**import** **math**

**import** **random**

random.seed(0)

**def** **rand**(a, b):

**return** (b - a) \* random.random() + a

**def** **make\_matrix**(m, n, fill=**0.0**):

mat = []

**for** i in range(m):

mat.append([fill] \* n)

**return** mat

**def** **sigmoid**(x):

**return** **1.0** / (**1.0** + math.exp(-x))

**def** **sigmoid\_derivative**(x):

**return** x \* (1 - x)

**class** **BPNeuralNetwork**:

**def** **\_\_init\_\_**(self):

self.input\_n = 0

self.hidden\_n = 0

self.output\_n = 0

self.input\_cells = []

self.hidden\_cells = []

self.output\_cells = []

self.input\_weights = []

self.output\_weights = []

self.input\_correction = []

self.output\_correction = []

**def** **setup**(self, ni, nh, no):

self.input\_n = ni + 1

self.hidden\_n = nh

self.output\_n = no

# init cells

self.input\_cells = [**1.0**] \* self.input\_n

self.hidden\_cells = [**1.0**] \* self.hidden\_n

self.output\_cells = [**1.0**] \* self.output\_n

# init weights

self.input\_weights = make\_matrix(self.input\_n, self.hidden\_n)

self.output\_weights = make\_matrix(self.hidden\_n, self.output\_n)

# random activate

**for** i in range(self.input\_n):

**for** h in range(self.hidden\_n):

self.input\_weights[i][h] = rand(-**0.2**, **0.2**)

**for** h in range(self.hidden\_n):

**for** o in range(self.output\_n):

self.output\_weights[h][o] = rand(-**2.0**, **2.0**)

# init correction matrix

self.input\_correction = make\_matrix(self.input\_n, self.hidden\_n)

self.output\_correction = make\_matrix(self.hidden\_n, self.output\_n)

**def** **predict**(self, inputs):

# activate input layer

**for** i in range(self.input\_n - 1):

self.input\_cells[i] = inputs[i]

# activate hidden layer

**for** j in range(self.hidden\_n):

total = **0.0**

**for** i in range(self.input\_n):

total += self.input\_cells[i] \* self.input\_weights[i][j]

self.hidden\_cells[j] = sigmoid(total)

# activate output layer

**for** k in range(self.output\_n):

total = **0.0**

**for** j in range(self.hidden\_n):

total += self.hidden\_cells[j] \* self.output\_weights[j][k]

self.output\_cells[k] = sigmoid(total)

**return** self.output\_cells[:]

**def** **back\_propagate**(self, case, label, learn, correct):

# feed forward

self.predict(case)

# get output layer error

output\_deltas = [**0.0**] \* self.output\_n

**for** o in range(self.output\_n):

error = label[o] - self.output\_cells[o]

output\_deltas[o] = sigmoid\_derivative(self.output\_cells[o]) \* error

# get hidden layer error

hidden\_deltas = [**0.0**] \* self.hidden\_n

**for** h in range(self.hidden\_n):

error = **0.0**

**for** o in range(self.output\_n):

error += output\_deltas[o] \* self.output\_weights[h][o]

hidden\_deltas[h] = sigmoid\_derivative(self.hidden\_cells[h]) \* error

# update output weights

**for** h in range(self.hidden\_n):

**for** o in range(self.output\_n):

change = output\_deltas[o] \* self.hidden\_cells[h]

self.output\_weights[h][o] += learn \* change + correct \* self.output\_correction[h][o]

self.output\_correction[h][o] = change

# update input weights

**for** i in range(self.input\_n):

**for** h in range(self.hidden\_n):

change = hidden\_deltas[h] \* self.input\_cells[i]

self.input\_weights[i][h] += learn \* change + correct \* self.input\_correction[i][h]

self.input\_correction[i][h] = change

# get global error

error = **0.0**

**for** o in range(len(label)):

error += **0.5** \* (label[o] - self.output\_cells[o]) \*\* 2

**return** error

**def** **train**(self, cases, labels, limit=10000, learn=**0.05**, correct=**0.1**):

**for** j in range(limit):

error = **0.0**

**for** i in range(len(cases)):

label = labels[i]

case = cases[i]

error += self.back\_propagate(case, label, learn, correct)

**def** **test**(self):

cases = [

[0, 0],

[0, 1],

[1, 0],

[1, 1],

]

labels = [[0], [1], [1], [0]]

self.setup(2, 5, 1)

self.train(cases, labels, 10000, **0.05**, **0.1**)

**for** case in cases:

**print**(self.predict(case))

**if** \_\_name\_\_ == '\_\_main\_\_':

nn = BPNeuralNetwork()

nn.test()