Label 1 : back-propagation

Outline

1. Introduction

2. Experiment setups
   A. Neural network
   B. Sigmoid functions
   C. Backpropagation

3. Results of testing

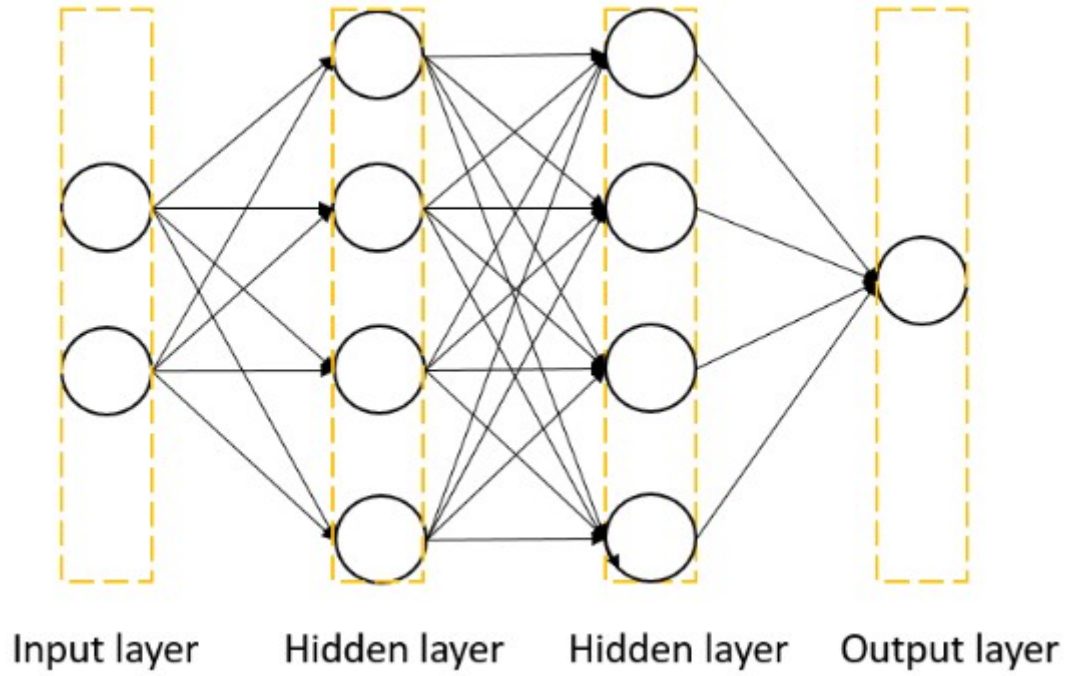
4. Discussion

1. Introduction

We need to classify one point is red or blue with two ruler, linear or XOR in this experiment.

2. Experiment setups

My neural network structure is 4 x 4 x1. One point’s x, y coordinate as Input data and one number to predict this is red or blue from output. If number is greater then 0.5, point would be blue, otherwise is red.

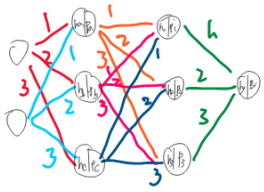


Activation function use sigmoid in this structure. Because it has domain of all real numbers, with return value monotonically increasing most often from 0 to 1 or alternatively from −1 to 1, depending on convention.

```
def sigmoid(self, h):
    return 1.0/ (1.0 + np.exp(-h))

def derivative_sigmoid(self, h):
    return self.sigmoid(h)*(1.0 - self.sigmoid(h))
```

Under this picture is my backpropagation derivation.



$$\frac{\partial L}{\partial W_1} = \frac{\partial h_y}{\partial W_1} \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y} = P_1 \times \left[G'(h_y) \times -(\hat{y} - P_y) \right]$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial h_y}{\partial W_2} \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y} = P_2 \times G'(h_y) \times -(\hat{y} - P_y)$$

$$\frac{\partial L}{\partial W_3} = \frac{\partial h_y}{\partial W_3} \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y} = P_3 \times G'(h_y) \times -(\hat{y} - P_y)$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial h_1}{\partial W_1} \times \left[\frac{\partial P_1}{\partial h_1} \times \frac{\partial h_y}{\partial P_1} \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y} \right] = P_a \times \left[G'(h_1) \times W_1 \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y} \right]$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial h_1}{\partial W_1} \times \frac{\partial P_1}{\partial h_1} \times \frac{\partial h_y}{\partial P_1} \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y} = P_b \times G'(h_1) \times W_1 \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y}$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial h_1}{\partial W_1} \times \frac{\partial P_1}{\partial h_1} \times \frac{\partial h_y}{\partial P_1} \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y} = P_c \times G'(h_1) \times W_1 \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y}$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial h_2}{\partial W_2} \times \left[\frac{\partial P_2}{\partial h_2} \times \frac{\partial h_y}{\partial P_2} \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y} \right] = P_a \times \left[G'(h_y) \times W_2 \times \frac{\partial P_y}{\partial h_y} \times \frac{\partial L}{\partial P_y} \right]$$

I find it has regular a way can be write in my script from derivation, so i decide to build a general class to flexible adjustment structure.

Learning rate is setting 0.8, and one point input network training once, all point run over equal one epoch.

Initialize weight is important. Use Glorot Initialization would be more then customize random number has training representation.

random (-u, u) : u = sqrt(6) / sqrt(inputUnits + units)

3. Results of testing

A. Linear of testing:

a. Total loss:

epoch: 5000 loss: 0.009309802269452475

epoch: 10000 loss: 8.063212229710132e-06

epoch: 15000 loss: 8.706083618607772e-05

epoch: 20000 loss: 0.004826199649409059

epoch: 25000 loss: 0.0048628656806365975

epoch: 30000 loss: 0.004593744924309925

epoch: 35000 loss: 0.005231838442809501

epoch: 40000 loss: 0.00014446675441003956

epoch: 45000 loss: 1.2937993491673358e-07

epoch: 50000 loss: 5.660083112234522e-08

epoch: 55000 loss: 0.008290286650940885

epoch: 60000 loss: 3.4251949502151027e-06

epoch: 65000 loss: 0.010870030605284407

epoch: 70000 loss: 0.007121522130459326

epoch: 75000 loss: 0.0033678135058469363

epoch: 80000 loss: 0.00018345575977160082

epoch: 85000 loss: 0.0050034843195890764

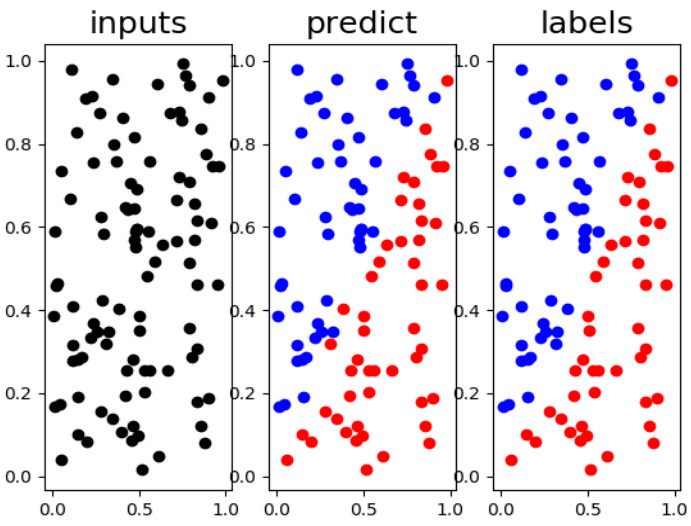
epoch: 90000 loss: 0.0004703685446036574

epoch: 95000 loss: 4.091936033379485e-05

epoch: 100000 loss: 0.002695977759741197

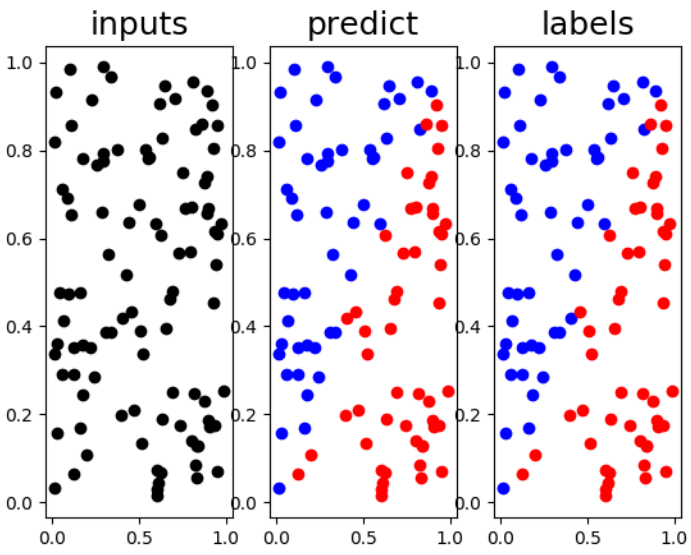
b. Predict:

epoch: 5000



[0.9996915830610462, 6.479708040066552e-06, 0.999712879847084, 0.9997124663674963, 0.9997056273204075, 6.501658317618151e-06, 0.9997121622110162, 5.072886317947441e-05, 0.9997112221395771, 6.478818311298078e-06, 0.9997074951600222, 6.540742832422742e-06, 0.9996703683094363, 0.9997122041575586, 0.9997051031294976, 0.9997121176227354, 0.9997124938909107, 0.36153784449915666, 6.543005238480125e-06, 0.7600596362535794, 0.9997123087231887, 6.514149294402335e-06, 0.9997114260975086, 0.999558657657793, 6.596816051092272e-06, 0.9997098244514865, 0.9996923942005707, 6.485323922115395e-06, 0.9997022956070163, 0.9997072865669153, 0.9997075714346636, 0.9997118880278957, 0.9800595346802804, 0.18899699203572437, 0.00026498371376057305, 6.641596439017494e-06, 6.478970578978408e-06, 0.9997064763938587, 0.999712169824505, 0.83080788649775, 0.00010548185054593272, 0.9997104147992502, 6.593835474202998e-06, 0.9997054377941638, 6.493548160092267e-06, 0.999705989550693, 0.9997089367785769, 6.5040673484961096e-06, 7.2217699849583504e-06, 0.9994078444672261, 7.210454543200948e-06, 6.480891099215733e-06, 6.4792121160681794e-06, 0.006558920223821045, 6.80152501703568e-06, 6.536670311760466e-06, 6.506380005358104e-06, 0.9997107225863244, 0.9997112852256904, 0.9997074094531038, 0.9996938908243704, 0.9996863477699451, 0.9997051045016698, 0.999708815156141, 6.630337487637592e-06, 6.514067453358163e-06, 6.741715204792157e-06, 6.47933099734905e-06, 0.9997101978982437, 0.9997080353029829, 7.657071742972482e-06, 0.9997118162648101, 6.487962973674174e-06, 0.999711149340431, 0.9997122098487625, 6.610085888552697e-06, 6.479450939259928e-06, 7.3125567492680205e-06, 6.724558378474497e-06, 7.0698009442395524e-06, 0.9997121154231651, 6.519315796151537e-06, 9.163612638614966e-06, 6.515298641184998e-06, 0.00036024750768413205, 0.999712801560313, 0.9997083192518158, 8.412393365460592e-06,6.590699732171638e-06, 0.9997069164833179, 6.5214463725599184e-06, 6.49543582489704e-06, 6.570066503883441e-06, 0.9997090493134053, 6.482435492958659e-06, 0.9997117573037856, 6.479405096117122e-06, 0.9997026080308987, 0.9997078359507531, 0.9996896221959362]

epoch: 100000



[0.9999003770450099, 7.2688609668178764e-06, 0.9998999565792038, 6.925542777983124e-06, 6.931223678396073e-06, 6.92795465094505e-06, 6.924008411889442e-06, 7.0413579955732755e-06, 0.9998994864016416, 0.0010880884265534482, 7.048119684802587e-06, 6.928545003252501e-06, 0.9999000586316562, 2.6941357811802294e-05, 0.9999002161315168, 6.959799940932827e-06, 0.9995699644108639, 7.208823220012675e-06, 0.9998947603680112, 6.938888768875812e-06, 7.120904056105586e-06, 0.9999004637825589, 6.9225040624779975e-06, 0.9996136904894877, 7.3367087723088685e-06, 0.9997187991625268, 6.97520357905445e-06, 6.959991853920593e-06,

6.953251335709542e-06, 0.9999001357789162, 0.9998997372937312, 6.9233380862001265e-06, 0.01171595596997522, 0.9999004625722793, 7.0289355260284485e-06, 0.999876696273913, 0.9999000876867543, 6.928366870143189e-06, 0.9998999195545808, 0.9997277076947267, 0.9999002251774763, 0.2658014279235869, 6.9393662188178875e-06, 0.9998998653417063, 6.923437289575669e-06, 0.9999000308802299, 0.9996386060031037, 6.9224422633591594e-06, 0.9998997683882013, 0.9998998622941346, 6.077353745538881e-05, 6.922319157626989e-06, 6.92288561511486e-06, 6.936457444192745e-06, 0.9999005291950391, 0.9999004307826312, 0.9999004440069402, 6.9220256952115055e-06, 0.9998998544352055, 0.999886940201455, 6.922568766125246e-06, 6.943044594809262e-06, 0.9998997653886537, 6.940901217445235e-06, 0.9999001644254403, 0.9999002180588361, 1.193246847252403e-05, 7.026017367880001e-06, 0.9999000385452977, 0.9999004456215005, 0.9999001967888125, 6.924686651986202e-06, 0.9999000765125647, 0.9999001903655109, 0.9999005149380872, 0.9998804694752775, 0.9999005298952491, 0.9998999757935557, 6.955765012929564e-06, 6.923850342236694e-06, 6.923324153530052e-06, 0.9999001399285974, 0.9998999989521948, 0.00044574623094277305, 0.9999001527710418, 6.922678431056153e-06, 6.9326903004006945e-06, 6.95305211496192e-06, 0.9999001964180807, 0.9998999653762787, 6.975623421685083e-06, 0.9999005203448845, 6.923931164043859e-06, 7.103567698346358e-06, 0.9999004256413623, 0.9999005186772082, 6.994483731052938e-06, 7.104567578365233e-06, 0.9971198685157807, 2.2171192795934377e-05]

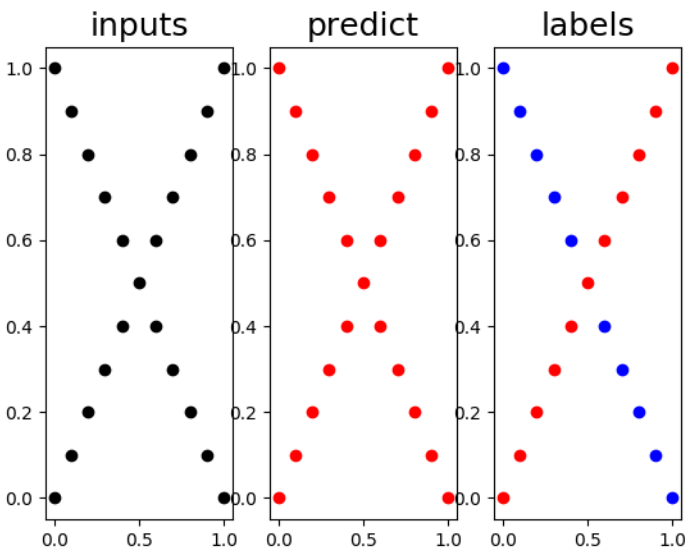
A. XOR of testing:

a. Total loss:

epoch: 200 loss: 0.1247796514953585
epoch: 400 loss: 0.12476356903269427
epoch: 600 loss: 0.12470897254039803
epoch: 800 loss: 0.12413153961194315
epoch: 1000 loss: 0.09409614814215185
epoch: 1200 loss: 0.0065287365892772
epoch: 1400 loss: 0.0006915673483860139
epoch: 1600 loss: 0.0003362998908145234
epoch: 1800 loss: 0.00021616166655529298
epoch: 2000 loss: 0.0001571423780341893

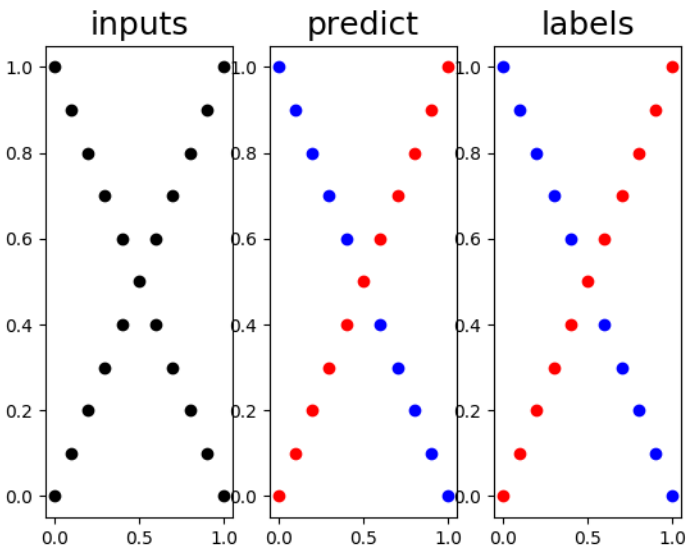
b. Predict:

epoch: 200



[0.48745912489398713, 0.4888836036255879, 0.48771010017185207, 0.4888561510273682, 0.4879721241677352, 0.48883075703889955, 0.4882395018230631, 0.4888074081226391, 0.4885062949866692, 0.48878607269324337, 0.48876670009719003, 0.48901540949104233, 0.48874921980178276, 0.4892479097955506, 0.488733540834743, 0.4894606843467531, 0.48871955151332186, 0.48965130577253513, 0.4887071194977229, 0.48981842416158583, 0.48869609219754856]

epoch: 2000



[0.0063920095556538326, 0.9978329735441336, 0.004920649909509832, 0.9978468800659442, 0.0054692791744468195, 0.997835438597714, 0.010326919803371186, 0.9974973996936815, 0.0279451056773365, 0.9601818811435494, 0.039519287905405216, 0.02734599047167511, 0.9647023053937687, 0.015836242572558334, 0.9981773938847133, 0.010105067654936123, 0.9983994451116026, 0.007387813990606117, 0.9982594941828992, 0.005993702913148017, 0.998040817099751 6]

4. Discussion

Coding this experiment first time, my training is not good. My total loss always convergence in 0.12 when I try very ways. I’m gratitude ta, he tell me, my learning rate is too small, but I try 0.5 before. Try 0.8 learning rate what amazing it can train very well. Because this experiment easy so learning rate too small it would be training not very well.

In backpropagation fist time, i comprehend wrong, compute backpropagation with update weight, so output layer weight update, then use this weight to update next layer. Resulting in training fail a lot of time.