Lab 6. Pattern Recognition

Yongjae Yoo, Ph. D.
Assistant Professor
Department of Artificial Intelligence
Hanyang University ERICA

So Far,

- We understood neural network
- Today, we’re going to learn how deep learning models find so phisticated patterns.

ABC Classifier

- Here we assume that we have three images, 6 by 5, consists o f A, B, and C.
- Assume that the letter is on-off (binary) data. That is:

```
#A
0 0 1 1 0 0
0 1 0 0 1 0
1 1 1 1 1 1
1 0 0 0 0 1
1 0 0 0 0 1
```

```
#B
0 1 1 1 1 0
0 1 0 0 1 0
0 1 1 1 1 0
0 1 0 0 1 0
0 1 1 1 1 0
```

```
#C
0 1 1 1 1 0
0 1 0 0 0 0
0 1 0 0 0 0
0 1 0 0 0 0
0 1 1 1 1 0
```

- We will set a label for classify multiple characters here:

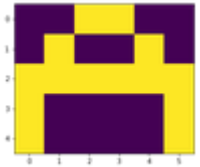
```
#Labels for each Letter
A=[1, 0, 0]
B=[0, 1, 0]
C=[0, 0, 1]
```

Make it sure it is A, B, and C

- Python code to visualize

```
import numpy as np
import matplotlib.pyplot as plt
# visualizing the data, plotting A.
plt.imshow(np.array(a).reshape(5, 6))
plt.show()
```

- Check it out for B, and C too.



Converting The Patterns to One-Hot-Vector

- Here we change the data 6x5 to 1x30

```
x =[np.array(a).reshape(1, 30),
    np.array(b).reshape(1, 30),
    np.array(c).reshape(1, 30)]


# Labels are also converted into NumPy array
y = np.array(y)

print(x, "\n\n", y)
```

- The output would be:

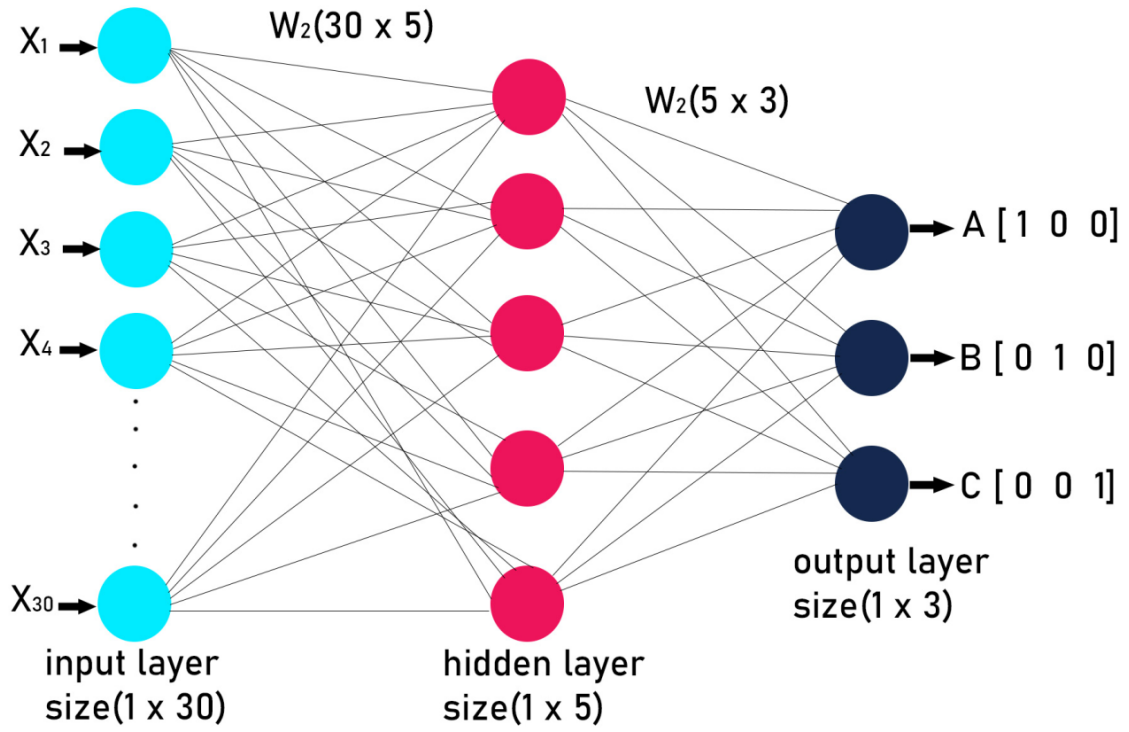
```
[array([[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1]]),
array([[0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0]]),
array([[0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0]])]



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

Setting Up Deep Learning

- Now we have 1x30 vector as an input (30 input data)
- Using a hidden layer of 1x5, we will build an ABC classifier.



```
# activation function
def sigmoid(x):
    return(1/(1 + np.exp(-x)))

# Creating the Feed forward neural network
# 1 Input layer(1, 30), # 1 hidden layer (1, 5), # 1 output layer(3, 3)

def f_forward(x, w1, w2):
    # hidden
    z1 = x.dot(w1)# input from layer 1
    a1 = sigmoid(z1)# out put of layer 2

    # Output layer
    z2 = a1.dot(w2)# input of out layer
    a2 = sigmoid(z2)# output of out layer
    return(a2)
```

```
# initializing the weights randomly
def generate_wt(x, y):
    l =[]
    for i in range(x * y):
        l.append(np.random.randn())
    return(np.array(l).reshape(x, y))

# for loss we will be using mean square error(MSE)
def loss(out, Y):
    s =(np.square(out-Y))
    s = np.sum(s)/len(y)
    return(s)
```

```
# Back propagation of error
def back_prop(x, y, w1, w2, alpha):
    # hidden layer
    z1 = x.dot(w1)# input from layer 1
    a1 = sigmoid(z1)# output of layer 2
    # Output layer
    z2 = a1.dot(w2)# input of out layer
    a2 = sigmoid(z2)# output of out layer
    # error in output layer
    d2 =(a2-y)
    d1 = np.multiply((w2.dot((d2.transpose()))).transpose(), (np.multiply(a1, 1-a1)))
    # Gradient for w1 and w2
    w1_adj = x.transpose().dot(d1)
    w2_adj = a1.transpose().dot(d2)
    # Updating parameters
    w1 = w1-(alpha*(w1_adj))
    w2 = w2-(alpha*(w2_adj))
    return(w1, w2)
```

```
def train(x, Y, w1, w2, alpha = 0.01, epoch = 10):
    acc =[]
    losss =[]
    for j in range(epoch):
        l =[]
        for i in range(len(x)):
            out = f_forward(x[i], w1, w2)
            l.append((loss(out, Y[i])))
            w1, w2 = back_prop(x[i], y[i], w1, w2, alpha)
        print("epochs:", j + 1, "======== acc:", (1-(sum(l)/len(x)))*100)
        acc.append((1-(sum(l)/len(x)))*100)
        losss.append(sum(l)/len(x))
    return(acc, losss, w1, w2)
```

```
def predict(x, w1, w2):
    Out = f_forward(x, w1, w2)
    maxm = 0
    k = 0
    for i in range(len(Out[0])):
        if(maxm<Out[0][i]):
            maxm = Out[0][i]
            k = i
    if(k == 0):
        print("Image is of letter A.")
    elif(k == 1):
        print("Image is of letter B.")
    else:
        print("Image is of letter C.")
    plt.imshow(x.reshape(5, 6))
    plt.show()
```

Let’s Try this

- Code to setting up a NN:

```
w1 = generate_wt(30, 5)
w2 = generate_wt(5, 3)
print(w1, "\n\n", w2)
```

- The results shows the arbitrary w1 and w2… values

```
[[ 0.75696605 -0.15959223 -1.43034587 0.17885107 -0.75859483] [-0.22870119 1.05882236 -0.15880572 0.11692122 0.58621482]
[ 0.13926738 0.72963505 0.36050426 0.79866465 -0.17471235] [ 1.00708386 0.68803291 0.14110839 -0.7162728 0.69990794]
[-0.90437131 0.63977434 -0.43317212 0.67134205 -0.9316605 ] [ 0.15860963 -1.17967773 -0.70747245 0.22870289 0.00940404]
[ 1.40511247 -1.29543461 1.41613069 -0.97964787 -2.86220777] [ 0.66293564 -1.94013093 -0.78189238 1.44904122 -1.81131482]
[ 0.4441061 -0.18751726 -2.58252033 0.23076863 0.12182448] [-0.60061323 0.39855851 -0.55612255 2.0201934 0.70525187]
[-1.82925367 1.32004437 0.03226202 -0.79073523 -0.20750692] [-0.25756077 -1.37543232 -0.71369897 -0.13556156 -0.34918718]
[ 0.26048374 2.49871398 1.01139237 -1.73242425 -0.67235417] [ 0.30351062 -0.45425039 -0.84046541 -0.60435352 -0.06281934]
[ 0.43562048 0.66297676 1.76386981 -1.11794675 2.2012095 ] [-1.11051533 0.3462945 0.19136933 0.19717914 -1.78323674]
[ 1.1219638 -0.04282422 -0.0142484 -0.73210071 -0.58364205] [-1.24046375 0.23368434 0.62323707 -1.66265946 -0.87481714]
[ 0.19484897 0.12629217 -1.01575241 -0.47028007 -0.58278292] [ 0.16703418 -0.50993283 -0.90036661 2.33584006 0.96395524]
[-0.72714199 0.39000914 -1.3215123 0.92744032 -1.44239943] [-2.30234278 -0.52677889 -0.09759073 -0.63982215 -0.51416013]
[ 1.25338899 -0.58950956 -0.86009159 -0.7752274 2.24655146] [ 0.07553743 -1.2292084 0.46184872 -0.56390328 0.15901276]
[-0.52090565 -2.42754589 -0.78354152 -0.44405857 1.16228247] [-1.21805132 -0.40358444 -0.65942185 0.76753095 -0.19664978]
[-1.5866041 1.17100962 -1.50840821 -0.61750557 1.56003127] [ 1.33045269 -0.85811272 1.88869376 0.79491455 -0.96199293]
[-2.34456987 0.1005953 -0.99376025 -0.94402235 -0.3078695 ] [ 0.93611909 0.58522915 -0.15553566 -1.03352997 -2.7210093 ]]
[[-0.50650286 -0.41168428 -0.7107231]
[ 1.86861492 -0.36446849 0.97721539]
[-0.12792125 0.69578056 -0.6639736 ]
[ 0.58190462 -0.98941614 0.40932723]
[ 0.89758789 -0.49250365 -0.05023684]]
```

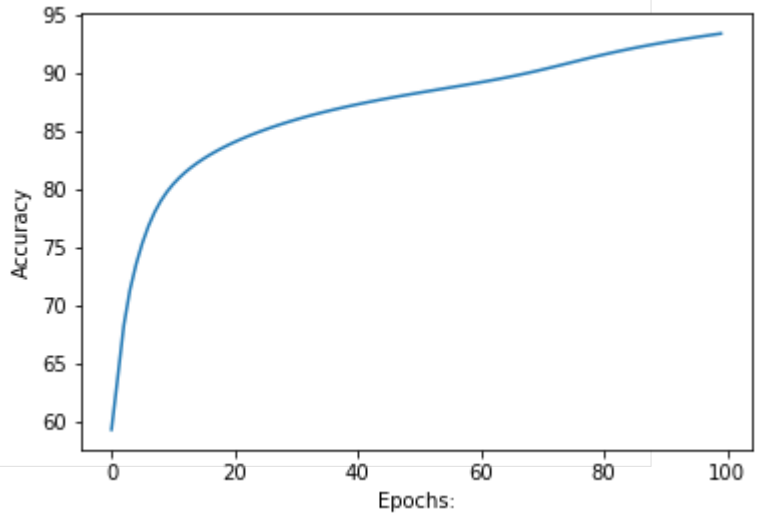
- Training the NN and plot the accuracy and loss

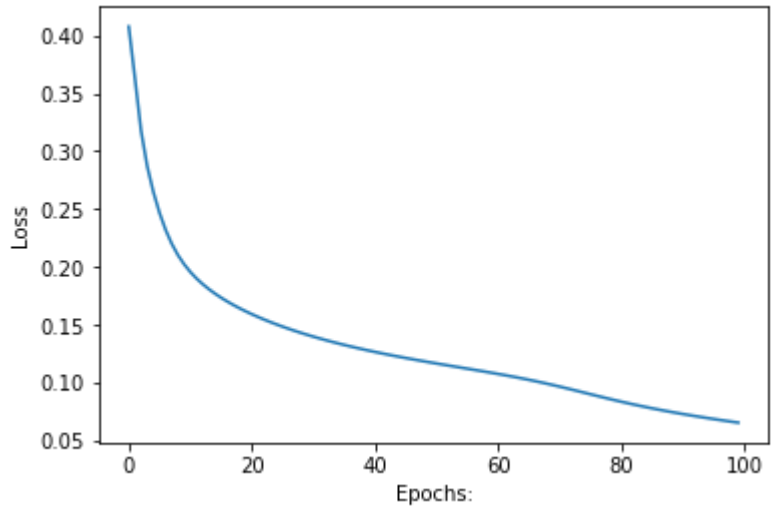
```
acc, losss, w1, w2 = train(x, y, w1, w2, 0.1, 100)
```

```
import matplotlib.pyplot as plt1

# plotting accuracy
plt1.plot(acc)
plt1.ylabel('Accuracy')
plt1.xlabel("Epochs:")
plt1.show()

# plotting Loss
plt1.plot(losss)
plt1.ylabel('Loss')
plt1.xlabel("Epochs:")
plt1.show()
```





- And trained weights:

```
print(w1, "\n", w2)
```

- Using this, we can predict an arbitrary pattern:

```
predict(x[1], w1, w2)
```

- (you may put in an arbitrary 1x30 data)

How about Build an ABCD classifier?

- Consider a new class D

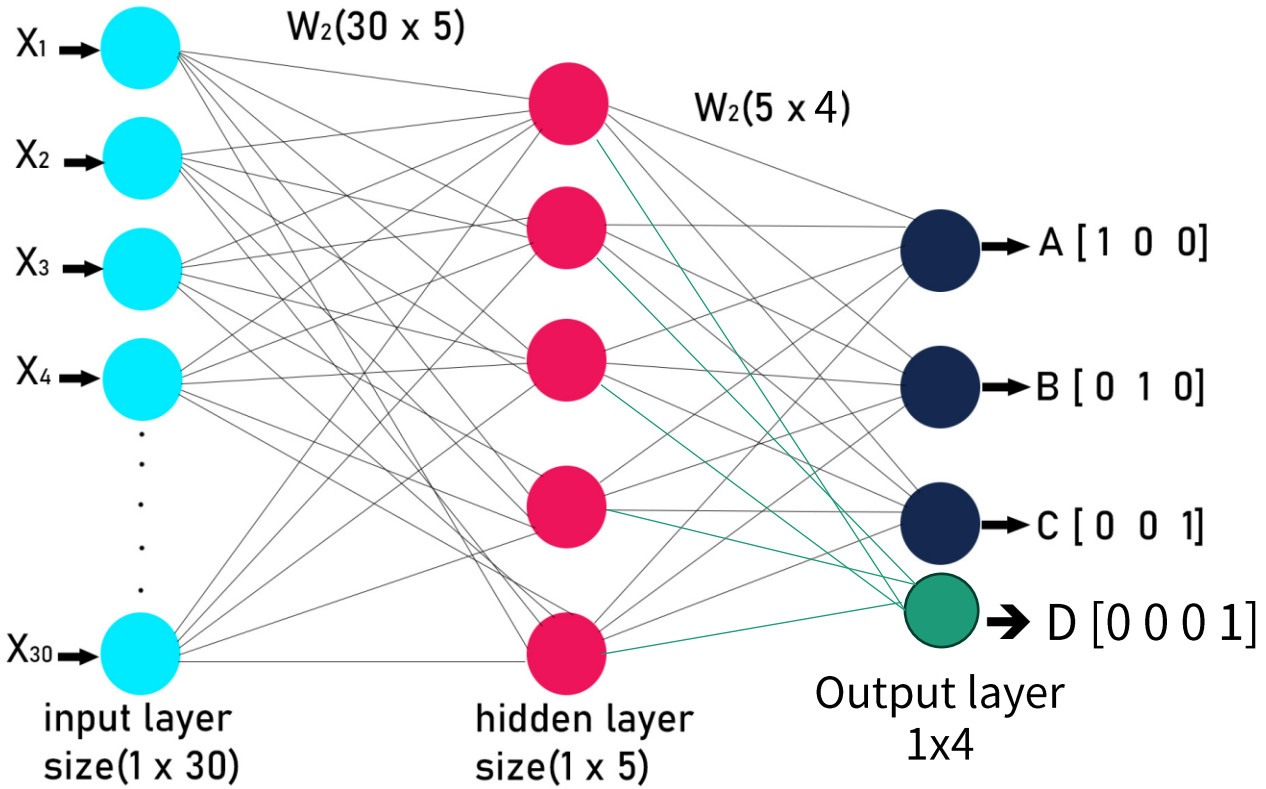
```
d =[0, 1, 1, 1, 0, 0,
   0, 1, 0, 0, 1, 0,
   0, 1, 0, 0, 1, 0,
   0, 1, 0, 0, 1, 0,
   0, 1, 1, 1, 0, 0]
```

```
x =[np.array(a).reshape(1, 30),
np.array(b).reshape(1, 30),
np.array(c).reshape(1, 30),
np.array(d).reshape(1,30)]
```

- Probably, the label should be 4x4

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

- We need Output layer of 1x4



- Setting up a new NN with 4 outputs.

```
w1 = generate_wt(30, 5)
w2 = generate_wt(5, 4)
print(w1, "\n\n", w2)
```

- You may also check predict part to consider
  - “This is the letter A, B, C, and D.” part

```
if(k == 0):
  print("Image is of letter A.")
elif(k == 1):
  print("Image is of letter B.")
elif(k == 2):
  print("Image is of letter C.")
else:
  print("Image is of letter D.")
```

- Run the same code and check out the results.
- Put in an E. What is the result?
  - ```
    e = np.array([0, 1, 1 ,1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
    1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0]).
    reshape(1, 30)
    ```
    ```

On Thursday and Friday Labs

- We will play with Tic-Tac-Toe and AI decisions