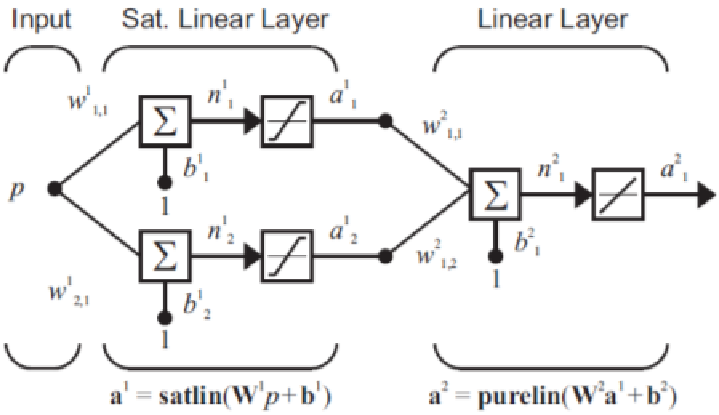
Νευρο-Ασαφής Υπολογιστική
Σειρά προβλημάτων: 1η

Λούκας Ελευθέριος Παναγιώτης 2029 - Νικητάκης Παναγιώτης 1717

Οκτώβριος 2018

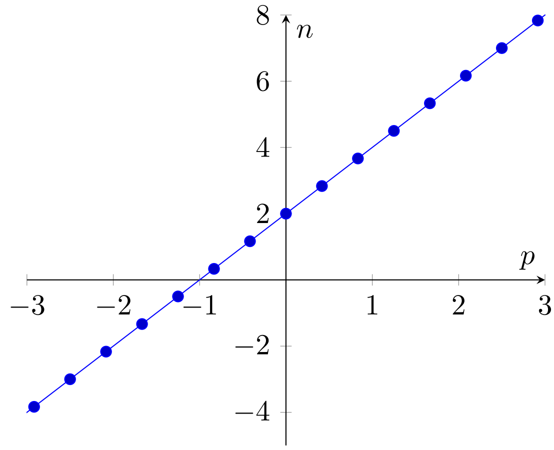
Άσκηση 1η



$$w^1_{1,1} = 2\,,\ w^1_{2,1} = 1\,,\ b^1_1 = 2\,,\ b^1_2 = -1\,,\ w^2_{1,1} = 1\,,\ w^2_{1,2} = -1\,,\ b^2_1 = 0$$

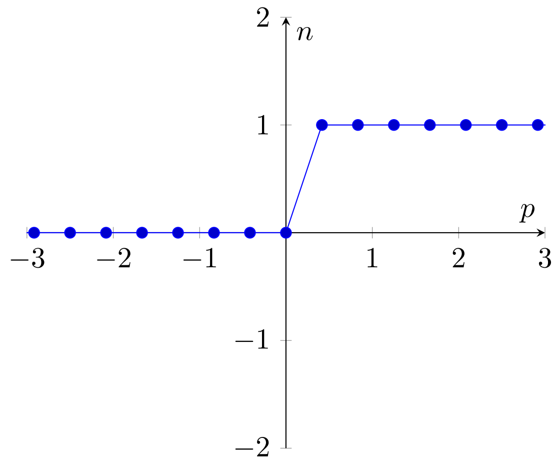
Το *activation* είναι συνάρτηση του n το οποίο είναι συνάρτηση του p και παίρνουμε τις περιπτώσεις -3 μεχρι 3.

1) $n_1^1 = p * w_{1,1}^1 + b_1^1 * 1 = 2 * p + 2$

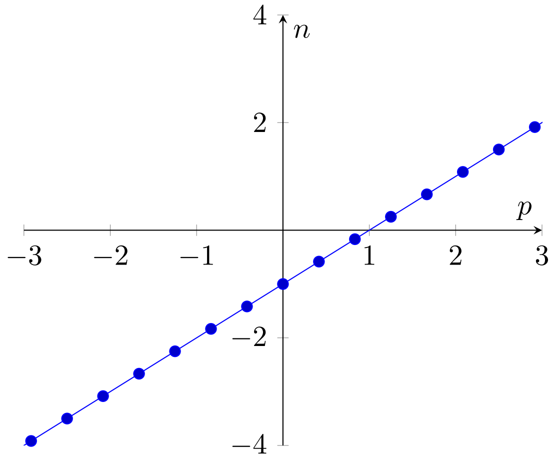


2) $a_1^1 = satlin(n_1^1)$

$$satlin(n) = \begin{cases} 0, & \text{Αν } n < 0. \\ n, & \text{Αν } n > 0 \text{ και } n < 1. \\ 1, & \text{Αν } n > 0. \end{cases} \tag{1}$$

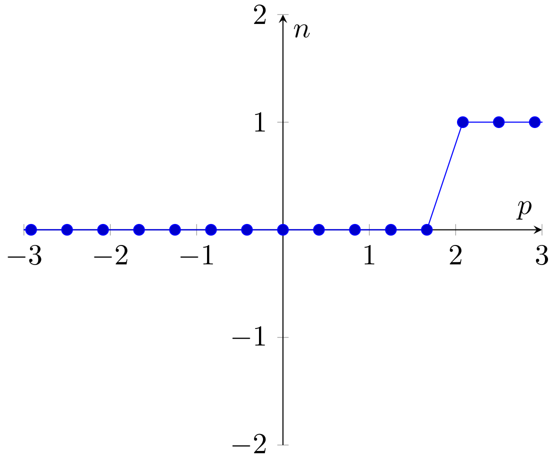


3) $n_2^1 = p * w_{2,1}^1 + 1 * b_2^1 = p - 1$



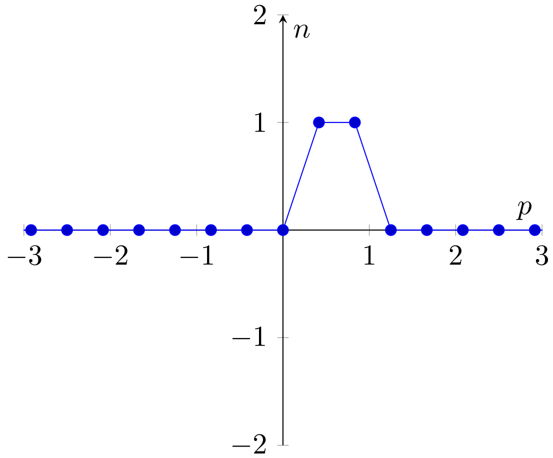
4) $a_2^1 = satlin(n_2^1)$

$$satlin(n) = \begin{cases} 0, & \text{Αν } n < 0. \\ n, & \text{Αν } n > 0 \text{ και } n < 1. \\ 1, & \text{Αν } n > 0. \end{cases} \tag{2}$$

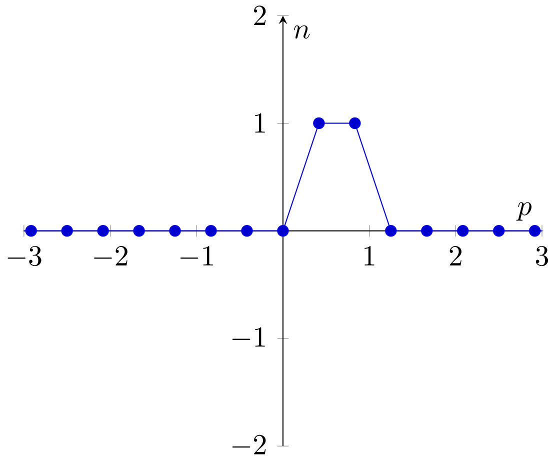


5) $n_1^2 = w_{1,1}^2 * a_1^1 + w_{1,2}^2 * a_2^1 + 1 * b_1^2 = a_1^1 - a_2^1$

$$satlin(n) = \begin{cases} 0, & \text{Αν } n < 0. \\ n, & \text{Αν } n > 0 \text{ και } n < 1. \\ 1, & \text{Αν } n > 0. \end{cases} \tag{3}$$

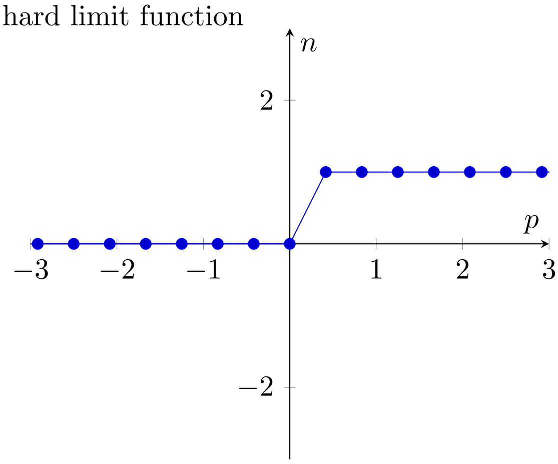


6) $a_1^2 = purelin(n_1^2)$ όπου $pureline(n) = n$

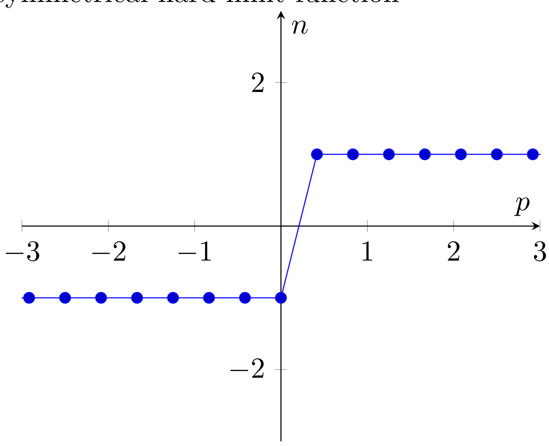


Άσκηση 2η

hard limit function



symmetrical hard limit function



Μία απλή έκφραση που κάνει map το [0,1] στο [-1,1] θα είναι της μορφής: $f(x) = a * x + c$

x	0	1
y	-1	1

Θέλουμε $f(0) = -1 => c = -1$ και $f(1) = a - 1 = 1 => a = 2$

Για το inverse mapping:

x	-1	1
y	0	1

Θέλουμε $f(-1) = 0 => -a + c = 0$ (1)
και $f(1) = 1 => a * 1 + c = 1 => a = 1 - c$ (2)
(1)(2) $=> -1 + c + c = 0 => 2 * c = 1 => c = 0.5$ (3)
(2)(3) $=> a = 1 - c = 1 - 0.5 => a = 0.5$
Άρα $f(x) = 0.5 * x + 0.5$

Για $n \geq 0$:

activation	target	error	function
1	1	0	hardlim
1	0	-1	hardlim
1	1	0	symmetric hardlim
1	0	-1	symmetric hardlim

Για $n = 0$:

activation	target	error	function
0	1	1	hardlim
0	0	0	hardlim
-1	0	1	symmetric hardlim
-1	-1	0	symmetric hardlim

Τα παραπάνω πινακάκια αφορούν πως θα κινηθούν οι τιμές του νεύρωνα για ίδιες εισόδους p, κάνοντας learning με το perceptron learning rule.
$w(k+1) = w(k) + e * p$ και $b(k+1) = b(k) + e$
Εφόσον βλέπουμε πως τα σφάλματα είναι διαφορετικά, τα βάρη και τα biases θα αποκτήσουν διαφορετική τιμή σύμφωνα με τον παραπάνω τρόπο.

Άσκηση 3η

Following, we have a simple Perceptron implementation in Python using NumPy, answering the first bullet of problem 3. The following link is the link to Colaboratory, Google's online Jupyter Notebook Service, where you can see, run, and even edit our code without any setup needed: https://colab.research.google.com/drive/1wp_HnFrVsJa4RFG7RQralMt8jOFuw3aN

```
import numpy
from random import choice

activate = lambda x: 0 if x < 0 else 1 # This will activate a neuron if its value
    is higher than 0
training_data = [ (np.array([1,4,1]), 0), (np.array([1,5,1]), 0),
    (np.array([2,4,1]), 0), (np.array([2,5,1]), 0), (np.array([3,1,1]), 1),
    (np.array([3,2,1]), 1), (np.array([4,1,1]), 1), (np.array([4,2,1]), 1)]
# The first two entries of the NumPy array are the two input values.
# The third entry of the array is the bias which is needed to move the threshold
    (also known as the decision boundary) up or down as needed by the activation
    function.
# The second element of the tuple is the expected result or the target value.

w = np.random.rand(3) # Initialize the weights randomly
learning_rate = 0.01
epochs = 100 # Arbitrary number, feel free to change
for i in range(epochs):
  x, expected = choice(training_data) # Get a random input set from the training
      data
```

```
  result = np.dot(w, x)
  error = expected - activate(result) # Can be -1, 0, 1

  w += learning_rate * error * x # Fix the weights through the unified perceptron
      learning rule

print("Final Weights: ", w)

i = 0
for x, _ in training_data:
  i = i + 1
  result = np.dot(x, w)
  print("p_%d"%(i),"--- " "{} --- Value before activation: {} -> Value after
      activation: {}".format(x[:2], result, activate(result)))
```

```
Final Weights:  [ 0.46909324 -0.35682067  0.32024546]
p_1 --- [1 4] --- Value before activation: -0.6379439895491833 -> Value after activat
p_2 --- [1 5] --- Value before activation: -0.994764663197043 -> Value after activati
p_3 --- [2 4] --- Value before activation: -0.16885074895096763 -> Value after activa
p_4 --- [2 5] --- Value before activation: -0.5256714225988273 -> Value after activat
p_5 --- [3 1] --- Value before activation: 1.370704512590827 -> Value after activatio
p_6 --- [3 2] --- Value before activation: 1.0138838389429674 -> Value after activati
p_7 --- [4 1] --- Value before activation: 1.8397977531890426 -> Value after activati
p_8 --- [4 2] --- Value before activation: 1.482977079541183 -> Value after activatio
```

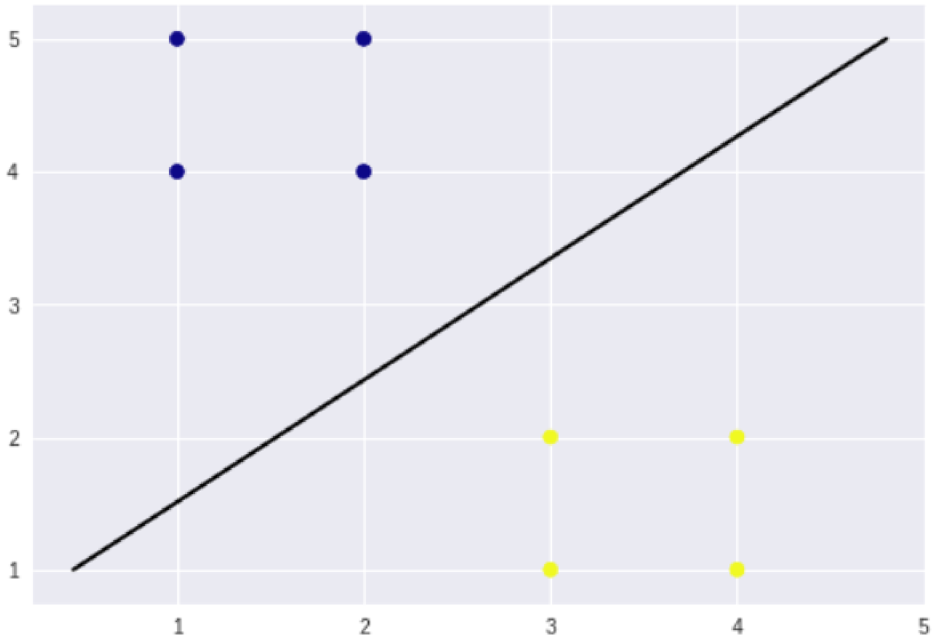
The program is also created on TensorFlow in order for us to have better intuitions and also create a robust solution for problem 3

```
# The program is also created on TensorFlow in order for us to have better
    intuitions and also create a robust solution for problem 3

# Libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Number of features & iterations, feel free to change
NUM_FEATURES = 2
NUM_ITERATIONS = 2000 # Feel free to change

# Initialize input p and targets for the variables
p = np.array([[1,4], [1,5], [2,4], [2,5], [3,1], [3,2], [4,1], [4,2]],
    np.float32) # 8x2 features
targets = np.array([0,0,0,0,1,1,1,1], np.float32).reshape([8,1]) # 8x1 target
    variables
```

```
# Create 2 placeholders for input p and target for the TensorFlow variables
P = tf.placeholder(tf.float32, shape=[8, NUM_FEATURES], name='P') # The
    NUM_FEATURES variable is added to ensure that the decision boundary of any
    solution will not intersect one of the original input vectors
T = tf.placeholder(tf.float32, shape=[8, 1], name='T') # 8x1, same as targets

# Create 2 TensorFlow vriables for the weight W and the bias B
W = tf.Variable(tf.random_normal([NUM_FEATURES, 1]), tf.float32, name='W') #
    Initialize weights randomly
B = tf.Variable(tf.zeros([1, 1]), tf.float32, name='B') # Initialize bias equal
    to zero

# Calculate the activation // activate(w.T*p + b)
predictions = tf.nn.sigmoid(tf.add(tf.matmul(P, W), B)) # tanh, ReLU, etc would
    work too

# Calculate the loss
loss = tf.nn.sigmoid_cross_entropy_with_logits(logits=predictions, labels=T) #
    'T' must have the same type and shape as logits.

# Calculate the training step
training_step = tf.train.AdamOptimizer(learning_rate).minimize(loss) # We could
    use Gradient Descent but we wanted to test how superior Adam is

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
with tf.Session() as sess:
  sess.run(init) # Run the session or else 'the computational graph'
  for epoch in range(NUM_ITERATIONS): # For some iterations
    sess.run(training_step, feed_dict={P: p, T: targets}) # Train the Perceptron
    #weights = W.eval() # Hold the final value of it
    #bias = B.eval()
    weights = sess.run(W)
    bias = sess.run(B)

# Prints
print("Final Weight 1", weights[0])
print("Final Weight 2", weights[1])
print("Bias:", bias)

# Decision Boundary
x1 = np.array([np.min(p[:, 1]), np.max(p[:, 1])])
x0 = np.squeeze(((-1/weights[0])*(weights[1]*x1+bias)))

# Scatterplot
plt.scatter(p[:, 0], p[:, 1], c=[0,0,0,0,1,1,1,1], cmap='plasma')

# Plot
```

```
plt.plot(x0, x1, color='k', linewidth=2)

# From the following plot, we can see that the accuracy is 100%
# since the decision boundary is in a good positiong, classifying the 2 labels
    correctly.

# Beware that some times it may need more epochs/generations for the perceptron
    to be correct.
```

```
Final Weight 1 [2.9660308]
Final Weight 2 [-3.2286415]
Bias: [[1.9091275]]
[<matplotlib.lines.Line2D at 0x7f2f1d0abac8>]
```



Άσκηση 4η

Υπάρχει ο Βοολεαν τύπος: $min(A, b) = 1 - max(1 - a, 1 - b)$ (1)

$t_L(A -> B) = min(1, 1 - t(A) + t(B)) = c$ (2)

(1)(2) $=> c = 1 - max(0, t(A) - t(B)) = minL$

1η περίπτωση:

Αν $max = 0 => c = 1$ τότε θα πρέπει να ισχύει $t(A) - t(B) \leq 0 => t(B) \geq t(A)$ όπου $t(A) \geq a \geq max[0, a]$

Άρα $t(B) \geq max[0, a] = max[0, a + 1 - 1]$ Αποδείχθηκε.

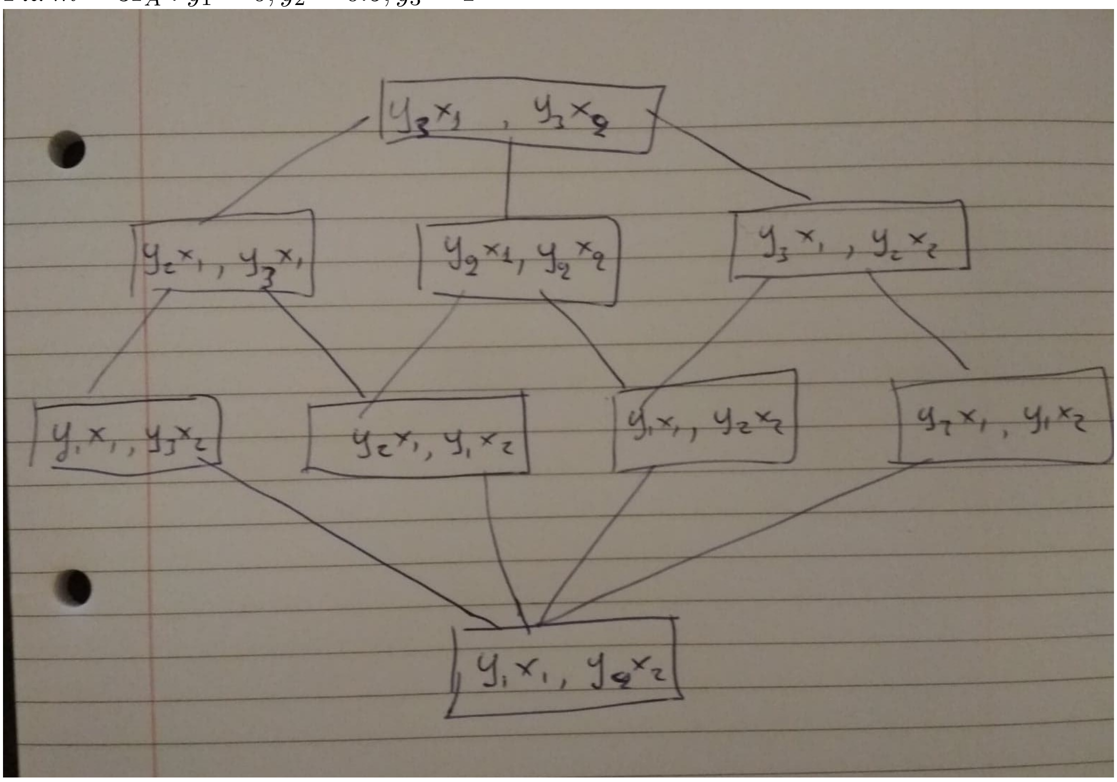
2η περίπτωση:

Αν $max = t(A) - t(B)$ τότε $c = 1 - t(A) + t(B)$ άρα $t(A) + t(B)$ (3)

Βέβαια $t(A) \geq a$ (4)
(3)(4) $=> 1 - c + t(B) \geq a <=> t(B) \geq a + c - 1$ από το οποίο μπορούμε να πούμε πως $t(B) \geq max[0, a + c - 1]$. Αποδείχτηκε.

Άσκηση 5η
Από άσκηση4 έχουμε $c = min(1, 1 - t(A) + t(B)$
Αν $min(...) = 1$ τότε σημαίνει πως $c = 1$ και $1 - t(A) + t(B) \geq 1 => -t(A) \geq t(b) => t(A) \leq t(B)$ όπου $t(B) \leq b$ από την εκφώνηση.
Άρα $t(A) \leq b \leq min(1, b)$ Αποδείχτηκε.
Αν $min(...) = 1 - t(A) + t(B)$ τότε σημαίνει πως $c = 1 - t(A) + t(B)$
άρα $t(B) = c - 1 + t(A)$ (1)
Βέβαια από εκφώνηση $t(B) \leq b$ άρα $c - 1 + t(A) \leq b => t(A) \leq b + 1 - c$ αφού $t : s- > [0, 1]$ από το οποίο μπορούμε να πούμε πως $t(A) \leq min(1, 1 - c + b)$

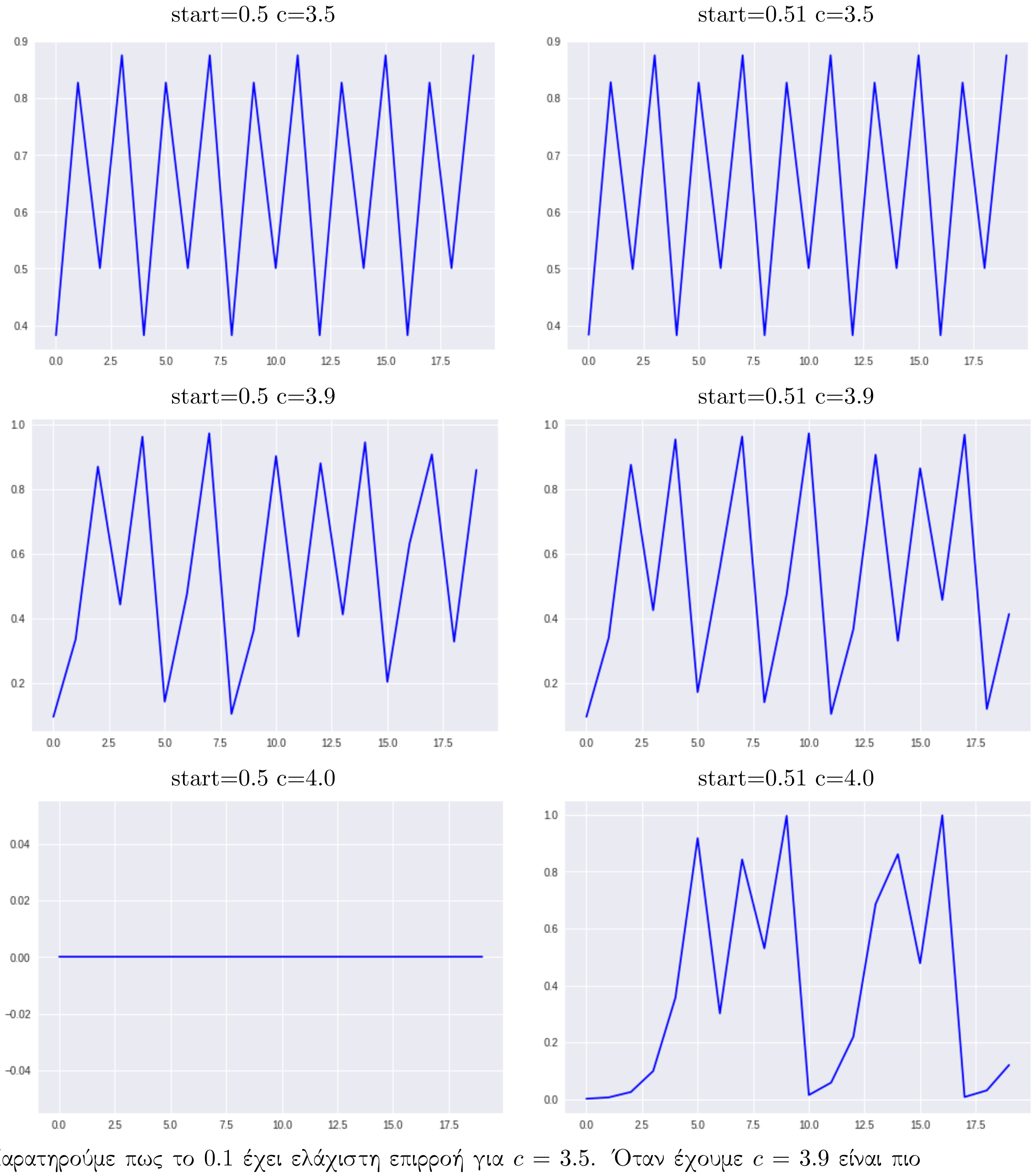
Άσκηση 6η
Το $x = x_1, ..., x_n$ είναι $nonfuzzy$ και έχουμε 2 τιμές στην I_A (0 ή 1) περιέχει $2^n nofuzzysubsets$.
Όταν η I_A έχει m πιθανές τιμές $y_1 = 0, y_2, ..., y_m = 1$ τότεο το x θα περιέχει $m^n fuzzysubsets$.
Για $m = 3 I_A : y_1 = 0, y_2 = 0.5, y_3 = 1$



Άσκηση 7η

Link to the code on Colaboratory: https://colab.research.google.com/drive/1wP0JmWh7K9nqMX1NOoy_9wzacPQG4_iE

```
# Libraries
import numpy as np
import matplotlib.pyplot as plt

# Init as objects
#X_array = []
#F_array = []

# Initialized values for c and x0
c = 3.9
start = 0.5

# Initialize X and F as lists
X = []
F = []

# Populate X
for k in range (20):
  if (k == 0):
    X.append( c * start * (1-start))
  else:
    X.append(c*X[k-1] * (1-X[k-1]))

# Populate F
for k in range (20):
    F.append(c*X[k]*(1-X[k]))

#print(len(X))
#print(X)
#print(len(F))
#print(F)

t = [i for i in range(0, 20)]
# Emeis theloume na kanoume to plot gia (t, F)
#plt.axis([0, 1, 0, 1])
plt.plot(t, F, 'b-')

plt.show()
```



Παρατηρούμε πως το 0.1 έχει ελάχιστη επιρροή για $c = 3.5$. Όταν έχουμε $c = 3.9$ είναι πιο εμφανής η διαφορά. Ενώ όταν $c = 4.0$ έχουμε χάος. Αυτό μπορεί να μας δείξει το πόσο μεγάλη διαφορά μπορεί να έχουν κάποιες μικρές αριθμητικές αλλαγές στο λεαρνινγ ρατε όταν ψάχνουμε μια λύση σε ένα οπτιμιζατιον προβλεμ. Ένα από αυτά είναι η διόρθωση των βαρών στα νευρωνικά δίκτυα.

Άσκηση 8η

8

Problem 08

No. Date 5

Αν $S = \frac{1}{1+e^{-cx}}$, τότε ν.δ.ο. $S' = cS(1-S)$

1)

$$S' = cS(1-S) = c\left(\frac{1}{1+e^{-cx}}\right)\left(1-\frac{1}{1+e^{-cx}}\right)$$

$$= c\cdot\left(\frac{1}{1+e^{-cx}}\right)\left(\frac{1+e^{-cx}-1}{1+e^{-cx}}\right)$$

$$= \boxed{c\left(\frac{1}{1+e^{-cx}}\right)\cdot\left(\frac{e^{-cx}}{1+e^{-cx}}\right) = S'} \ (1)$$

$$\frac{dS}{dx} = \left(\left(1+e^{-cx}\right)^{-1}\right)' = -\frac{1}{(1+e^{-cx})^2}\left(-ce^{-cx}\right) = c\cdot\frac{1}{(1+e^{-cx})^2}\cdot e^{-cx}$$

$$= c\cdot\left(\frac{1}{1+e^{-cx}}\right)\left(\frac{e^{-cx}}{1+e^{-cx}}\right) = \frac{dS}{dx} \ (2)$$

(1) = (2) άρα αποδείχτηκε

http://www.tanenghong.com

No. Date 6

2) $S = e^{-cx^2}$, $S' = ?$

Έστω $S = e^{u}$ με $u = -cx^2$

$$S' = e^{u'} = (u)' \cdot e^{u} = (-cx^2)' \cdot e^{-cx^2} = -2cx \cdot e^{-cx^2}$$

Άρα $\boxed{S' = -2cxe^{-cx^2}}$

Αποδείχτηκε

3) Αν $S = \tanh(cx) = \frac{e^{x}-e^{-x}}{e^{x}+e^{x}}$, τότε ν.δ.ο $S' = c(1-S^2)$

$$\frac{dS}{dx}\quad S' = \frac{(1-e^{-2cx})'(1+e^{-2cx}) - (1-e^{-2cx})(1+e^{-2cx})'}{(1+e^{-2cx})^2}$$

$$= \frac{(2ce^{-2cx})(1+e^{-2cx}) - (1-e^{-2cx})(-2ce^{-2cx})}{(1+e^{-2cx})^2}$$

$$= \frac{2ce^{-2cx} + \cancel{2c(e^{-2cx})^2} + 2ce^{-2cx} - \cancel{2c(e^{-2cx})^2}}{(1+e^{-2cx})^2} = \boxed{\frac{4ce^{-2cx}}{(1+e^{-2cx})^2}}\ (1)$$

$$c(1-S^2) = c\left(1 - \frac{(1-e^{-2cx})^2}{(1+e^{-2cx})^2}\right) = c\left(\frac{(1+e^{-2cx})^2 - (1-e^{-2cx})^2}{(1+e^{-2cx})^2}\right) =$$

$$c\left(\frac{1+2e^{-2cx}+(e^{-2cx})^2 - 1 + 2e^{-2cx} - (e^{-2cx})^2}{(1+e^{-2cx})^2}\right) = c\left(\frac{4e^{-2cx}}{(1+e^{-2cx})^2}\right)$$

$$= \frac{c \cdot 4e^{-2cx}}{(1+e^{-2cx})^2}\ (2)$$

(1) = (2) οπότε αποδείχτηκε

No. Date 7

4) $S = e^{cx}$, ντε νδο $\frac{d^n S}{dx^n} = c^n e^{cx}$

$\frac{d^1 S}{dx^1} = e^{cx} \cdot (cx)' = c^1 e^{cx}$

$\frac{d^2 S}{dx^2} = (c^1 e^{cx})' = e^{cx} \cdot (cx)' \cdot c^1 = e^{cx} \cdot c \cdot c = c^2 e^{cx}$

αναλογικα

$\vdots$

$\frac{d^n S}{dx^n} = (c^{n-1} e^{cx})' = \ldots = c^n e^{cx}$ ✓

5) $S = 1 - e^{-cx}$, τότε $S'' = -c^2 e^{-cx}$

$S' = (1 - e^{-cx})' = -(-cx)' e^{-cx} = +c e^{-cx}$

$S'' = (c e^{-cx})' = c e^{-cx} \cdot (-cx)' = -c \cdot c \cdot e^{-cx} = -c^2 e^{-cx}$

Αποδείχτηκε

6) Αν $S = \frac{x^n}{c + x^n}$, τότε $S' = \frac{c \cdot n x^{n-1}}{(c + x^n)^2}$

$\frac{dS}{dx} = \frac{(x^n)'(c + x^n) - x^n (c + x^n)'}{(c + x^n)^2} = \frac{(n \cdot x^{n-1})(c + x^n) - x^n (n \cdot x^{n-1})}{(c + x^n)^2}$

$= \frac{(n \cdot x^{n-1})(\cancel{c + x^n - x^n})}{(c + x^n)^2} = \frac{n \cdot x^{n-1}}{(c + x^n)^2} \cdot c = S'$

Αποδείχτηκε

http://www.tanenghong.com

Άσκηση 9η

1)

$s(x) * (1 + e^{-cx}) = 1 => s(x) + s(x) * e^{-cx} = 1 => s(x) * e^{-cx} = 1 - s(x) => lne^{-cx} = \frac{1-s(x)}{s(x)} => lne^{-cx} = ln(\frac{1-s(x)}{s(x)}) => -cx = ln(\frac{1-s(x)}{s(x)}) => x = \frac{ln(1-s(x))-lns(x)}{-c}$

2)

$x^{'} = \frac{ln(1-s(x))^{'}}{-c} - \frac{lns(x)^{'}}{-c} => x^{'} = \frac{1}{c} * \frac{s(x)}{1-s(x)} + \frac{1}{c} * \frac{s^{'}(x)}{s(x)} => x^{'} = \frac{1}{c} * (\frac{s(x)}{1-s(x)} + \frac{s^{'}(x)}{s(x)})$

Η σιγμοειδής συνάρτηση είναι ≥ 0. Η $s^{'} = c * s(1-s)$ από την ασκ8 όπου και αυτή ≥ 0. Οπότε η $x^{'}$ είναι πάντα θετική, συνεπώς η x γνησίως αύξουσα. Άρα το x συνεχώς αυξάνεται όπως και το s.

3)

Η συνάρτη έχει αντίστροφο την s και η $s^{'} > 0$. Άρα όσο μεγαλώνει το s τόσο θα μεγαλώνει και το x.

Το παραπάνω επιβεβαιώνεται και με τη γραφική παράσταση της συνάρτησης.

Άσκηση 10η

10

Problem 10:

No. Date

Η απόδειξη του Perceptron convergence ουσιαστικά ασχολείται με ένα lower και ένα upper bound.

- Έστω το lower bound L.
  Έχει ήδη αποδειχθεί πως
  το $x^{*T}x(k) = x^{*T}z'(0) + x^{*T}z'(1) + x^{*T}z'(k-1)$

  ή, αλλιώς $x^{*T}z'(i) \geq \delta > 0$
  Αφού $\delta > 0$, τότε $\boxed{x^{*}z'(i) \geq 0 = \text{threshold}}$

  οπότε το zero threshold ΔΕΝ είναι εμπόδιο στο πρώτο μισό της απόδειξης

- Όσον αφορά το upper bound U, ο Novikoff στο paper "On convergence proofs for Perceptrons" για το Stanford το 1962 λέει πως μπορούμε να πάρουμε $\theta = 0 = T = \text{threshold}$ αλλά αυτό θα μας φέρει πολύ μεγάλους αριθμούς.
  Βέβαια, είναι κάτι που ισχύει.

  Η "κανονική" απόδειξη καταλήγει πως το perceptron θα κάνει το μέγιστο $\frac{R^2}{T^2}$ λάθη.

  $\|x_t\| \leq R$
  Βέβαια, για $T = 0$.
  Ακόμη σαν threshold T ορίζουμε το $T = \min y_i \langle w^*, x_i \rangle$ $i = 1, \ldots$

  Αφού δεν μπορούμε να διαιρέσουμε με το 0 και αυτό το 0 ορίζει μια απόσταση, η διαίσθησή μας θα ήταν να ασχοληθούμε με κάποια 'ασαφή λογική' της απόστασης για να επαληθεύσουμε το παραπάνω.