Deep Learning

Assignment#3 - Shallow Neural Networks for XOR

2019028313 - Kim Hyewon

SNN의 구조

- 1 hidden layer: activation function – tanh

  [tanh] unlinear한 활성화 함수이다. tanh를 hidden layer의 활성화 함수로 사용함으로써 비선형적인 학습을 가능하게 하며 선형적으로는 학습할 수 없었던 XOR을 학습시킨다.

- ouput layer: activation function - sigmoid

Source code

1. init

```python
import numpy as np
import matplotlib.pyplot as plt
```

✓ 0.3s Python

```python
# Data preparation
x_seeds = np.array([(0, 0),(1, 0),(0, 1),(1, 1)], dtype = np.float64)
y_seeds = np.array([0, 1, 1, 0]) # XOR

# data를 random하게 준비한다.
N = 1000
idxs = np.random.randint(0,4,N) # 0~3사이의 수 N개 생성 -> index가 된다.

X = x_seeds[idxs]
Y = y_seeds[idxs]

X += np.random.normal(scale = 0.25, size = X.shape)
```

✓ 0.3s Python

2. model

```python
#  Model
class shallow_neural_network():
    def __init__(self, num_input_features, num_hiddens):
        self.num_input_features = num_input_features
        self.num_hiddens = num_hiddens # hidden layer의 뉴런 수

        # random하게 model parameter Initialize
        self.W1 = np.random.normal(size=(num_hiddens, num_input_features))
        self.b1 = np.random.normal(size=num_hiddens)
        self.W2 = np.random.normal(size=num_hiddens)
        self.b2 = np.random.normal(size=1)

    def sigmoid(self,x):
        return 1/(1 + np.exp(-x))

    def predict(self, x):
        z1 = np.matmul(self.W1, x) + self.b1
        a1 = np.tanh(z1)

        z2 = np.matmul(self.W2, a1) + self.b2
        a2 = self.sigmoid(z2)
        return a2, (z1,a1,z2,a2)

```

✓ 0.3s Python

3. train

```python
def train(X, Y, model, lr = 0.1):
    dW1 = np.zeros_like(model.W1)
    db1 = np.zeros_like(model.b1)
    dW2 = np.zeros_like(model.W2)
    db2 = np.zeros_like(model.b2)
    m = len(X)
    cost = 0.0
    for x,y in zip(X,Y):
        a2, (z1,a1, z2, _) = model.predict(x)
        if y == 1:
            cost -= np.log(a2)
        else:
            cost -= np.log(1-a2)

        diff = a2-y
        # layer 2
        # db2
        db2 += diff
        # dW2
        dW2 += diff * a1

        # layer1
        # db1
        db1 += (1-a1**2)*model.W2*diff
        #db2
        dW1 += np.outer((1-a1**2)*model.W2,x) * diff

    cost /= m
    model.W1 -= lr * dW1/m
    model.b1 -= lr * db1/m
    model.W2 -= lr * dW2/m
    model.b2 -= lr * db2/m
    return cost
```

4. training

```python
lr_list = [1.0,0.5,0.3,0.1]
n = 1000

for lr in lr_list:
    # print("learning rate : ", lr)
    model = shallow_neural_network(2,3)
    list = []
    for epoch in range(n):
        cost = train(X,Y, model, lr)
        if epoch % 10 ==0:
            list.append(cost)
    plt.figure(figsize=(4,4))
    plt.title("XOR - lr: "+str(lr))
    plt.plot(range(0,int(n/10)),list)
    print(list[-1])
plt.show()
```

✓ 1m 51.7s Python

```python
lr_list = [0.1]
n = 20000
modelToTest = []

for lr in lr_list:
    # print("learning rate : ", lr)
    model = shallow_neural_network(2,3)
    list = []
    for epoch in range(n):
        cost = train(X,Y, model, lr)
        if epoch % 10 ==0:
            list.append(cost)
    plt.figure(figsize=(4,4))
    plt.title("XOR - lr: "+str(lr))
    plt.plot(range(0,int(n/10)),list)
    print(list[-1])
    modelToTest = model
plt.show()
```

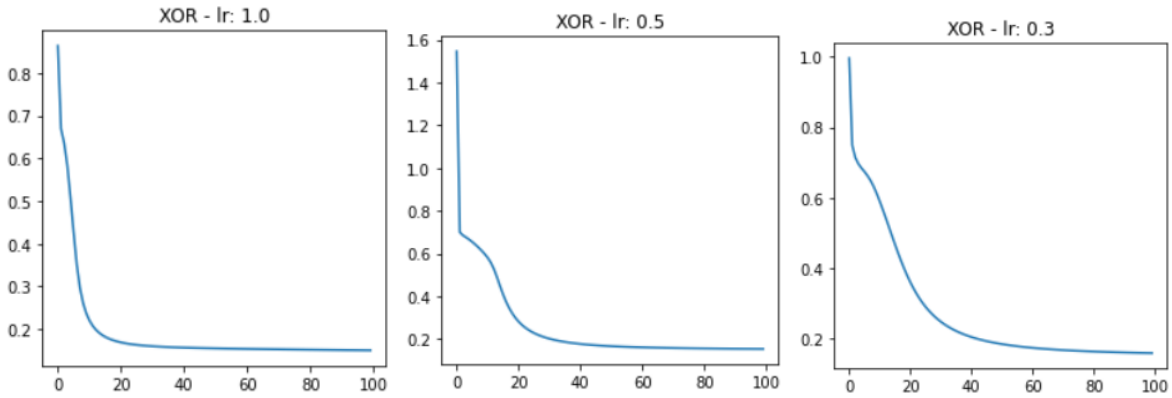
```python
lr_list = [0.05, 0.01]
n = 30000

for lr in lr_list:
    # print("learning rate : ", lr)
    model = shallow_neural_network(2,3)
    list = []
    for epoch in range(n):
        cost = train(X,Y, model, lr)
        if epoch % 10 ==0:
            list.append(cost)
    plt.figure(figsize=(4,4))
    plt.title("XOR - lr: "+str(lr))
    plt.plot(range(0,int(n/10)),list)
    print(list[-1])
plt.show()
```

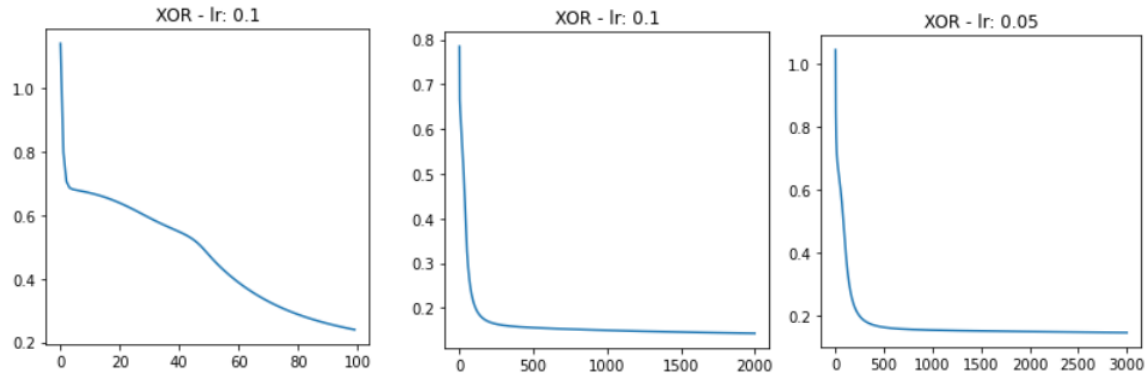
✓ 27m 52.7s

Result

- learning rate: 1.0, 0.5, 0.3 / epoch: 1000



- learning rate: 0.1 / epoch: 1000, 20000 , lr: 0.05 / epoch: 30000



설계한 SNN으로 XOR을 학습한 결과 learning rate: 0.1, epoch: 20000으로 학습시킨 결과 가장 성능이 좋은 모델을 얻을 수 있었습니다. 해당 모델을 통해 test한 결과 다음과 같은 결과를 확인할 수 있었습니다.

```python
def model_test(model):
    for x in x_seeds:
        print(x, model.predict(x)[0].item())
```

0.3s Python

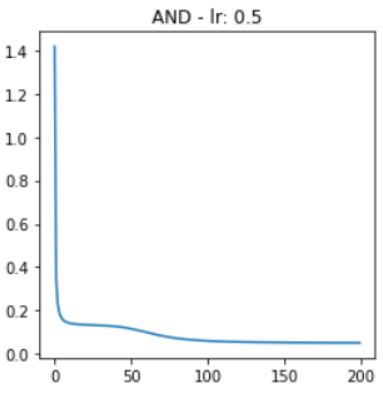
```python
model_test(modelToTest)
```

0.3s Python

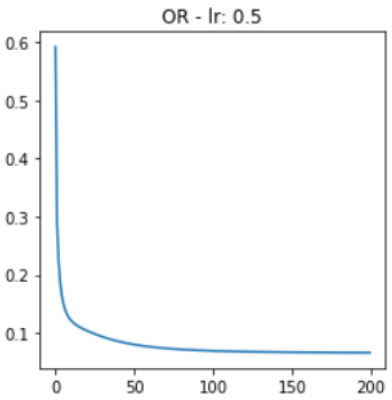
```
[0. 0.] 0.011366569725627381
[1. 0.] 0.9792825469604189
[0. 1.] 0.9980199121721411
[1. 1.] 0.0015760559541431148
```

입력에 따른 정답과 매우 유사한 값을 반환하는 것을 확인할 수 있습니다.

SNN으로 linear한 AND gate, OR gate를 학습시킨 결과 아래와 같이 학습되는 것 또한 확인할 수 있었습니다.



[AND]



[OR]