```
import theano
import theano.tensor as T
import numpy as np

X = T.dmatrix('X')
y = T.dmatrix('y')

W1 = theano.shared(np.random.randn(2, 2))
b1 = theano.shared(np.zeros((2,)))
W2 = theano.shared(np.random.randn(2, 1))
b2 = theano.shared(np.zeros((1,)))

z1 = T.nnet.sigmoid(T.dot(X, W1) + b1)
z2 = T.nnet.sigmoid(T.dot(z1, W2) + b2)

loss = T.mean((y - z2)**2)
params = [W1, b1, W2, b2]
grads = T.grad(loss, params)

updates = [(p, p - 0.1 * g) for p, g in zip(params, grads)]

train = theano.function([X, y], loss, updates=updates)
predict = theano.function([X], z2)

data_X = np.array([[0,0],[0,1],[1,0],[1,1]])
data_y = np.array([[0],[1],[1],[0]])

for i in range(10000):
    train(data_X, data_y)

print("Output:", np.round(predict(data_X)))
```

```
Output: [[0.]
        [1.]
        [1.]
        [0.]]
```

==

```
import lasagne
import theano
import theano.tensor as T
import numpy as np

input_var = T.matrix('inputs')
target_var = T.matrix('targets')

l_in = lasagne.layers.InputLayer(shape=(None, 2), input_var=input_var)
l_hidden = lasagne.layers.DenseLayer(l_in, num_units=2, nonlinearity=lasagne.nonlinearities.sigmoid)
l_out = lasagne.layers.DenseLayer(l_hidden, num_units=1, nonlinearity=lasagne.nonlinearities.sigmoid)

prediction = lasagne.layers.get_output(l_out)
loss = lasagne.objectives.squared_error(prediction, target_var).mean()
params = lasagne.layers.get_all_params(l_out, trainable=True)
updates = lasagne.updates.sgd(loss, params, learning_rate=0.1)

train_fn = theano.function([input_var, target_var], loss, updates=updates)
predict_fn = theano.function([input_var], prediction)

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

for _ in range(10000):
    train_fn(X, y)

print("Output:", np.round(predict_fn(X))) // giv output
```

OUTPUT

```
Output: [[0.]
           [1.]
           [1.]
           [0.]]
```

==

```
import mxnet as mx
from mxnet import gluon, nd, autograd
import numpy as np

X = nd.array([[0,0],[0,1],[1,0],[1,1]])
y = nd.array([[0],[1],[1],[0]])

net = gluon.nn.Sequential()
net.add(gluon.nn.Dense(2, activation='sigmoid'))
net.add(gluon.nn.Dense(1, activation='sigmoid'))
net.initialize()

loss_fn = gluon.loss.L2Loss()
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate': 0.1})

for epoch in range(10000):
    with autograd.record():
        output = net(X)
        loss = loss_fn(output, y)
    loss.backward()
    trainer.step(batch_size=4)

print("Output:", nd.round(net(X)))
```

OUTPUT

```
[[0.]
 [1.]
 [1.]
 [0.]]
```

```
===================================================================================================
import cntk as C
import numpy as np

X = np.array([[0,0],[0,1],[1,0],[1,1]], dtype=np.float32)
y = np.array([[0],[1],[1],[0]], dtype=np.float32)

input_var = C.input_variable(2)
label_var = C.input_variable(1)

model = C.layers.Sequential([
    C.layers.Dense(2, activation=C.sigmoid),
    C.layers.Dense(1, activation=C.sigmoid)
])

output = model(input_var)
loss = C.squared_error(output, label_var)
learner = C.sgd(model.parameters, lr=0.1)
trainer = C.Trainer(output, (loss, None), [learner])

for _ in range(10000):
    trainer.train_minibatch({input_var: X, label_var: y})

print("Output:", np.round(output.eval({input_var: X})))
```

OUTPUT

```
Output:
[[0.]
 [1.]
 [1.]
 [0.]]
```