Neural networks
- Assignment 2 -

Markus Wiktorin

October 16, 2017

Learning processes

Learning paradigms

- Learning with a teache
  - Supervised learning
  - Teacher has knowledge of environme
  - Environment unknown to neural networ
  - Neural network tries to emulate the teacl
  - Difference between response and desired should be decrea
- Learning without a teache
  - Reinforcement learnin
    - Learning formed through continue interaction with environment
    - Can learn under delayed reinforceme
    - Goal: Minimize cost-to-go functio
    - Closely related to dynamic programmi
  - Unsupervised learning
    - No teacher or criti
    - Task independent measur
    - Use competitive learning ru

Learning tasks

- Pattern associatior
  - Autoassociation: store a set of patterns by present them
  - Heteroassociation: arbitrary set of input patterns paired with arbitrary set of output patterns
  - Storage phase: learnin
  - Recall phase: retrieval of memorized patte
- Pattern recognitior
  - A received pattern is assigned to one prescribed cl
  - Training: present data with catego
  - Later a new pattern has to be classifi
- Function approximatior
  - A non-linear unknown function needs to be learn
  - Samples of the function are give
  - Supervised learning
- Control
  - A plant should be controlle
  - Brain is perfect example because it controls many actua
  - Controller has to invert plant's input-output mappi
  - Indirect or direct learnin
- Filtering
  - Extract information about a prescribed quantity of interest from noisy
  - Signal may come from different sourc
  - Example cocktail-party-problem: listen to one person while many are spea
  - Beamforming: distinguish a target signal from background n
- Smoothing
  - Information does not have to be available immediat
  - Later data can have influence on previous da
- Prediction
  - Derive information for forecasti

Basic learning rules

- error-correctior
  - e(n) = d(n) - y(n
  - output should become closer to the desire
  - step-by-step adjustment until stabl
  - delta-rule / Wodrow-Hoff rul
  - error signal directly measurat
- memory-basec
  - experiences are store
  - desired response is scala
  - Needs criterion to define the local neighborhood of a test ve
  - Applies learning rule to training samples in local neighborhc
  - Euclidean distance can be used for local neighborho
  - k-nearest neighbor is alternativ
- Hebbian
  - If two neurons are activated simultaneously their synapse should be incre
  - If two neurons are activated asynchronously their synapse should be decre
  - These synapses are called Hebbian synap
  - Anti-Hebbian: increase and decrease the other way aro
  - Non-Hebbian, no increase or decrease dependent on acti
- competitive
  - Output neurons compete
  - Only single output neuron is active at a tin
  - Neurons which are same except for randomly distributed wei
  - Limit on strenght of neuron
  - Winner-takes-all neuron is active for a set of inp
  - Neurons become feature detectors for classes of input patte
- Boltzmann
  - Stochastic learning algorithm derived from statistical mechai
  - Neural network design on basis of Boltzmann learning is called Boltzmann mac
  - Neurons are recurrent and binary (either on or
  - Machine is characterized by energy functi
  - No self-feedbacl
  - Flip random neurons with a certain probabil
  - There are visible and hidden neuro
- Credit-assignment problen
  - Assign credit or blame for outcomes to each internal deci
  - Assign to actions: temporal credit-assignment proble
  - Assign to internal decisions: structural credit-assignment probl

In [2]:

```python
import numpy as np
import sympy as sp
sp.init_printing()
```

1.13

a)

In [3]:

```python
a, b = sp.symbols("a, b")

def phi(x):
    return 1 / (1 + sp.exp(x))
```

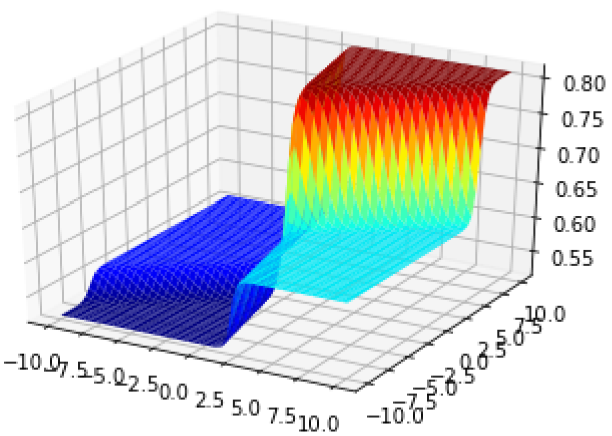
In [14]:

```python
a1 = a * 5 + b
b1 = -3 * b + 2 * a
a2 = phi(a1)
b2 = phi(b1)
a3 = 3 * a2 - b2
b3 = 6 * b2 + 4 * a2
a4 = phi(a3)
b4 = phi(b3)

def net(_a, _b):
    return phi(-2 * a4 + b4).subs(a, _a).subs(b, _b).evalf()
```

In [46]:

```python
sp.plotting.plot3d(phi(-2 * a4 + b4))
```



Out[46]:

<sympy.plotting.plot.Plot at 0x7f239acc5cc0>

In [28]:

```
values = np.zeros((400,3))
idx = 0
for x in range(-10, 10, 1):
    for y in range(-10, 10, 1):
        values[idx] = np.array([x, y, net(x,y)])
        idx = idx + 1
```

Out[28]:

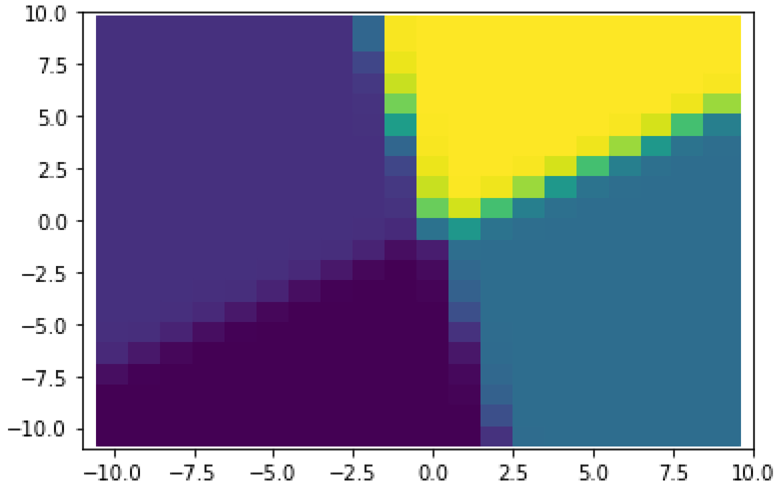
```
array([[-10.        , -10.        ,   0.51920915],
       [-10.        ,  -9.        ,   0.51925152],
       [-10.        ,  -8.        ,   0.52006775],
       ...,
       [  9.        ,   7.        ,   0.80845175],
       [  9.        ,   8.        ,   0.81132379],
       [  9.        ,   9.        ,   0.8114706 ]])
```

In [45]:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(values[:,0], values[:,1], c=values[:,2], marker=",", s=300)
```

Out[45]:

<matplotlib.collections.PathCollection at 0x7f239a980be0>



For two given inputs a and b the input-output mapping is given in the previous cell.

b)

If the last neuron is linear instead of the function φ then the input-output mapping is calculated in the next call, where w is the weight of the last neuron.

In [57]:

```
def phi_linear(x):
    return 5 * x
```

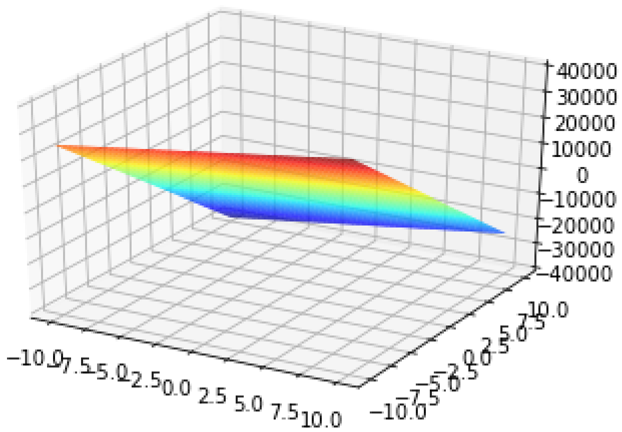
In [58]:

```
a1_l = a * 5 + b
b1_l = -3 * b + 2 * a
a2_l = phi_linear(a1_l)
b2_l = phi_linear(b1_l)
a3_l = 3 * a2_l - b2_l
b3_l = 6 * b2_l + 4 * a2_l
a4_l = phi_linear(a3_l)
b4_l = phi_linear(b3_l)

def net_linear(_a, _b):
    return phi_linear(-2 * a4_l + b4_l).subs(a, _a).subs(b, _b).evalf()
```

In [59]:

```
sp.plotting.plot3d(phi_linear(-2 * a4_l + b4_l))
```



Out[59]:

<sympy.plotting.plot.Plot at 0x7f239a23d7f0>

In [60]:

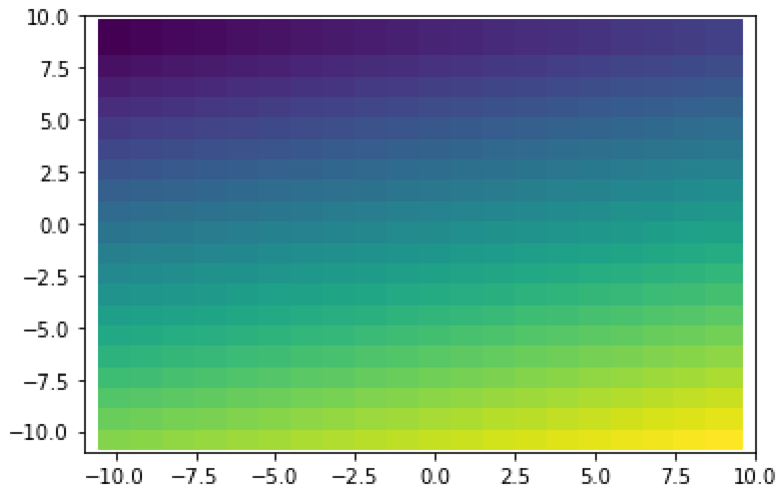
```
values_l = np.zeros((400,3))
idx = 0
for x in range(-10, 10, 1):
    for y in range(-10, 10, 1):
        values_l[idx] = np.array([x, y, net_linear(x,y)])
        idx = idx + 1
```

In [61]:

```
plt.scatter(values_l[:,0], values_l[:,1], c=values_l[:,2], marker=",", s=300)
```

Out[61]:

```
<matplotlib.collections.PathCollection at 0x7f239a3d7748>
```



New Classification Example

New data:

$C_1 = (1, 2), (1, 0), (0, 0)$

$C_2 = (-1, 0), (-1, 2), (0, 2)$

Apply trick to C_2:

$C_1 = (1, 1, 2), (1, 1, 0), (1, 0, 0)$

$C_2 = (-1, 1, 0), (-1, 1, -2), (-1, 0, -2)$

Adjusted pattern	Weight applied	w(n)x(n)	Update?	New weight
(1, 1, 2)	(1, 0, 0)	1	No	(1, 0, 0)
(1, 1, 0)	(1, 0, 0)	1	No	(1, 0, 0)
(1, 0, 0)	(1, 0, 0)	1	No	(1, 0, 0)
(-1, 1, 0)	(1, 0, 0)	-1	Yes	(0, 1, 0)
(-1, 1, -2)	(0, 1, 0)	1	No	(0, 1, 0)
(-1, 0, -2)	(0, 1, 0)	0	Yes	(-1, 1, -2)

end epoch 1

Adjusted pattern	Weight applied	w(n)x(n)	Update?	New weight
(1, 1, 2)	(-1, 1, -2)	-4	Yes	(0, 2, 0)
(1, 1, 0)	(0, 2, 0)	2	No	(0, 2, 0)
(1, 0, 0)	(0, 2, 0)	0	Yes	(1, 2, 0)

Adjusted pattern	Weight applied	w(n)x(n)	Update?	New weight
(-1, 1, 0)	(1, 2, 0)	1	No	(1, 2, 0)
(-1, 1, -2)	(1, 2, 0)	1	No	(1, 2, 0)
(-1, 0, -2)	(1, 2, 0)	-1	Yes	(0, 2, -2)

end epoch 2

Adjusted pattern	Weight applied	w(n)x(n)	Update?	New weight
(1, 1, 2)	(0, 2, -2)	-2	Yes	(1, 3, 0)
(1, 1, 0)	(1, 3, 0)	4	No	(1, 3, 0)
(1, 0, 0)	(1, 3, 0)	1	No	(1, 3, 0)
(-1, 1, 0)	(1, 3, 0)	2	No	(1, 3, 0)
(-1, 1, -2)	(1, 3, 0)	2	No	(1, 3, 0)
(-1, 0, -2)	(1, 3, 0)	-1	Yes	(0, 3, -2)

end epoch 3

Adjusted pattern	Weight applied	w(n)x(n)	Update?	New weight
(1, 1, 2)	(0, 3, -2)	-1	Yes	(1, 4, 0)
(1, 1, 0)	(1, 4, 0)	5	No	(1, 4, 0)
(1, 0, 0)	(1, 4, 0)	1	No	(1, 4, 0)
(-1, 1, 0)	(1, 4, 0)	3	No	(1, 4, 0)
(-1, 1, -2)	(1, 4, 0)	3	No	(1, 4, 0)
(-1, 0, -2)	(1, 4, 0)	-1	Yes	(0, 4, -2)

end epoch 4

Adjusted pattern	Weight applied	w(n)x(n)	Update?	New weight
(1, 1, 2)	(0, 4, -2)	0	Yes	(1, 5, 0)
(1, 1, 0)	(1, 5, 0)	6	No	(1, 5, 0)
(1, 0, 0)	(1, 5, 0)	1	No	(1, 5, 0)
(-1, 1, 0)	(1, 5, 0)	4	No	(1, 5, 0)
(-1, 1, -2)	(1, 5, 0)	4	No	(1, 5, 0)
(-1, 0, -2)	(1, 5, 0)	-1	Yes	(0, 5, -2)

end epoch 5

Adjusted pattern	Weight applied	w(n)x(n)	Update?	New weight
(1, 1, 2)	(0, 5, -2)	1	No	(0, 5, -2)
(1, 1, 0)	(0, 5, -2)	5	No	(0, 5, -2)
(1, 0, 0)	(0, 5, -2)	0	Yes	(1, 5, -2)

Adjusted pattern	Weight applied	w(n)x(n)	Update?	New weight
(-1, 1, 0)	(1, 5, -2)	4	No	(1, 5, -2)
(-1, 1, -2)	(1, 5, -2)	8	No	(1, 5, -2)
(-1, 0, -2)	(1, 5, -2)	3	No	(1, 5, -2)

end epoch 6

Adjusted pattern	Weight applied	w(n)x(n)	Update?	New weight
(1, 1, 2)	(1, 5, -2)	2	No	(1, 5, -2)
(1, 1, 0)	(1, 5, -2)	6	No	(1, 5, -2)
(1, 0, 0)	(1, 5, -2)	1	No	(1, 5, -2)
(-1, 1, 0)	(1, 5, -2)	4	No	(1, 5, -2)
(-1, 1, -2)	(1, 5, -2)	8	No	(1, 5, -2)
(-1, 0, -2)	(1, 5, -2)	3	No	(1, 5, -2)

no change

bias is 1