ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

Lab 02&03

Xử lý ảnh số và video số 20_23

Giảng viên hướng dẫn – Nguyễn Mạnh Hùng

Thành phố Hồ Chí Minh - 2022

MỤC LỤC

THÔNG TIN SINH VIÊN

MSSV	Họ Tên	Email	Ghi chú
20120201	Phạm Gia Thông	20120201@student.hcmus.edu.vn	

I. Thực nghiệm Lab02

a) Tăng tốc độ học của mô hình

Tăng tốc độ học lần lượt ở các mức 0.5, 5, 15 (vẫn giữ kích thước số lớp input, hidden)

```
        print("#" + str(i) + " Loss: " + str(torch.mean((y - NN(X)) ** 2).detach().item()))
        # training with learning rate = 0.1
        NN.train(X, y, 0.5)
    # save weights
    NN.save_weights(NN, "NN")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
#958 Loss: 0.003638955531641841
#959 Loss: 0.00363855180330574 5
#960 Loss: 0.0036381424870342016
#961 Loss: 0.003637745976448059
#962 Loss: 0.0036373294424265623
#963 Loss: 0.003636932233348489
#964 Loss: 0.0036365229170769453
#965 Loss: 0.003636113600805402
#966 Loss: 0.0036357103381305933
#967 Loss: 0.003635307075455785
#968 Loss: 0.0036349035799503326
#969 Loss: 0.003634494496509433
#970 Loss: 0.003634092165157199
#971 Loss: 0.0036336828488856554
#972 Loss: 0.003633280051872134
#973 Loss: 0.003632868407294154
#990 Loss: 0.0036259808111935854
#991 Loss: 0.003625578014180064
#992 Loss: 0.0036251712590456 01
#993 Loss: 0.0036247659008949995
#994 Loss: 0.0036243610084056854
#995 Loss: 0.0036239561159163713
#996 Loss: 0.0036235495936125517
#997 Loss: 0.003623147262260318
#998 Loss: 0.0036227398086339235
#999 Loss: 0.0036223363131284714
Predict data based on trained weights:
Input:
tensor([0.3750, 1.0000, 0.5000])
Output:
tensor([0.9319])
```

```
        print("#" + str(i) + " Loss: " + str(torch.mean((y - NN(X)) ** 2).detach().item()))
        # training with learning rate = 0.1
        NN.train(X, y, 5)
    # save weights
    NN.save_weights(NN, "NN")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
#957 Loss: 0.0001888448023237288
#958 Loss: 0.0001886019017547369
#959 Loss: 0.0001883589429780841
#960 Loss: 0.0001881162606878206
#961 Loss: 0.0001878755138022825
#962 Loss: 0.0001876332680694759
#963 Loss: 0.0001873939303050097078
#964 Loss: 0.0001871528220362961 3
#965 Loss: 0.0001869131228886544 7
#966 Loss: 0.000186675213626585 9
#967 Loss: 0.0001864355872385203 8
#968 Loss: 0.0001861972996266558 8
#969 Loss: 0.0001859599287854507 6
#970 Loss: 0.0001857227325672283 8
#971 Loss: 0.0001854877773439511 7
#972 Loss: 0.0001852498535299673 7
#973 Loss: 0.0001850153203122317 8
#974 Loss: 0.000184780583367683
#975 Loss: 0.0001845471415435895 3
#976 Loss: 0.0001843131176428869 4
#977 Loss: 0.0001840808108681813
#978 Loss: 0.0001838478056015446 8
#979 Loss: 0.0001836160081438720 2
#980 Loss: 0.0001833834248827770 4
#996 Loss: 0.0001797497388906777
#997 Loss: 0.0001795256393961608 4
#998 Loss: 0.0001793033588910475 4
#999 Loss: 0.0001790812966646626 6
Predict data based on trained weights:
Input:
tensor([0.3750, 1.0000, 0.5000])
Output:
tensor([0.9776])
```

```
        print("#" + str(i) + " Loss: " + str(torch.mean((y - NN(X)) ** 2).detach().item()))
        # training with learning rate = 0.1
        NN.train(X, y, 15)
    # save weights
    NN.save_weights(NN, "NN")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
#973 Loss: 0.0001266073522856459
#974 Loss: 0.0001263304293388500 8
#975 Loss: 0.0001260548742720857 3
#976 Loss: 0.0001257803960470482 7
#977 Loss: 0.0001255075185326859 4
#978 Loss: 0.0001252342626685276 6
#979 Loss: 0.000124963829875923 7
#980 Loss: 0.000124692815006710 6
#981 Loss: 0.0001244222949026152 5
#982 Loss: 0.0001241543650394305 6
#983 Loss: 0.0001238873664988204 8
#984 Loss: 0.0001236192620126530 5
#985 Loss: 0.000123354679089970 9
#986 Loss: 0.0001230888738064095 4
#987 Loss: 0.0001228254259331151 8
#988 Loss: 0.0001225630257977172 7
#989 Loss: 0.0001222299839858897 03
#990 Loss: 0.0001220398116856813 4
#991 Loss: 0.0001217784592881798 7
#992 Loss: 0.0001215196971315890 6
#993 Loss: 0.0001212606221088208 3
#994 Loss: 0.0001210032714880071 6
#995 Loss: 0.0001207464447361417 1
#996 Loss: 0.0001204904983751475 8
#997 Loss: 0.0001202355197165161 4
#998 Loss: 0.0001199824546347372 2
#999 Loss: 0.000119719729178550 1875
Predict data based on trained weights:
Input:
tensor([0.3750, 1.0000, 0.5000])
Output:
tensor([0.9690])
```

b) Tăng tốc độ học kết hợp với thay đổi kích thước các lớp trong mô hình

Tăng tốc độ học lần lượt ở các mức 0.5, 5, 15 (thay đổi kích thước số lớp input, hidden)

```
# import PyTorch
import torch
# import Feed Forward Neural Network class from nn_simple module
from ffnn import *

# sample input and output value for training
X = torch.tensor(([2, 9, 0, 7], [1, 5, 1, 8], [3, 6, 2, 1], [3, 1, 2, 9]), dtype=torch.float)  # 4 X 4 tensor
y = torch.tensor(([90], [100], [88], [120]), dtype=torch.float)  # 4 X 1 tensor

# scale units by max value
X_max, _ = torch.max(X, 0)
X = torch.div(X, X_max)
y = y / 120  # for max test score is 100

# sample input x for predicting
x_predict = torch.tensor(([3, 8, 4, 5]), dtype=torch.float)  # 1 X 4 tensor
```

```
# Feed Forward Neural Network class
class FFNeuralNetwork(nn.Module):
    # initialization function
    def __init__(self, ):
        # init function of base class
        super(FFNeuralNetwork, self).__init__()

        # corresponding size of each layer
        self.inputSize = 4
        self.hiddenSize = 5
        self.outputSize = 1

        # random weights from a normal distribution
        self.W1 = torch.randn(self.inputSize, self.hiddenSize)  # 4 X 5 tensor
        self.W2 = torch.randn(self.hiddenSize, self.outputSize)  # 5 X 1 tensor
```

```
        print("#" + str(i) + " Loss: " + str(torch.mean((y - NN(X)) ** 2).detach().item()))
        # training with learning rate = 0.1
        NN.train(X, y, 0.5)
    # save weights
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
#966 Loss: 0.0014936664374545217
#967 Loss: 0.001492927665822208
#968 Loss: 0.0014921965776011348
#969 Loss: 0.001491460483521223
#970 Loss: 0.0014907284639775753
#971 Loss: 0.0014899959787726402
#972 Loss: 0.0014892679173499346
#973 Loss: 0.0014885378768667579
#974 Loss: 0.0014878123765811324
#975 Loss: 0.0014870867598801851
#976 Loss: 0.0014863645192235708
#977 Loss: 0.0014856402995064855
#978 Loss: 0.001484918873757124
#979 Loss: 0.001484201173298061
#980 Loss: 0.001483480678871274
#981 Loss: 0.0014827648410573602
#982 Loss: 0.0014820521464571357
#983 Loss: 0.0014813367743045092
#984 Loss: 0.0014806214021518826
#985 Loss: 0.0014799117343500257
#986 Loss: 0.0014792042784392834
#987 Loss: 0.0014784925151616335
#988 Loss: 0.0014777842443436384
#989 Loss: 0.0014770771376788616
#998 Loss: 0.001470774644985795
#999 Loss: 0.001470082439482212
Predict data based on trained weights:
Input:
tensor([0.3750, 1.0000, 0.5000, 0.6250])
Output:
tensor([0.7262])
```

```
        print("#" + str(i) + " Loss: " + str(torch.mean((y - NN(X)) ** 2).detach().item()))
        # training with learning rate = 0.1
        NN.train(X, y, 5)
    # save weights
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
#973 Loss: 6.272918108152226e-05
#974 Loss: 6.263982504606247e-05
#975 Loss: 6.255153857637197e-05
#976 Loss: 6.246229895623401e-05
#977 Loss: 6.237412890186533e-05
#978 Loss: 6.228692654985934e-05
#979 Loss: 6.219847273314372e-05
#980 Loss: 6.211051368154585e-05
#981 Loss: 6.202350050443783e-05
#982 Loss: 6.193670560605824e-05
#983 Loss: 6.184896483318888e-05
#984 Loss: 6.176268652779981e-05
#985 Loss: 6.167597894091159e-05
#986 Loss: 6.158945325296372e-05
#987 Loss: 6.150324770715088e-05
#988 Loss: 6.141776248114184e-05
#989 Loss: 6.133233546279371e-05
#990 Loss: 6.12464064033702e-05
#991 Loss: 6.116108852438629e-05
#992 Loss: 6.107586523285136e-05
#993 Loss: 6.099112579249777e-05
#994 Loss: 6.090694296290167e-05
#995 Loss: 6.082276377128437e-05
#996 Loss: 6.073825352359563e-05
#997 Loss: 6.065424531698227e-05
#998 Loss: 6.0569913330255076e-05
#999 Loss: 6.048692011972889e-05
Predict data based on trained weights:
Input:
tensor([0.3750, 1.0000, 0.5000, 0.6250])
Output:
tensor([0.6793])
```

```
        print("#" + str(i) + " Loss: " + str(torch.mean((y - NN(X)) ** 2).detach().item()))
        # training with learning rate = 0.1
        NN.train(X, y, 15)
    # save weights
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
#973 Loss: 1.776189310476184e-05
#974 Loss: 1.7734211724018678e-05
#975 Loss: 1.7706992366584018e-05
#976 Loss: 1.7679412849247456e-05
#977 Loss: 1.7652735550655052e-05
#978 Loss: 1.762498686730396e-05
#979 Loss: 1.759834231052082e-05
#980 Loss: 1.757166683091782e-05
#981 Loss: 1.7544807633385062e-05
#982 Loss: 1.7517912056064233e-05
#983 Loss: 1.749184593791142e-05
#984 Loss: 1.746523594192695e-05
#985 Loss: 1.7438711438444443e-05
#986 Loss: 1.741256710374728e-05
#987 Loss: 1.738640457915608e-05
#988 Loss: 1.73601511050947e-05
#989 Loss: 1.7334456060780212e-05
#990 Loss: 1.7308673704974353e-05
#991 Loss: 1.7282789485761896e-05
#992 Loss: 1.7257185390917584e-05
#993 Loss: 1.7231439414899796e-05
#994 Loss: 1.720591535558924e-05
#995 Loss: 1.718040948617272e-05
#996 Loss: 1.7155089153675362e-05
#997 Loss: 1.7129723346442915e-05
#998 Loss: 1.71041683643125e-05
#999 Loss: 1.707904812064953e-05
Predict data based on trained weights:
Input:
tensor([0.3750, 1.0000, 0.5000, 0.6250])
Output:
tensor([0.5798])
```

c) Tăng kích thước của các lớp trong mô hình

Chỉ thay đổi kích thước số lớp (input, hidden,..), giữ nguyên tốc độ học 0.1

```python
# import PyTorch
import torch
# import Feed Forward Neural Network class from nn_simple module
from ffnn import *

# sample input and output value for training
X = torch.tensor(([2, 9, 0, 7], [1, 5, 1, 8], [3, 6, 2, 1], [3, 1, 2, 9]), dtype=torch.float)  # 4 X 4 tensor
y = torch.tensor(([90], [100], [88], [120]), dtype=torch.float)  # 4 X 1 tensor

# scale units by max value
X_max, _ = torch.max(X, 0)
X = torch.div(X, X_max)
y = y / 120  # for max test score is 100

# sample input x for predicting
x_predict = torch.tensor(([3, 8, 4, 5]), dtype=torch.float)  # 1 X 4 tensor
```

```python
# Feed Forward Neural Network class
class FFNeuralNetwork(nn.Module):
    # initialization function
    def __init__(self, ):
        # init function of base class
        super(FFNeuralNetwork, self).__init__()

        # corresponding size of each layer
        self.inputSize = 4
        self.hiddenSize = 5
        self.outputSize = 1

        # random weights from a normal distribution
        self.W1 = torch.randn(self.inputSize, self.hiddenSize)  # 4 X 5 tensor
        self.W2 = torch.randn(self.hiddenSize, self.outputSize)  # 5 X 1 tensor
```

```
        print("#" + str(i) + " Loss: " + str(torch.mean((y - NN(X)) ** 2).detach().item()))
        # training with learning rate = 0.1
        NN.train(X, y, 0.1)
    # save weights
```

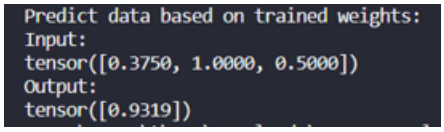
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

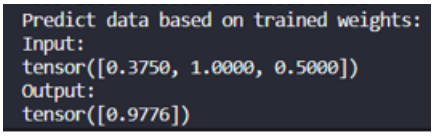
```
#973 Loss: 0.007548005320131779
#974 Loss: 0.00754585769027471 5
#975 Loss: 0.007543710060417652
#976 Loss: 0.007541564293205738
#977 Loss: 0.00753942271694541
#978 Loss: 0.007537278346717358
#979 Loss: 0.007535127457231283
#980 Loss: 0.007532985880970955
#981 Loss: 0.007530842907726765
#982 Loss: 0.007528700400143862
#983 Loss: 0.007526554632931948
#984 Loss: 0.007524404674768448
#985 Loss: 0.0075222644954919815
#986 Loss: 0.007520121522247791
#987 Loss: 0.0075179776176810265
#988 Loss: 0.0075158323161304
#989 Loss: 0.00751368748024106
#990 Loss: 0.007511541247367859
#991 Loss: 0.007509398274123669
#992 Loss: 0.007507253438234329
#993 Loss: 0.0075051141902804375
#994 Loss: 0.007502962835133076
#995 Loss: 0.007500816136598587
#996 Loss: 0.007498676888644695
#997 Loss: 0.007496532052755356
#998 Loss: 0.007494388148188591
#999 Loss: 0.00749224005267024
Predict data based on trained weights:
Input:
tensor([0.3750, 1.0000, 0.5000, 0.6250])
Output:
tensor([0.7671])
```

d) Nhận xét

Việc thay đổi tốc độ học sẽ khiến cho mô hình học có sự thay đổi rõ rệt:

- Nếu tốc độ học càng nhỏ thì mô hình sẽ không được học sâu vì lượng kiến thức học được thì không nhiều (tốc độ học nhỏ), học sâu hơn sẽ đem lại kết quả mà mô hình trả về có độ chính xác giảm đáng kể.
- Ngược lại, nếu mô hình có tốc độ học lớn, mô hình sẽ được học nhiều lượng kiến thức hơn, sẽ được học sâu hơn , kết quả của mô hình trả về sẽ được chính xác hơn đối với mô hình có tốc độ học nhỏ

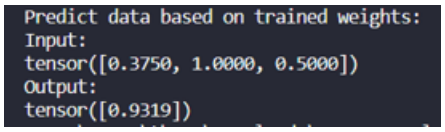


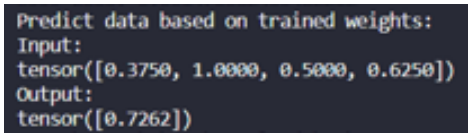


Có thể dễ dàng nhận thấy sự thay đổi ở output ở 2 mô hình có tốc độ học khác nhau. Đối với mô hình (hình trái) có tốc độ học nhỏ hơn so với mô hình (hình phải) có tốc độ học lớn hơn

Việc tăng kích thước các lớp trong mô hình cũng dẫn đến việc trả về kết quả có sự thay đổi:

- Nếu kích thước các lớp càng lớn thì kết quả mà mô hình trả về cũng có độ chính xác giảm
- Ngược lại, nếu kích thước càng nhỏ thì kết quả mà mô hình sẽ có độ chính xác cao hơn





Cùng một tốc độ học, nhưng ở mô hình (hình trái) có kích thước lớp nhỏ trả về output có sự chính xác hơn so với mô hình (hình phải) có kích thước lớp lớn thì kết quả trả về có sự chính xác không bằng.

II. Thực nghiệm Lab03

a) Tăng tốc độ học của mô hình

Tăng tốc độ học lần lượt ở các mức 0.5, 5, 15 (vẫn giữ kích thước số lớp input, hidden)

```
18
19  # train your network
20  net.use(Loss.mse, LossPrime.mse_prime)
21  net.fit(x_train, y_train, epochs=100, alpha=0.5)
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
On epoch 51 an average error = tensor(0.2350)
On epoch 52 an average error = tensor(0.2350)
On epoch 53 an average error = tensor(0.2349)
On epoch 54 an average error = tensor(0.2348)
On epoch 55 an average error = tensor(0.2348)
On epoch 56 an average error = tensor(0.2347)
On epoch 57 an average error = tensor(0.2346)
On epoch 58 an average error = tensor(0.2346)
On epoch 59 an average error = tensor(0.2345)
On epoch 60 an average error = tensor(0.2344)
On epoch 61 an average error = tensor(0.2344)
On epoch 62 an average error = tensor(0.2343)
On epoch 63 an average error = tensor(0.2343)
On epoch 64 an average error = tensor(0.2342)
On epoch 65 an average error = tensor(0.2342)
On epoch 66 an average error = tensor(0.2341)
On epoch 67 an average error = tensor(0.2341)
On epoch 68 an average error = tensor(0.2340)
On epoch 69 an average error = tensor(0.2340)
On epoch 70 an average error = tensor(0.2339)
On epoch 71 an average error = tensor(0.2339)
On epoch 91 an average error = tensor(0.2331)
On epoch 92 an average error = tensor(0.2331)
On epoch 93 an average error = tensor(0.2330)
On epoch 94 an average error = tensor(0.2330)
On epoch 95 an average error = tensor(0.2330)
On epoch 96 an average error = tensor(0.2330)
On epoch 97 an average error = tensor(0.2329)
On epoch 98 an average error = tensor(0.2329)
On epoch 99 an average error = tensor(0.2329)
On epoch 100 an average error = tensor(0.2328)
[tensor([[0.0112]]), tensor([[0.7458]]), tensor([[0.7189]]), tensor([[0.7660]])]
```

```
18
19  # train your network
20  net.use(Loss.mse, LossPrime.mse_prime)
21  net.fit(x_train, y_train, epochs=100, alpha=5)
22
23  # test
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
On epoch 64 an average error = tensor(0.5000)
On epoch 65 an average error = tensor(0.5000)
On epoch 66 an average error = tensor(0.5000)
On epoch 67 an average error = tensor(0.5000)
On epoch 68 an average error = tensor(0.5000)
On epoch 69 an average error = tensor(0.5000)
On epoch 70 an average error = tensor(0.5000)
On epoch 71 an average error = tensor(0.5000)
On epoch 78 an average error = tensor(0.5000)
On epoch 79 an average error = tensor(0.5000)
On epoch 80 an average error = tensor(0.5000)
On epoch 81 an average error = tensor(0.5000)
On epoch 82 an average error = tensor(0.5000)
On epoch 83 an average error = tensor(0.5000)
On epoch 84 an average error = tensor(0.5000)
On epoch 85 an average error = tensor(0.5000)
On epoch 86 an average error = tensor(0.5000)
On epoch 87 an average error = tensor(0.5000)
On epoch 88 an average error = tensor(0.5000)
On epoch 89 an average error = tensor(0.5000)
On epoch 90 an average error = tensor(0.5000)
On epoch 91 an average error = tensor(0.5000)
On epoch 92 an average error = tensor(0.5000)
On epoch 93 an average error = tensor(0.5000)
On epoch 94 an average error = tensor(0.5000)
On epoch 95 an average error = tensor(0.5000)
On epoch 96 an average error = tensor(0.5000)
On epoch 97 an average error = tensor(0.5000)
On epoch 98 an average error = tensor(0.5000)
On epoch 99 an average error = tensor(0.5000)
On epoch 100 an average error = tensor(0.5000)
[tensor([[1.0000]]), tensor([[1.0000]]), tensor([[1.0000]]), tensor([[1.0000]])]
```

```
18
19    # train your network
20    net.use(Loss.mse, LossPrime.mse_prime)
21    net.fit(x_train, y_train, epochs=100, alpha=15)
22
23    # test
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
On epoch 70 an average error = tensor(0.5000)
On epoch 71 an average error = tensor(0.5000)
On epoch 72 an average error = tensor(0.5000)
On epoch 73 an average error = tensor(0.5000)
On epoch 74 an average error = tensor(0.5000)
On epoch 75 an average error = tensor(0.5000)
On epoch 76 an average error = tensor(0.5000)
On epoch 77 an average error = tensor(0.5000)
On epoch 78 an average error = tensor(0.5000)
On epoch 79 an average error = tensor(0.5000)
On epoch 80 an average error = tensor(0.5000)
On epoch 81 an average error = tensor(0.5000)
On epoch 82 an average error = tensor(0.5000)
On epoch 83 an average error = tensor(0.5000)
On epoch 84 an average error = tensor(0.5000)
On epoch 85 an average error = tensor(0.5000)
On epoch 86 an average error = tensor(0.5000)
On epoch 87 an average error = tensor(0.5000)
On epoch 88 an average error = tensor(0.5000)
On epoch 89 an average error = tensor(0.5000)
On epoch 90 an average error = tensor(0.5000)
On epoch 91 an average error = tensor(0.5000)
On epoch 92 an average error = tensor(0.5000)
On epoch 93 an average error = tensor(0.5000)
On epoch 94 an average error = tensor(0.5000)
On epoch 95 an average error = tensor(0.5000)
On epoch 96 an average error = tensor(0.5000)
On epoch 97 an average error = tensor(0.5000)
On epoch 98 an average error = tensor(0.5000)
On epoch 99 an average error = tensor(0.5000)
On epoch 100 an average error = tensor(0.5000)
[tensor([[1.]]), tensor([[1.]]), tensor([[1.]]), tensor([[1.]])]
```

b) Tăng tốc độ học kết hợp với thay đổi kích thước các lớp trong mô hình

Tăng tốc độ học lần lượt ở các mức 0.5, 5, 15 (thay đổi kích thước số lớp input, hidden, số epoch,..)

```
import torch

from layer_simple import FCLayer, ActivationLayer
from function_simple import Activation, ActivationPrime
from function_simple import Loss, LossPrime
from network_simple import Network

# training data
x_train = torch.tensor([[[0, 0, 1, 0]], [[0, 1, 0, 1]], [[1, 0, 1, 1]], [[1, 1, 0, 0]], [[1, 1, 1, 1]]], dtype=torch.float)
y_train = torch.tensor([[[0]], [[1]], [[1]], [[0]], [[2]]], dtype=torch.float)

# network architecture
net = Network()
net.add(FCLayer(4, 5))
net.add(ActivationLayer(Activation.tanh, ActivationPrime.tanh_derivative))
net.add(FCLayer(5, 1))
net.add(ActivationLayer(Activation.tanh, ActivationPrime.tanh_derivative))
```

```
# train your network
net.use(Loss.mse, LossPrime.mse_prime)
net.fit(x_train, y_train, epochs=150, alpha=0.5)

# test
out = net.predicts(x_train)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
On epoch 108 an average error = tensor(0.5999)
On epoch 109 an average error = tensor(0.5999)
On epoch 110 an average error = tensor(0.5999)
On epoch 111 an average error = tensor(0.5999)
On epoch 112 an average error = tensor(0.5999)
On epoch 113 an average error = tensor(0.5999)
On epoch 114 an average error = tensor(0.5999)
On epoch 115 an average error = tensor(0.5999)
On epoch 116 an average error = tensor(0.5999)
On epoch 117 an average error = tensor(0.5999)
On epoch 118 an average error = tensor(0.5999)
On epoch 119 an average error = tensor(0.5999)
On epoch 120 an average error = tensor(0.5999)
On epoch 121 an average error = tensor(0.5999)
On epoch 122 an average error = tensor(0.5999)
On epoch 123 an average error = tensor(0.5999)
On epoch 124 an average error = tensor(0.5999)
On epoch 125 an average error = tensor(0.5999)
On epoch 126 an average error = tensor(0.5999)
On epoch 127 an average error = tensor(0.5999)
On epoch 128 an average error = tensor(0.5999)
On epoch 129 an average error = tensor(0.5999)
On epoch 144 an average error = tensor(0.5999)
On epoch 145 an average error = tensor(0.5999)
On epoch 146 an average error = tensor(0.5999)
On epoch 147 an average error = tensor(0.5999)
On epoch 148 an average error = tensor(0.5999)
On epoch 149 an average error = tensor(0.5999)
On epoch 150 an average error = tensor(0.5999)
[tensor([[0.9998]]), tensor([[1.0000]]), tensor([[0.9997]]), tensor([[0.9999]]), tensor([[1.0000]])]
```

```
19  # train your network
20  net.use(Loss.mse, LossPrime.mse_prime)
21  net.fit(x_train, y_train, epochs=150, alpha=5)
22
23  # test
24  out = net.predicts(x_train)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
On epoch 122 an average error = tensor(3.8000)
On epoch 123 an average error = tensor(3.8000)
On epoch 124 an average error = tensor(3.8000)
On epoch 125 an average error = tensor(3.8000)
On epoch 126 an average error = tensor(3.8000)
On epoch 127 an average error = tensor(3.8000)
On epoch 128 an average error = tensor(3.8000)
On epoch 129 an average error = tensor(3.8000)
On epoch 130 an average error = tensor(3.8000)
On epoch 131 an average error = tensor(3.8000)
On epoch 132 an average error = tensor(3.8000)
On epoch 133 an average error = tensor(3.8000)
On epoch 134 an average error = tensor(3.8000)
On epoch 135 an average error = tensor(3.8000)
On epoch 136 an average error = tensor(3.8000)
On epoch 137 an average error = tensor(3.8000)
On epoch 138 an average error = tensor(3.8000)
On epoch 139 an average error = tensor(3.8000)
On epoch 140 an average error = tensor(3.8000)
On epoch 141 an average error = tensor(3.8000)
On epoch 142 an average error = tensor(3.8000)
On epoch 143 an average error = tensor(3.8000)
On epoch 144 an average error = tensor(3.8000)
On epoch 145 an average error = tensor(3.8000)
On epoch 146 an average error = tensor(3.8000)
On epoch 147 an average error = tensor(3.8000)
On epoch 148 an average error = tensor(3.8000)
On epoch 149 an average error = tensor(3.8000)
On epoch 150 an average error = tensor(3.8000)
[tensor([[-1.]]), tensor([[-1.]]), tensor([[-1.]]), tensor([[-1.]]), tensor([[-1.]])]
```

```
19    # train your network
20    net.use(Loss.mse, LossPrime.mse_prime)
21    net.fit(x_train, y_train, epochs=150, alpha=15)
22
23    # test
24    out = net.predicts(x_train)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
On epoch 122 an average error = tensor(0.6000)
On epoch 123 an average error = tensor(0.6000)
On epoch 124 an average error = tensor(0.6000)
On epoch 125 an average error = tensor(0.6000)
On epoch 126 an average error = tensor(0.6000)
On epoch 127 an average error = tensor(0.6000)
On epoch 128 an average error = tensor(0.6000)
On epoch 129 an average error = tensor(0.6000)
On epoch 130 an average error = tensor(0.6000)
On epoch 131 an average error = tensor(0.6000)
On epoch 132 an average error = tensor(0.6000)
On epoch 133 an average error = tensor(0.6000)
On epoch 134 an average error = tensor(0.6000)
On epoch 135 an average error = tensor(0.6000)
On epoch 136 an average error = tensor(0.6000)
On epoch 137 an average error = tensor(0.6000)
On epoch 138 an average error = tensor(0.6000)
On epoch 139 an average error = tensor(0.6000)
On epoch 140 an average error = tensor(0.6000)
On epoch 141 an average error = tensor(0.6000)
On epoch 142 an average error = tensor(0.6000)
On epoch 143 an average error = tensor(0.6000)
On epoch 144 an average error = tensor(0.6000)
On epoch 145 an average error = tensor(0.6000)
On epoch 146 an average error = tensor(0.6000)
On epoch 147 an average error = tensor(0.6000)
On epoch 148 an average error = tensor(0.6000)
On epoch 149 an average error = tensor(0.6000)
On epoch 150 an average error = tensor(0.6000)
[tensor([[1.]]), tensor([[1.]]), tensor([[1.]]), tensor([[1.]]), tensor([[1.]])]
```

c) Tăng kích thước của các lớp trong mô hình

Chỉ thay đổi kích thước số lớp (input, hidden, số epoch,..), giữ nguyên tốc độ học 0.1

```
import torch

from layer_simple import FCLayer, ActivationLayer
from function_simple import Activation, ActivationPrime
from function_simple import Loss, LossPrime
from network_simple import Network

# training data
x_train = torch.tensor([[[0, 0, 1, 0]], [[0, 1, 0, 1]], [[1, 0, 1, 1]], [[1, 1, 0, 0]], [[1, 1, 1, 1]]], dtype=torch.float)
y_train = torch.tensor([[[0]], [[1]], [[1]], [[0]], [[2]]], dtype=torch.float)

# network architecture
net = Network()
net.add(FCLayer(4, 5))
net.add(ActivationLayer(Activation.tanh, ActivationPrime.tanh_derivative))
net.add(FCLayer(5, 1))
net.add(ActivationLayer(Activation.tanh, ActivationPrime.tanh_derivative))
```

```
19  # train your network
20  net.use(Loss.mse, LossPrime.mse_prime)
21  net.fit(x_train, y_train, epochs=150, alpha=0.1)
22
23  # test
24  out = net.predicts(x_train)
```

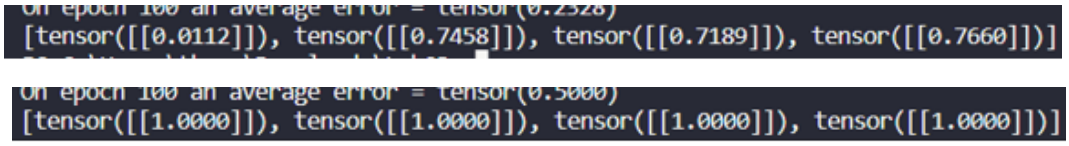
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
On epoch 122 an average error = tensor(0.2017)
On epoch 123 an average error = tensor(0.2016)
On epoch 124 an average error = tensor(0.2016)
On epoch 125 an average error = tensor(0.2016)
On epoch 126 an average error = tensor(0.2016)
On epoch 127 an average error = tensor(0.2016)
On epoch 128 an average error = tensor(0.2016)
On epoch 129 an average error = tensor(0.2016)
On epoch 130 an average error = tensor(0.2016)
On epoch 131 an average error = tensor(0.2015)
On epoch 132 an average error = tensor(0.2015)
On epoch 133 an average error = tensor(0.2015)
On epoch 134 an average error = tensor(0.2015)
On epoch 135 an average error = tensor(0.2015)
On epoch 136 an average error = tensor(0.2015)
On epoch 137 an average error = tensor(0.2015)
On epoch 138 an average error = tensor(0.2015)
On epoch 139 an average error = tensor(0.2015)
On epoch 140 an average error = tensor(0.2014)
On epoch 141 an average error = tensor(0.2014)
On epoch 142 an average error = tensor(0.2014)
On epoch 143 an average error = tensor(0.2014)
On epoch 144 an average error = tensor(0.2014)
On epoch 145 an average error = tensor(0.2014)
On epoch 146 an average error = tensor(0.2014)
On epoch 147 an average error = tensor(0.2014)
On epoch 148 an average error = tensor(0.2014)
On epoch 149 an average error = tensor(0.2014)
On epoch 150 an average error = tensor(0.2013)
[tensor([[0.0045]]), tensor([[0.9982]]), tensor([[0.9962]]), tensor([[0.0032]]), tensor([[0.9967]])]
```

d) Nhận xét

Việc thay đổi tốc độ học sẽ khiến cho mô hình học có sự thay đổi rõ rệt:

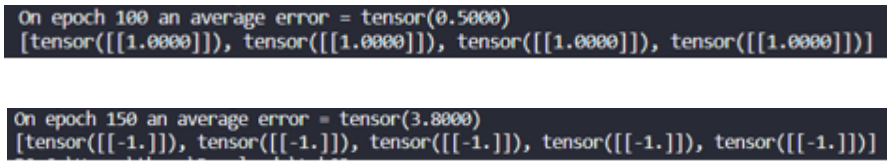
- Nếu tốc độ học càng nhỏ thì mô hình sẽ không được học sâu vì lượng kiến thức học được thì không nhiều (tốc độ học nhỏ), học sâu hơn sẽ đem lại kết quả mà mô hình trả về có độ chính xác giảm đáng kể.
- Ngược lại, nếu mô hình có tốc độ học lớn, mô hình sẽ được học nhiều lượng kiến thức hơn, sẽ được học sâu hơn , kết quả của mô hình trả về sẽ được chính xác hơn đối với mô hình có tốc độ học nhỏ



Có thể dễ dàng nhận thấy sự thay đổi ở output ở 2 mô hình có tốc độ học khác nhau. Đối với mô hình (hình trên) có tốc độ học nhỏ thì độ chính xác thấp hơn so với mô hình (hình dưới) có tốc độ học lớn hơn.

Việc tăng kích thước các lớp trong mô hình cũng dẫn đến việc trả về kết quả có sự thay đổi:

- Nếu kích thước các lớp càng lớn thì kết quả mà mô hình trả về cũng có độ chính xác giảm
- Ngược lại, nếu kích thước càng nhỏ thì kết quả mà mô hình sẽ có độ chính xác cao hơn



Cùng một tốc độ học đã có sự thay đổi rõ rệt ở kết quả trả về ở 2 mô hình, ở mô hình (hình trên) có kích thước lớp nhỏ trả về output có sự chính xác hơn so với mô hình (hình dưới) có kích thước lớp lớn thì kết quả trả về có sự chính xác không bằng.

TÀI LIỆU THAM KHẢO

Danh mục tài liệu tham khảo:

[1] *https://pytorch.org/*
[2]*https://viblo.asia/p/thiet-ke-mang-quantum-neural-network-voi-pytorch-va-qiskit-m2vJPwna4eK*
[3] *https://phamdinhkhanh.github.io/2019/08/10/PytorchTurtorial1.html*