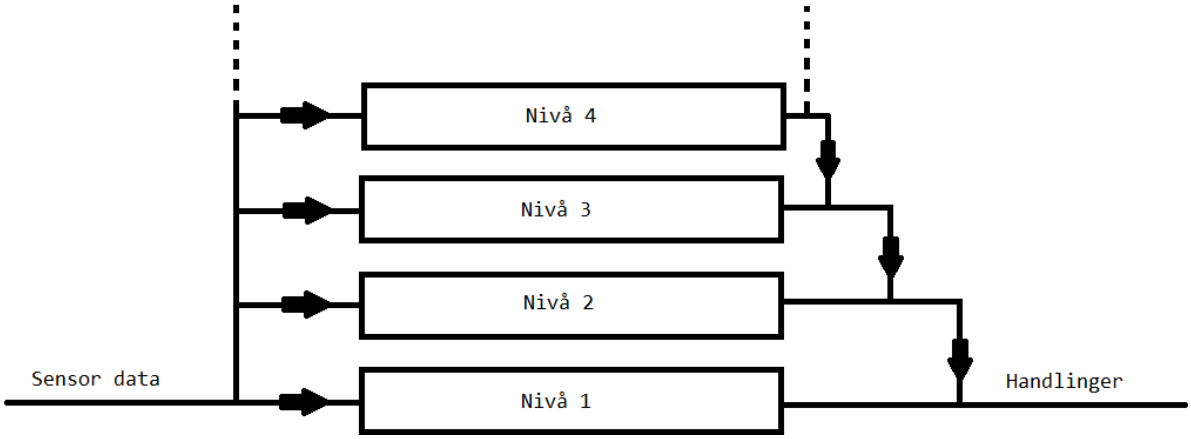
Oppgave 1:

a) I Brooks "subsumption architecture" er det ikke noe intern representasjon av verden. Selve verden er modellen. Brooks mente at kompleks oppførsel ikke trenger å bety at selve systemet er komplekst. I stedet bryter Brooks ned målet til modellen i enklere oppførsel og systemet etterligner menneskelig oppførsel på en mer evolusjonær måte. Det vil si at mer kompleks oppførsel bygger på enklere oppførsel i lavere nivåer. I stedet for å bruke en symbolsk representasjon av verden og oppdatere og agere i forhold til det. Physical symbol systems har vanskeligheter å "grounde" manipulasjonen av symboler i den fysiske verden, men subsumption klarer å vise at selve verden er en god nok representasjon og at fysisk interaksjon med den virkelige verden er en god nok kilde for inputs til systemet.

b)



Hvert lag representer mer kompleks oppførsel jo høyer laget er i arkitekturen. Lag integrere/kombinerer også lavere nivå til mer helhetlig og kompleks oppførsel. Lagene arbeider i parallelt og skaper output samtidig, men høyere nivåer har mulighet til å inkludere (subsume/override) output/oppførsel fra lavere nivå. Lavere nivå fungerer derfor slik som de er ment til å fungere selv uten de høyere nivåene.
Hvert lag består av AFSM (augmented finite state machines) som lar lagene interface med andre moduler/AFSMs, som for eksempel i andre nivåer.
Input data kan bli stoppet (supressed) og output data fra ett lag kan bli tapt (inhibited). Som for eksempel output fra en AFSM i lag 1 kan bli "inhibited" av en AFSM i lag 2 slik at den neste AFSM i nivå 1 som "egentlig" skulle ha den dataen ikke får den, (input suppressed).

Oppgave 2:

a)

```
class perceptron():

    def __init__(self, lrate):
        self.learning_rate = lrate
        self.treshold = random.uniform(-0.5, 0.5)
        self.weights = np.random.uniform(low=(-0.5), high=0.5, size=(2))

    def activation(self, inputs):
        self.inputs = inputs
        weighted_sum = 0
        for i, w in enumerate(self.weights):
            zum = self.inputs[i] * self.weights[i] - self.treshold
            weighted_sum += zum

        perceptron_output = 1 if weighted_sum > 0 else 0
        return perceptron_output

    def update_weights(self, prediction, label):
        self.error = label - prediction
        for i, w in enumerate(self.weights):
            delta_rule = self.learning_rate * self.error * self.inputs[i]
            self.weights[i] += delta_rule
```

b)

AND labels:

```
Initial weights: [-0.40027146  0.31206176]
Epcoh #1
############################################################
Weights @ epoch #1 & iteration #1: [-0.40027146  0.31206176]
Weights @ epoch #1 & iteration #2: [-0.40027146  0.21206176]
Weights @ epoch #1 & iteration #3: [-0.50027146  0.21206176]
Weights @ epoch #1 & iteration #4: [-0.50027146  0.21206176]
############################################################
Epcoh #2
############################################################
Weights @ epoch #2 & iteration #1: [-0.50027146  0.21206176]
Weights @ epoch #2 & iteration #2: [-0.50027146  0.11206176]
Weights @ epoch #2 & iteration #3: [-0.60027146  0.11206176]
Weights @ epoch #2 & iteration #4: [-0.60027146  0.11206176]
############################################################
Epcoh #3
############################################################
Weights @ epoch #3 & iteration #1: [-0.60027146  0.11206176]
Weights @ epoch #3 & iteration #2: [-0.60027146  0.01206176]
Weights @ epoch #3 & iteration #3: [-0.70027146  0.01206176]
Weights @ epoch #3 & iteration #4: [-0.70027146  0.01206176]
```

```
############################################################
Epcoh #4
############################################################
Weights @ epoch #4 & iteration #1: [-0.70027146  0.01206176]
Weights @ epoch #4 & iteration #2: [-0.70027146 -0.08793824]
Weights @ epoch #4 & iteration #3: [-0.70027146 -0.08793824]
Weights @ epoch #4 & iteration #4: [-0.60027146  0.01206176]
############################################################
```

OR labels:

```
Initial weights: [-0.31440872  0.14745699]
Epcoh #1
############################################################
Weights @ epoch #1 & iteration #1: [-0.31440872  0.14745699]
Weights @ epoch #1 & iteration #2: [-0.31440872  0.14745699]
Weights @ epoch #1 & iteration #3: [-0.21440872  0.14745699]
Weights @ epoch #1 & iteration #4: [-0.11440872  0.24745699]
############################################################
Epcoh #2
############################################################
Weights @ epoch #2 & iteration #1: [-0.11440872  0.24745699]
Weights @ epoch #2 & iteration #2: [-0.11440872  0.24745699]
Weights @ epoch #2 & iteration #3: [-0.01440872  0.24745699]
Weights @ epoch #2 & iteration #4: [-0.01440872  0.24745699]
############################################################
Epcoh #3
############################################################
Weights @ epoch #3 & iteration #1: [-0.01440872  0.24745699]
Weights @ epoch #3 & iteration #2: [-0.01440872  0.24745699]
Weights @ epoch #3 & iteration #3: [0.08559128 0.24745699]
Weights @ epoch #3 & iteration #4: [0.08559128 0.24745699]
############################################################
Epcoh #4
############################################################
Weights @ epoch #4 & iteration #1: [0.08559128 0.24745699]
Weights @ epoch #4 & iteration #2: [0.08559128 0.24745699]
Weights @ epoch #4 & iteration #3: [0.18559128 0.24745699]
Weights @ epoch #4 & iteration #4: [0.18559128 0.24745699]
############################################################
```

Når man endrer verdien på de initielle vektene og treshold verdien så får man forskjellige vekter på slutten, selvom man kjører mange epochs.

c)

Aktivering:

$$y_3 = \text{Sigmoid}(x_1 w_{13} + x_2 w_{23} - \theta_3)$$

$$= \frac{1}{1 + e^{-(0 \times 0{,}5 + 1 \times 0.0 - 1 \times 0.8)}} = 0.31$$

$y_3 = 0.31$

$$y_4 = \text{Sigmoid}(x_1 w_{14} + x_2 w_{24} - \theta_4)$$

$$= \frac{1}{1 + e^{-(0 \times 0{,}0 + 1 \cdot 0.9 - 1 \times (-0.1)}} = 0.7311$$

$y_4 = 0.7311$

Kalkulere resultat i output laget:

$$y_5 = \text{Sigmoid}(y_3 w_{35} + y_4 w_{45} - \theta 5)$$

$$= \frac{1}{1 + e^{-(0.31 \cdot 0.4 + 0.7311 \cdot -1.2 - 1 \cdot 0.3)}} = 0.2586$$

$y_5 = 0.2586$

$$y_6 = \text{Sigmoid}(y_3 w_{36} + y_4 w_{46} - \theta 6)$$

$$= \frac{1}{1 + e^{-(0.31 \cdot 1.0 + 0.7311 \cdot 1.1 - 1 \cdot 0{,}5)}} = 0.6489$$

$e_{Y5} = d_5 - Y_5 = 0 - 0.2586 = -0.2586$

$e_{Y6} = d_6 - Y_6 = 1 - 0.6489 = 0.3511$

Vekt Trening:

Error gradient for Y_5:

$\delta_5 = Y_5(1 - Y_5)e = 0.2586 \times (1 - 0.2586) \times -0.2586$

$\delta_5 = -0.0496$

Error gradient for Y_6:

$\delta_6 = Y_6(1 - Y_6)e = 0.6489(1 - 0.6489)\,0.3511$

$\delta_6 = 0.08$

Determine the weight corrections: LR, $\alpha = 0.1$

$\Delta W_{35} = \alpha \cdot Y_3 \cdot \delta_5 = 0.1 \cdot 0.31 \cdot (-0.0496) = -0.0015$

$\Delta W_{36} = \alpha \cdot Y_3 \cdot \delta_6 = 0.1 \cdot 0.31 \cdot (0.08) = 0.0025$

$\Delta \theta_5 = \alpha \cdot (-1) \cdot \delta_5 = 0.1 \cdot (-1) \cdot (-0.0496) = 0.00496$

$\Delta W_{45} = \alpha \cdot Y_4 \cdot \delta_5 = 0.1 \cdot 0.7311 \cdot (-0.0496) = -0.0036$

$\Delta W_{46} = \alpha \cdot Y_4 \cdot \delta_6 = 0.1 \cdot 0.7311 \cdot (0.08) = 0.0058$

$\Delta \theta_6 = \alpha \cdot (-1) \cdot \delta_6 = 0.1 \cdot (-1) \cdot 0.08 = -0.008$

Error gradients for node 3 og 4 i hidden layer: Guessing!? Need help!

$$\delta_3 = y_3(1-y_3) \cdot (\delta_5 \cdot w_{35} + \delta_6 \cdot w_{36})$$

$$= 0.31(1-0.31) \cdot (-0.0496 \cdot 0.4 + 0.08 \cdot 1,0)$$

$$\boxed{\delta_3 = 0.0129}$$

$$\delta_4 = y_4(1-y_4) \cdot (\delta_5 \cdot w_{45} + \delta_6 \cdot w_{46})$$

$$= 0.7311(1-0.7311) \cdot (-0.0496 \cdot -1,2 + 0.08 \cdot 1,1)$$

$$\delta_4 = 0.029$$

Determine weight corrections:

$$\Delta w_{13} = \alpha \cdot x_1 \cdot \delta_3 = 0.1 \cdot 0 \cdot 0.0129 = 0$$

$$\Delta w_{14} = \alpha \cdot x_1 \cdot \delta_4 = 0.1 \cdot 0 \cdot 0.029 = 0$$

$$\Delta \theta_3 = \alpha \cdot (-1) \cdot \delta_3 = 0.1 \cdot (-1) \cdot 0.0129 = -0.0013$$

$$\Delta w_{23} = \alpha \cdot x_2 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0129 = 0.0013$$

$$\Delta w_{24} = \alpha \cdot x_2 \cdot \delta_4 = 0.1 \cdot 1 \cdot 0.029 = 0.0029$$

$$\Delta \theta_4 = \alpha \cdot (-1) \cdot \delta_4 = 0.1 \cdot (-1) \cdot 0.029 = -0.0029$$

$W_{13} = W_{13} + \Delta W_{13} = 0.5 + 0 = \boxed{0.5}$

$W_{14} = W_{14} + \Delta W_{14} = 0.0 + 0 = \boxed{0.0}$

$W_{23} = W_{23} + \Delta W_{23} = 0.0 + 0.0013 = \boxed{0.0013}$

$W_{24} = W_{24} + \Delta W_{24} = 0.9 + 0.0029 = \boxed{0.9029}$

$W_{35} = W_{35} + \Delta W_{35} = 0.4 + (-0.0015) = \boxed{0.395}$

$W_{36} = W_{36} + \Delta W_{36} = 1.0 + 0.0025 = \boxed{1.0025}$

$W_{45} = W_{45} + \Delta W_{45} = -1.2 + (-0.0036) = \boxed{-1.2036}$

$W_{46} = W_{46} + \Delta W_{46} = 1.1 + 0.0058 = \boxed{1.1058}$

$\theta_3 = \theta 3 + \Delta \theta 3 = 0.8 + (-0.0013) = \boxed{0.7987}$

$\theta 4 = \theta 4 + \Delta \theta 4 = -0.1 + (-0.0029) = \boxed{-0.1029}$

$\theta 5 = \theta 5 + \Delta \theta 5 = 0.3 + 0.00496 = \boxed{0.30496}$

$\theta 6 = \theta 6 + \Delta \theta 6 = 0.6 + (-0.008) = \boxed{0.592}$

Oppgave 3:

a) Med 6 noder så begynner gjenskapningen in output noden å hoppe litt mer, dog veldig lite. For eksempel sender jeg inn tallet «8» og får output 6.9, men det skjer ikke så ofte. Nede på 4 noder ser det ut som det er større forskjelligheter mellom svaret i output noden mellom «runs. Nede på 3 noder i hidden-layer så begynner det å slite med å gjenskape input konsistent i output, ved høye hele tall som input.
Konklusjon: For å skape konsekvent godt resultat burde ikke antall noder i hidden-layer være mindre enn 6 .

b) Det nevrale nettet har gjenskapt et floating point tall (desimal tall) som. Nøyaktigheten til outputen kan bestemmes på hvor nærme avrundingen av desimal tallet er til input heltallet. For eksempel når vi «aktiverer» heltall 1 får vi ofte 0.999... eller 1.019... som return verdi. Hvis vi hadde gjort om dette til heltall hadde vi fått det samme som inputtet. Samme gjelder heltall 8 som ofte får return verdi på 7.6... eller lignende som kan bli rundet opp til 8.

c) Tall over 8 får dårlig gjenskapelse. Negative tall er også dårlig og gjenskapelsen har som regel ikke noe konsekvent gjenskapelses verdi.
Desimal tall derimot ser ut som har god gjenskapelse i output laget. Så lenge desimal tallet er innenfor rekkevidden til datasettet så ser det ut som det klarer å gjenskape et tall som er innenfor en akseptabel rekkevidde i forhold til input tallet. Det skal sies at det kan være vanskelig å si om dette egentlig fungerer siden det er vanskelig å runde av et desimal tall til ett annet. Hvis input tallet er 6.7 og outpt er mellom 6.5 eller 6.9 så kan man kanskje si at det er en god gjenskapelse, men med en gang man har med desimal tall å gjøre så burde man kun akseptere en høyere nøyaktighet i output laget derfor vil jeg konkludere med at desimal tall også ikke fungerer med denne auto-encoderen.

Kildekode:

```
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.tools.shortcuts import buildNetwork
from pybrain.structure import TanhLayer
from pybrain.datasets import SupervisedDataSet
from pybrain.structure import SoftmaxLayer

dataset = SupervisedDataSet(1,1)
for sample in range(1,9):
    dataset.addSample((sample,), (sample,))

# Endrer antall hidden noder gjennom
net = buildNetwork(1,6,1, bias=True, hiddenclass=TanhLayer)

trainer = BackpropTrainer(net, dataset)

x = trainer.trainUntilConvergence(verbose=False, validationProportion=0.15,
                                  maxEpochs=1000, continueEpochs=10)

# Endre hvilket heltall man skal sjekke gjenskapning på
y = net.activate([6.7])
print(y)
```