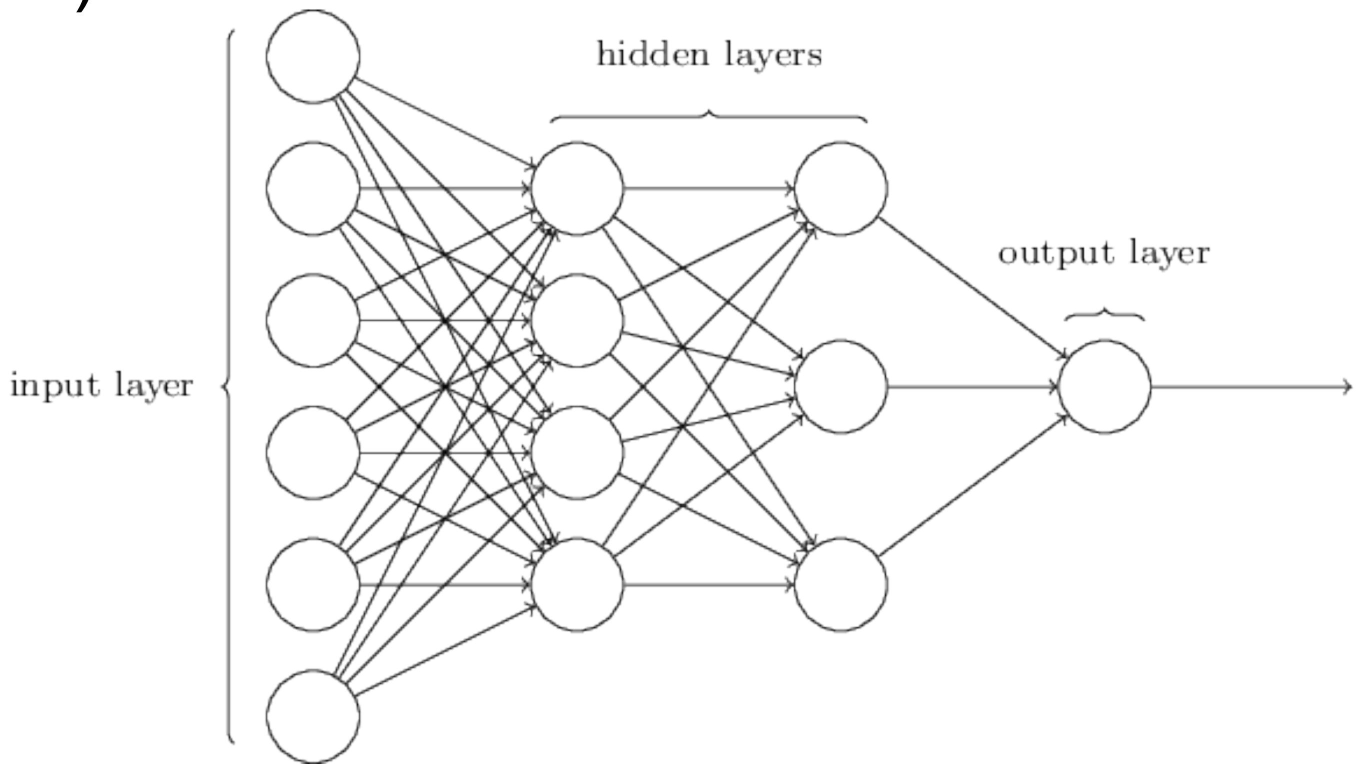
3. Convolutional Neural Networks (CNN)

Key learning algorithm: Back Propagation (BP)

A recent work: Dosovitskiy, Alexey, et al. "Flownet: Learning optical flow with convolutional networks." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2758-2766. 2015.

ANN

1. Feed-forward NN (ANN): one-direction, fully-connected
   1. Single-layer perceptron
   2. Multi-layer perceptron (MLP)
   3. Deep Neural Network (DNN)

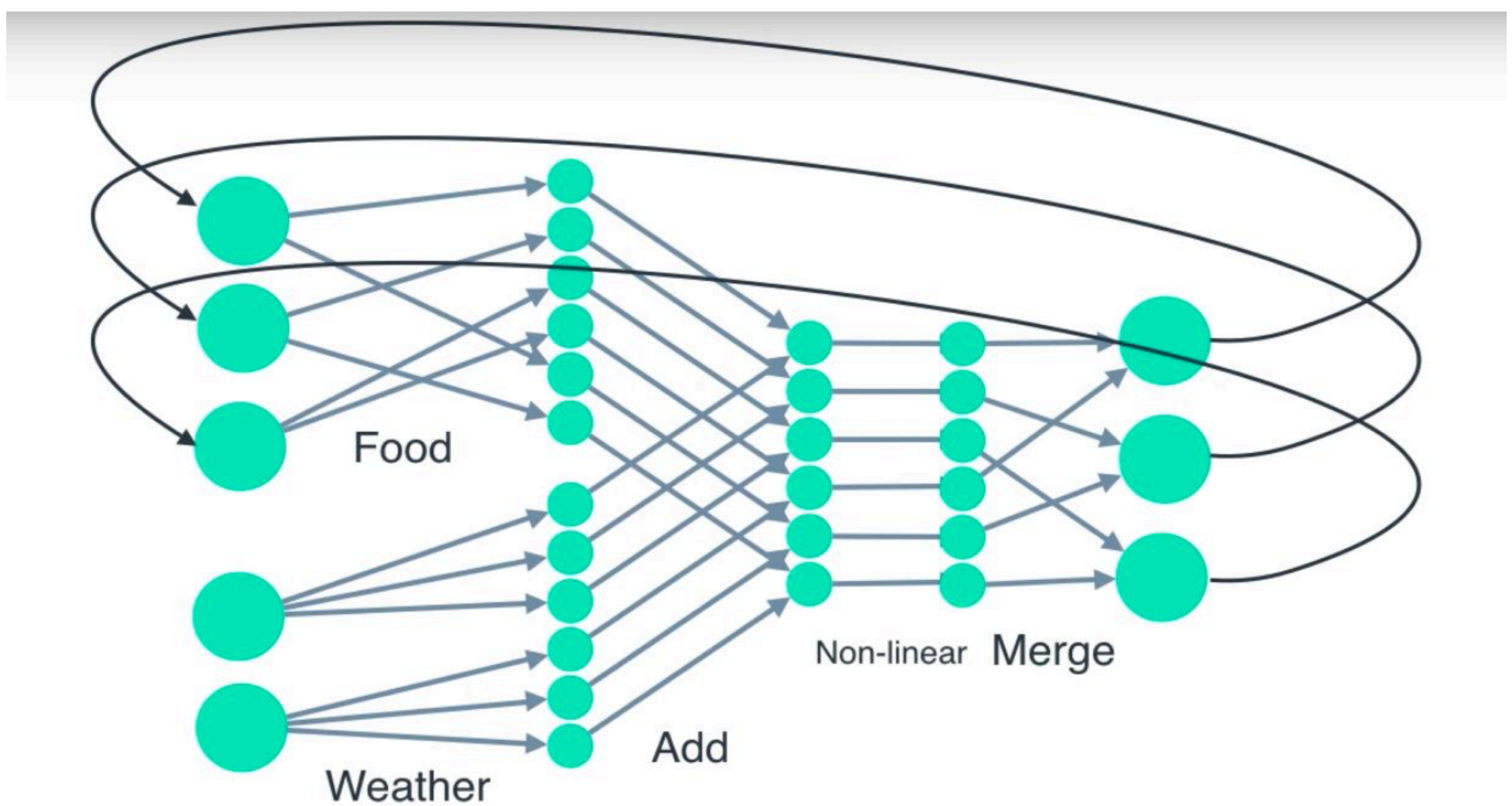


Picture borrowed from https://people.cs.pitt.edu/~xianeizhang/notes/NN/NN.html

RNN

2- Recurrent Neural Network (RNN)

- Directed cycles and delays
- Recognize pattern in time

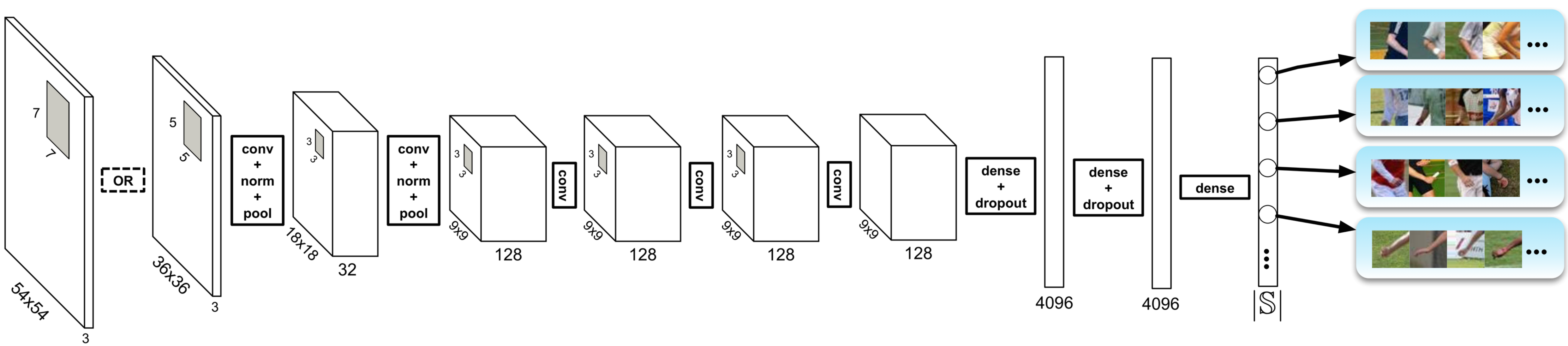


Picture borrowed from http://cseweb.ucsd.edu/~mkchandraker/classes/CSE291/Winter2018/Lectures/TrackingRNN.pdf

CNN

3- Convolutional Neural Networks (CNN)

- Not fully-connected. Connected in convolutions style
- Recognize pattern in space



Picture borrowed from http://www.stat.ucla.edu/~xianjie.chen/projects/pose_estimation/pose_estimation.html

Exercise

Now let's look at some examples

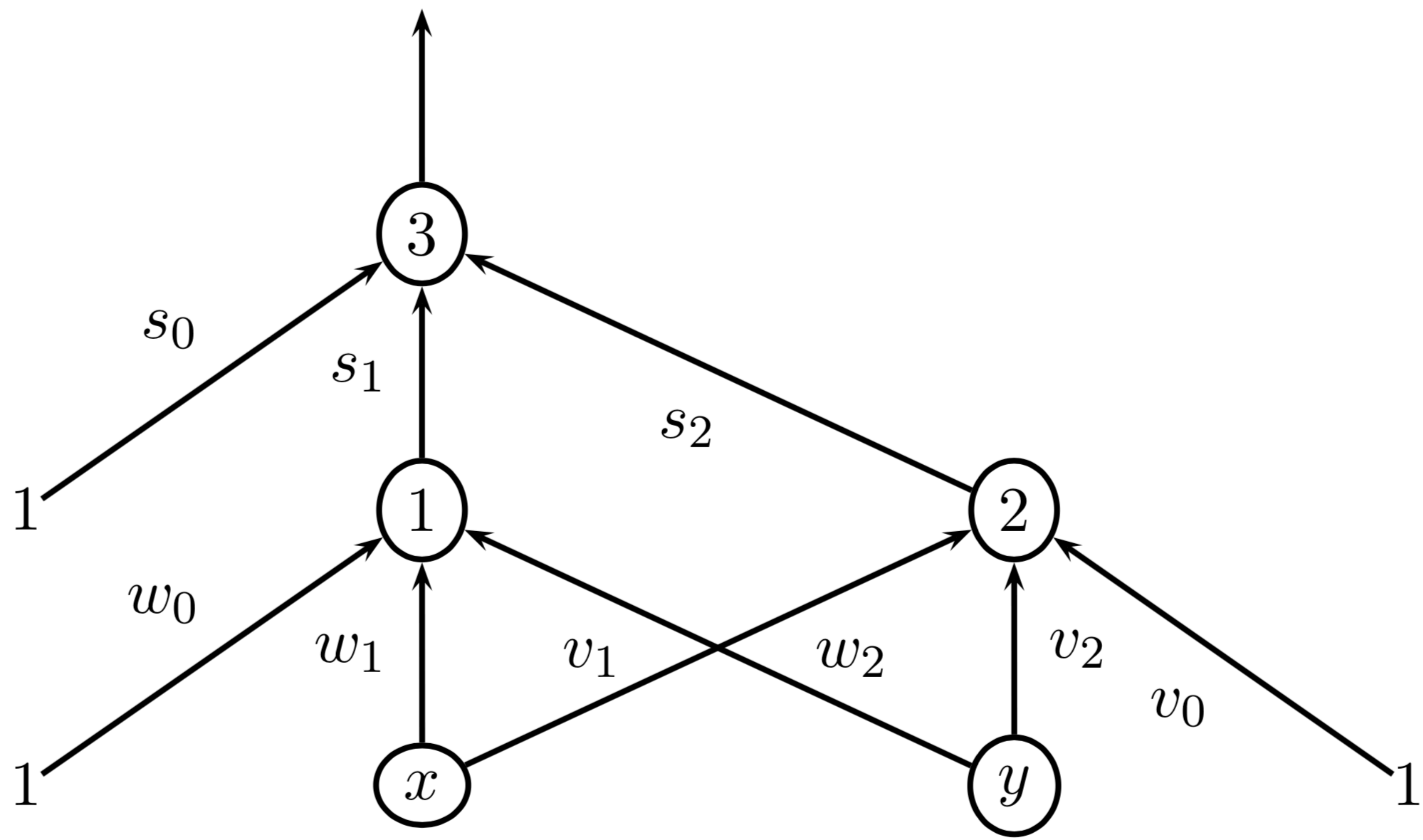
These examples are borrowed from Dr. Vibhav Gogate's Machine Learning class. (Fall 2014 Midterm and Spring 2012 Final)

Exercise 1

Draw a neural network that represents the function $f(x, y)$ defined below:

x	y	$f(x, y)$
0	0	10
0	1	-5
1	0	-5
1	1	10

Exercise 1 Solution



Exercise 1 Solution

Nodes labeled by 1 and 2 are simple threshold units while the node labeled by 3 is a linear unit.

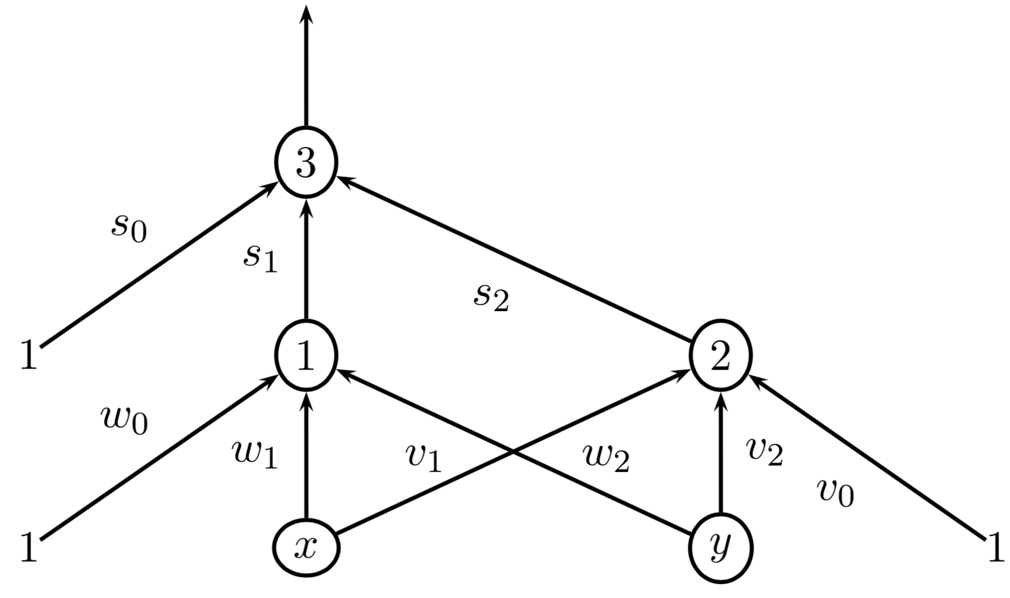
A possible setting of the weights is given below. Recall that the simple threshold unit is given by:

$$out = \begin{cases} +1 & \text{if } \sum_i w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

o_1, which is the output of node labeled by 1 implements the following function

$$o_1 = \begin{cases} +1 & \text{if } \neg x \wedge \neg y \text{ is true} \\ -1 & \text{otherwise} \end{cases}$$

To achieve this, we can use $w_0 = 1$ and $w_1 = w_2 = -2$



Exercise 1 Solution

o_2, which is the output of node labeled by 2 implements the following function

$$o_2 = \begin{cases} +1 & \text{if } x \wedge y \text{ is true} \\ -1 & \text{otherwise} \end{cases}$$

To achieve this, we can use $v_0 = -2$ and $v_1 = v_2 = 1.5$.

o_3 implements the following function

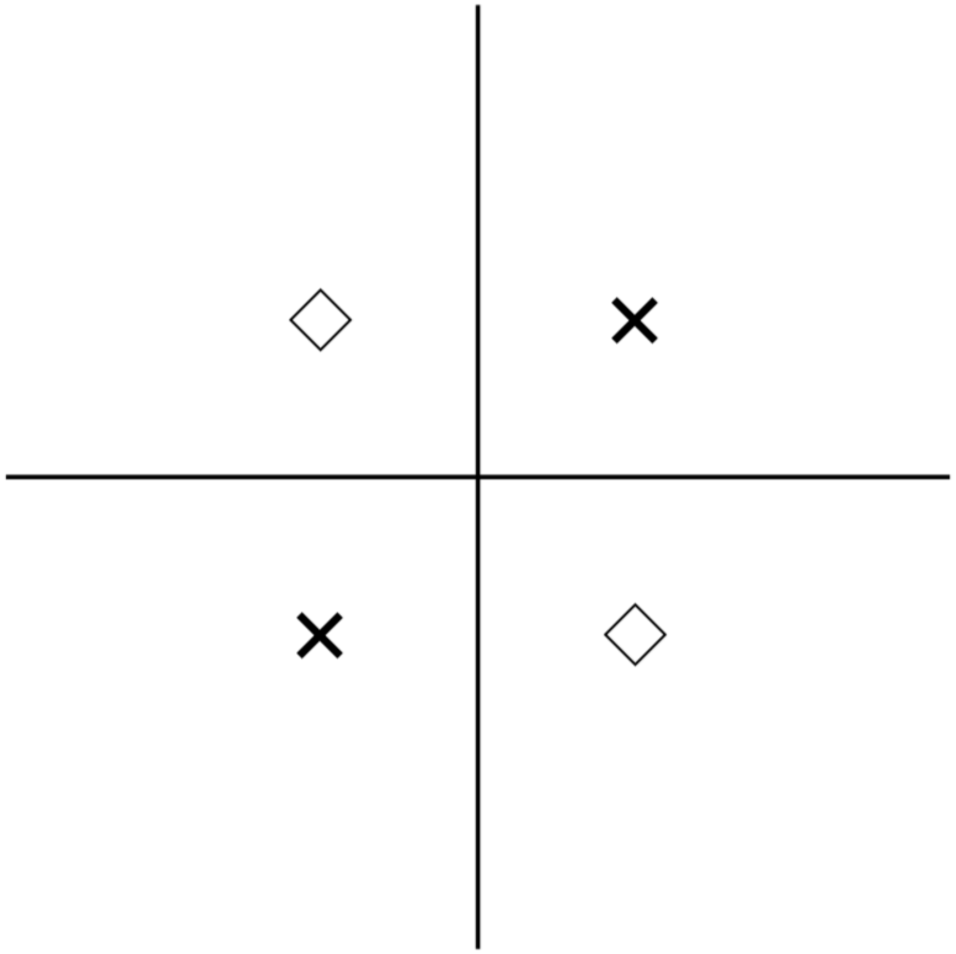
$$o_3 = \begin{cases} +10 & \text{if } o_1 = +1 \text{ or } o_2 = +1 \\ -5 & \text{otherwise} \end{cases}$$

Note that since 3 is a linear unit, we need it to obey the following constraints:
$s_0 + s_1 - s_2 = 10$ (if $o_1 = +1$ and $o_2 = -1$)
$s_0 - s_1 + s_2 = 10$ (if $o_1 = -1$ and $o_2 = +1$)
$s_0 - s_1 - s_2 = -5$ (if $o_1 = -1$ and $o_2 = -1$)

Notice that the case $o_1 = +1$ and $o_2 = +1$ can never happen.

A solution to the three equations is $s_0 = 10$ and $s_1 = s_2 = 7.5$.

Exercise 2

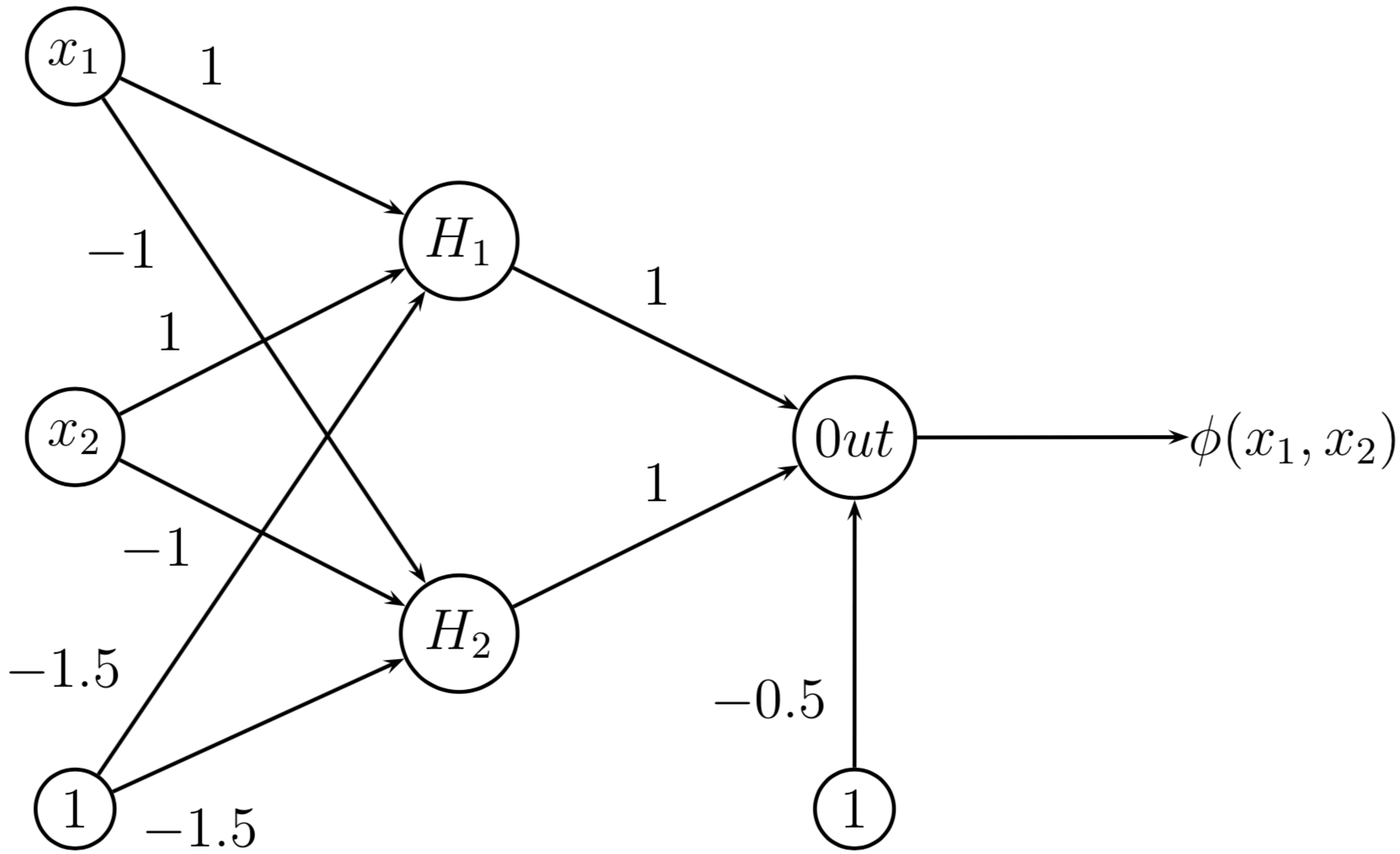


(5 points) Consider the data set given above. Assume that the co-ordinates of the points are (1,1), (1,-1), (-1,1) and (-1,-1). Draw a neural network that will have zero training error on this dataset. (Hint: you will need exactly one hidden layer and two hidden nodes).

Exercise 2 Solution

Solution: There are many possible solutions to this problem. I describe one way below. Notice that the dataset is not linearly separable. Therefore, we will need at least two hidden units. Intuitively, each hidden unit will represent a line that classifies one of the squares (or crosses) correctly but mis-classifies the other. The output unit will resolve the disagreement between the two hidden units. I am assuming that the symbol $\times$ is positive and the other symbol implies negative class.

Exercise 2 Solution



All hidden and output units are simple threshold units (aka sign units). Recall that each sign unit will output a $+1$ if $w_0x_0 + w_1x_1 + \ldots + w_nx_n > 0$ and -1 otherwise.

Back Propagation

1. Initialize all weights to some small random value
2. Until convergence do
    1. for each example in training data do
        1. input the example x and compute output
        2. for each unit k do

$$\delta_k \leftarrow \underbrace{o_k(1-o_k)}(t_k - o_k)$$

for sigmoid. change for other functions

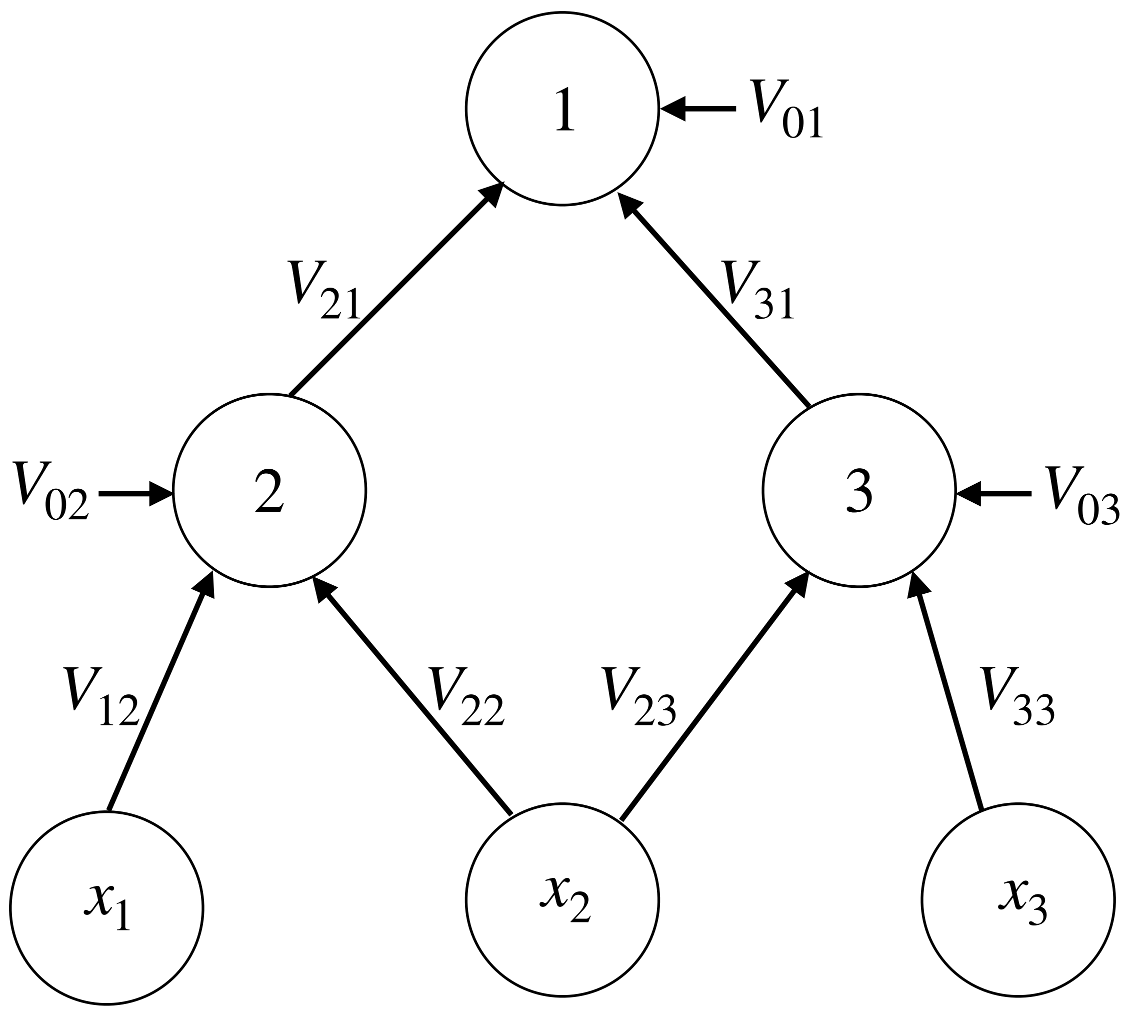
3. for each hidden unit h do

$$\delta_h \leftarrow \overbrace{o_h(1-o_h)} \sum_{u \in next\_layer} w_{h,u}\delta_u$$

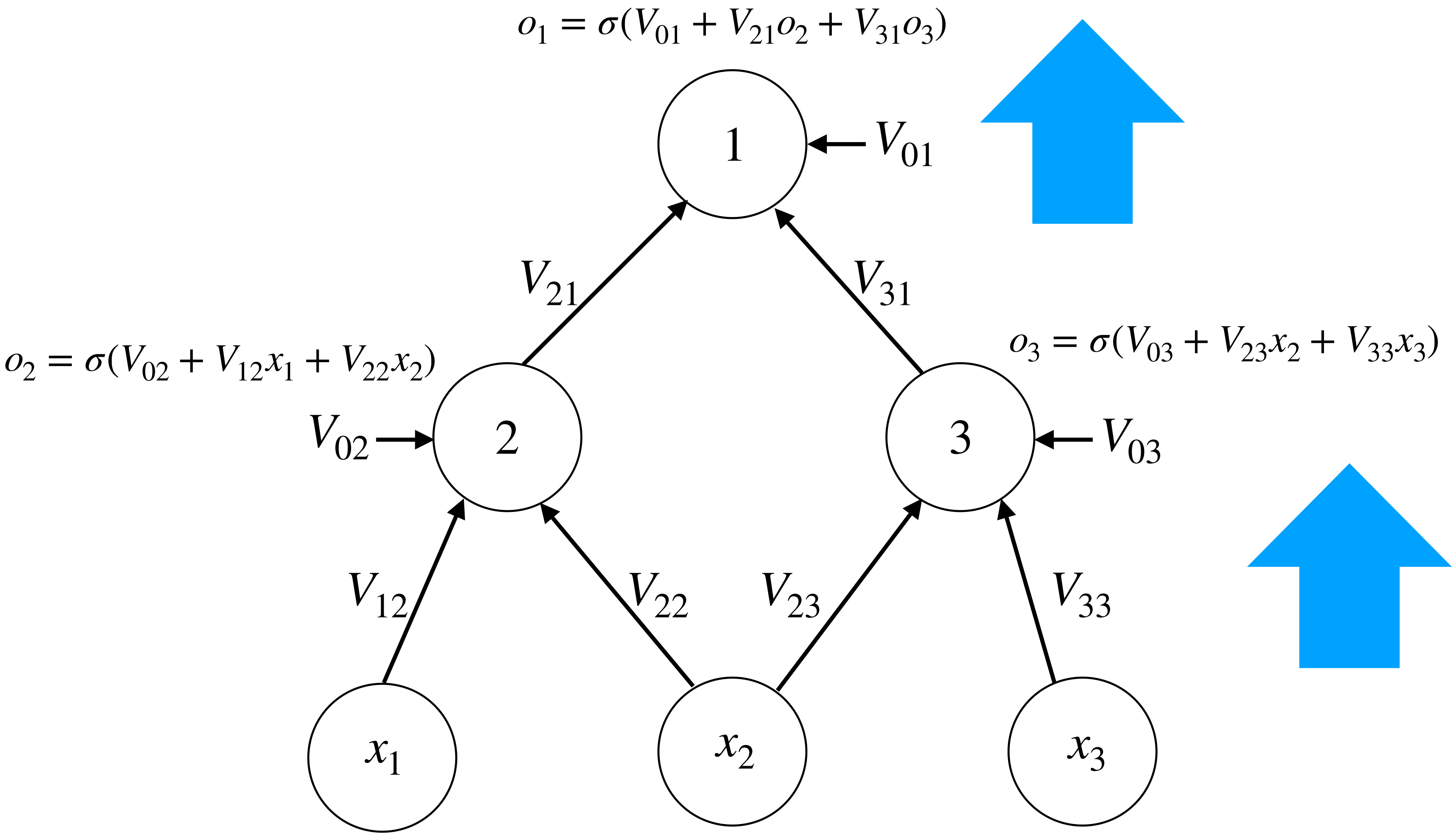
5. Update each network weights

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j} \qquad \Delta w_{i,j} = \eta \delta_j o_{i,j}$$

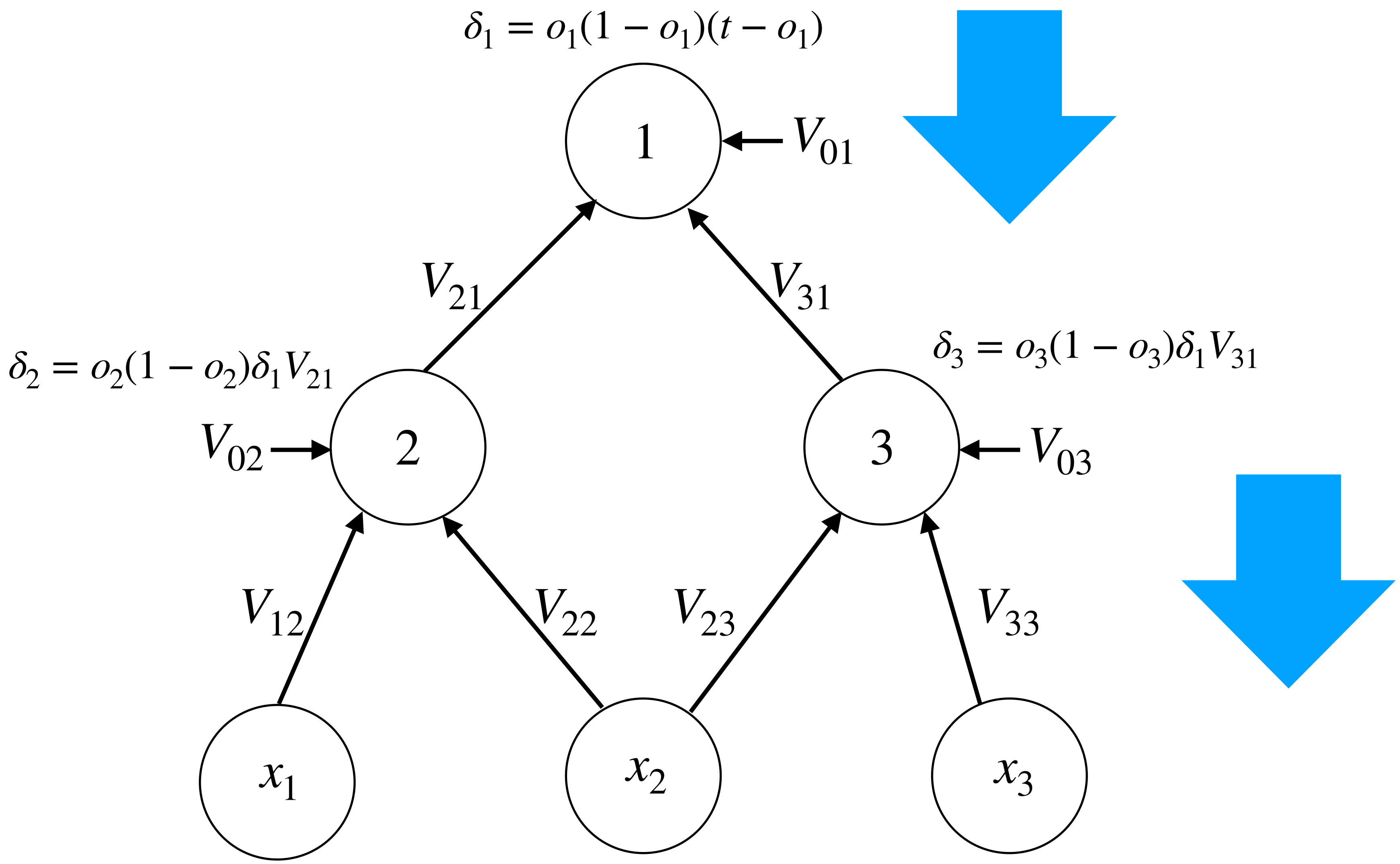
Back Propagation in Action



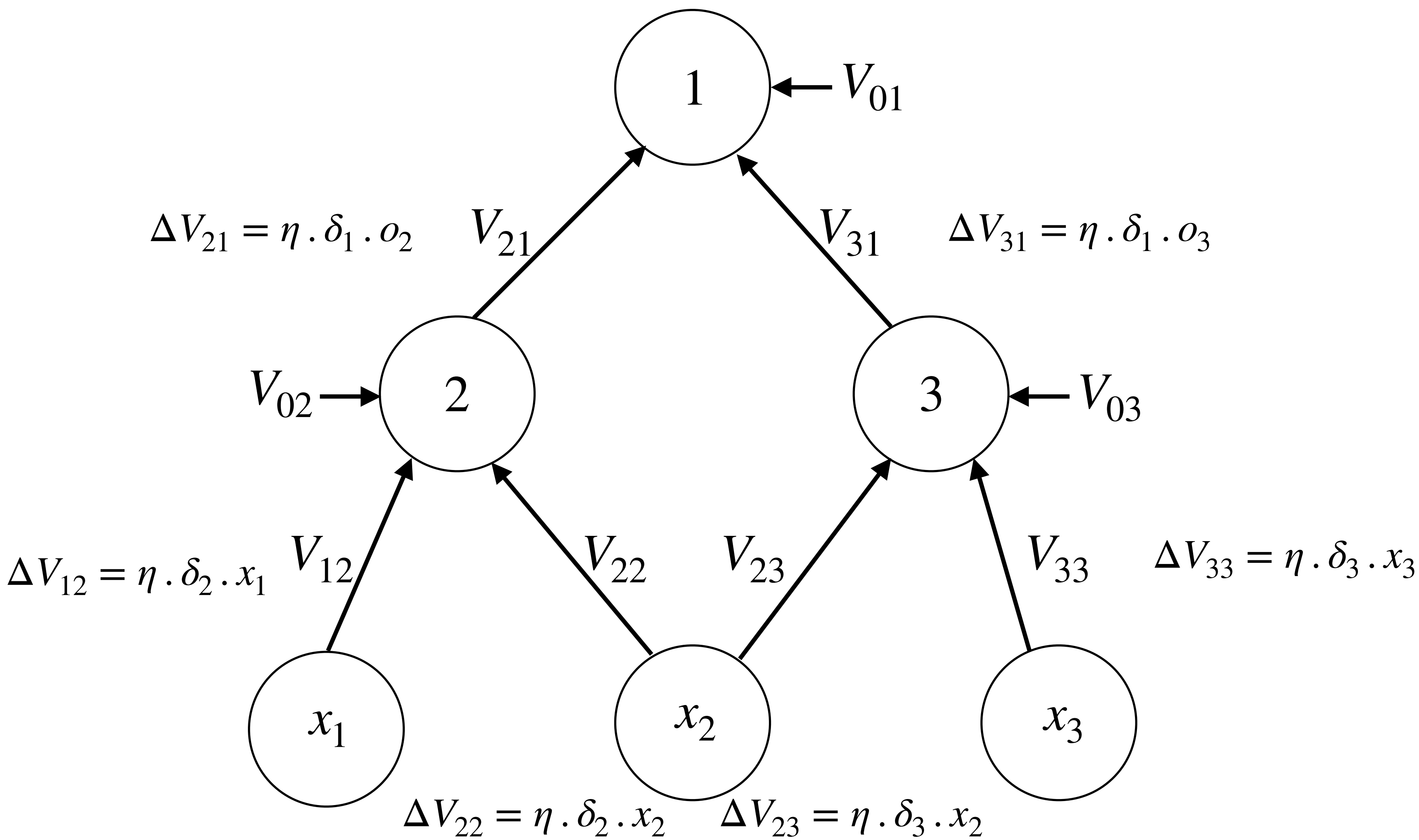
Back Propagation in Action



Back Propagation in Action

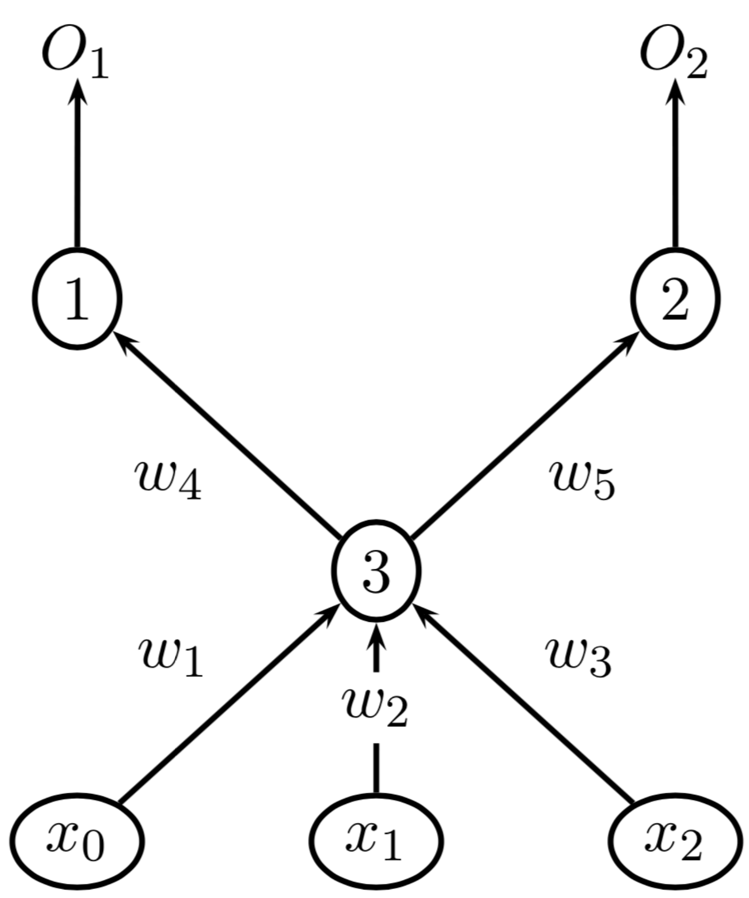


Back Propagation in Action



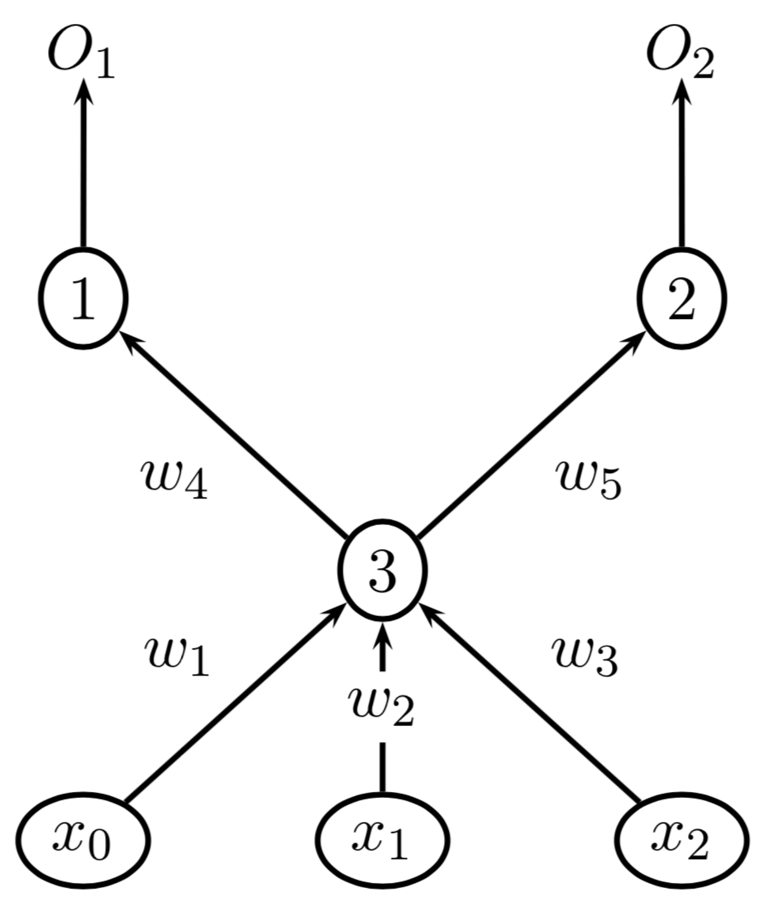
Exercise 3

Run the Back Propagation algorithm on the following neural network.



Exercise 3

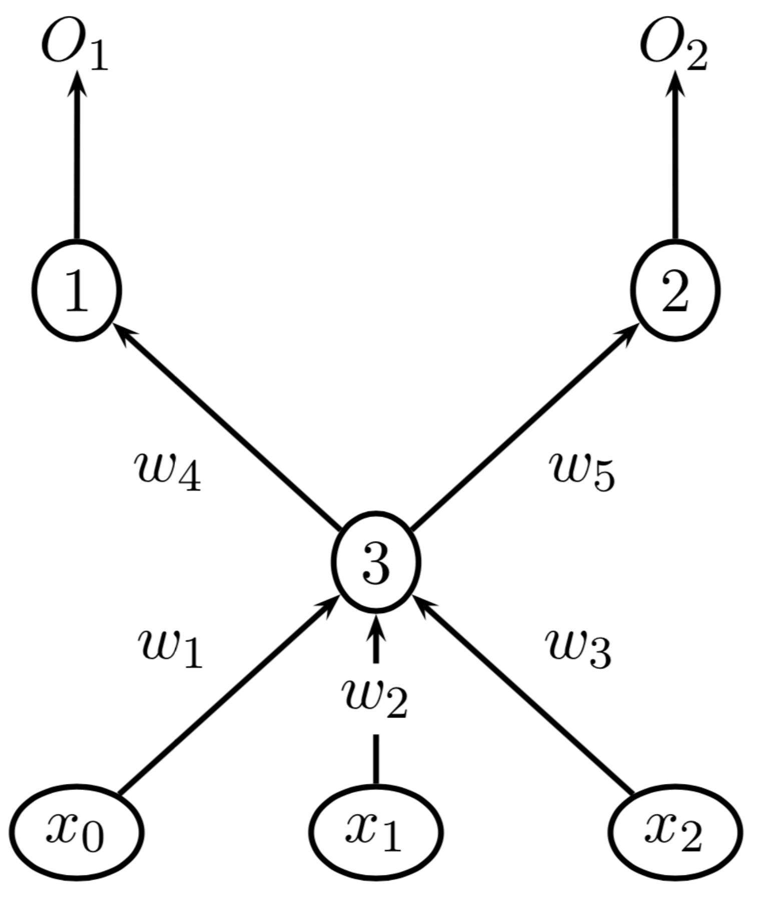
Assume that all internal nodes compute the sigmoid function. Write an explicit expression that shows how back propagation (applied to minimize the least squares error function) changes the values of w_1, w_2, w_3, w_4 and w_5 when the algorithm is given the example $x_1 = 0$, $x_2 = 1$, with the desired response $y_1 = 0$ and $y_2 = 1$ ($x_0 = 1$ is the bias term). Assume that the learning rate is α and that the current values of the weights are: $w_1 = 3$, $w_2 = 2$, $w_3 = 2$, $w_4 = 3$ a1nd $w_5 = 2$. Let O_1 and O_2 be the output of the output units 1 (which models y_1) and 2 (which models y_2) respectively . Let O_3 be the output of the hidden unit 3.



Exercise 3 Solution

(5 points) Forward propagation. Write equations for O_1, O_2 and O_3 in terms of the given weights and example.

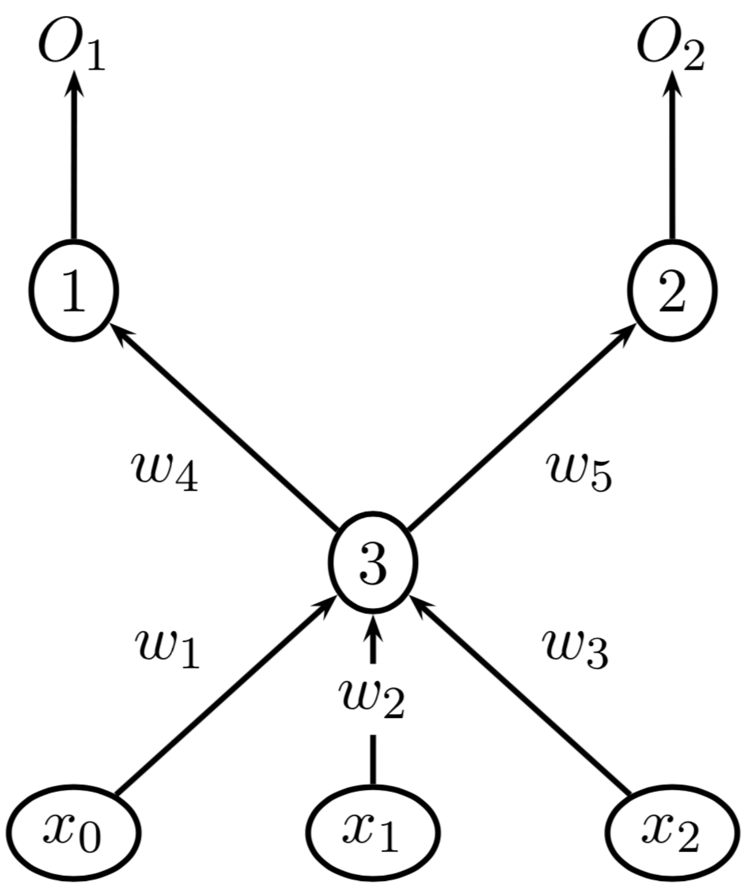
Solution: $O_3 = \sigma(w_1 x_0 + w_2 x_1 + w_3 x_2) = \sigma(3 * 1 + 2 * 0 + 2 * 1) = \sigma(5)$
$O_2 = \sigma(w_5 o_3) = \sigma(2\sigma(5))$
$O_1 = \sigma(w_4 o_3) = \sigma(3\sigma(5))$



Exercise 3 Solution

(5 points) Backward propagation. Write equations for δ_1, δ_2 and δ_3 in terms of the given weights and example where δ_1, δ_2 and δ_3 are the values propagated backwards by the units denoted by 1 and 2 and 3 respectively in the neural network.

Solution: $\delta_1 = (y_1 - o_1)o_1(1 - o_1) = o_1^3 - o_1^2$
$\delta_2 = (y_2 - o_2)o_2(1 - o_2) = o_2(1 - o_2)^2$
$\delta_3 = o_3(1 - o_3)(w_4\delta_1 + w_5\delta_2) = o_3(1 - o_3)(3\delta_1 + 2\delta_2)$



Exercise 3 Solution

(5 points) Give an explicit expression for the new (updated) weights w_1, w_2, w_3, w_4 and w_5 after backward propagation.

Solution: Let η denote the learning rate

$w_1 = w_1 + \eta\delta_3 x_0 = 3 + \eta\delta_3 \times 1 = 3 + \eta\delta_3$

$w_2 = w_2 + \eta\delta_3 x_1 = 2 + \eta\delta_3 \times 0 = 2$

$w_3 = w_3 + \eta\delta_3 x_2 = 2 + \eta\delta_3 \times 1 = 2 + \eta\delta_3$

$w_4 = w_4 + \eta\delta_1 o_3 = 3 + \eta\delta_1 o_3$

$w_5 = w_5 + \eta\delta_2 o_3 = 2 + \eta\delta_2 o_3$