



دانشگاه تهران

دانشکده فنی

دپارتمان مهندسی برق و کامپیوتر

سیستم های هوشمند

پروژه نهایی درس

محمدحسین و عیدی

810197605

سیدمحمدمتین آل محمد

810197457

محمدحیدری

810197494

محمدرضاتوکلی

810197477

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF TEHRAN

بهمن 1400

1 فهرست مطالب

بخش اول تولید تصویر	3
بخش دوم آنالیز معنایی متن	30
بخش سوم شبکه پیچشی	45
نکات انتهایی	53
ارجاعات	54

بخش اول: تولید تصویر

در قسمت اول این پروژه قصد داریم که به بررسی شبکه های عصبی در حوزه پردازش تصویر بپردازیم.

ایده اصلی این کار مربوط به بکارگیری درست شبکه های مولد^۱ خواهد بود. لذا در ابتدا به توضیح این شبکه ها خواهیم پرداخت:

بطور کلی روش های تولید تصویر و اغلب کارهای پردازش تصویر با استفاده از شبکه های پیچشی^۲ صورت خواهند پذیرفت.

شبکه های مولد: مدل سازی مولد یک روش یادگیری بدون نظارت در یادگیری ماشین است که شامل کشف و یادگیری خودکار قوانین یا الگوهای موجود در داده های ورودی است به گونه ای که بتواند از مدل برای تولید خروجی یا نمونه های جدیدی استفاده کند که به طور قابل قبولی می توانند از مجموعه داده اصلی استخراج شده باشند.

مادر این پروژه قصد داریم تا با بکارگیری اصول شبکه های مولد یک شبکه عصبی پیچشی^۳ را بدون استفاده از کتابخانه و تنها با بکارگیری کتابخانه numpy پیاده سازی کنیم و در ادامه با استفاده از موارد مطرح شده در صورت سوال به تولید تصویر از کلاس های دلخواه بپردازیم.

* باتوجه به مکاتبه هایی که با دستیار آموزشی محترم این پروژه گردید بنامش که برای سادگی کار از دیتاست MNIST استفاده نماییم.

توضیحات دیتاست: این دیتاست شامل دست نوشته هایی از اعداد 0 تا 9 میباشد که در 10 کلاس مختلف ارایه شده است که در ادامه میتوانید کلاس های این دیتاست را مشاهده نمایید:



Figure1: MNIST دیتاست

¹ Generative

² convolutional

³ CNN

در ادامه قصد داریم تئوری مربوط به شبکه های CNN را بررسی کنیم چراکه این نوع از شبکه به عنوان هسته اصلی تولید تصویر در این سوال عمل میکند و طراحی دقیق و تعیین پارامترها نقش بسیار موثری در دقت نهایی طبقه بند ما خواهند داشت.

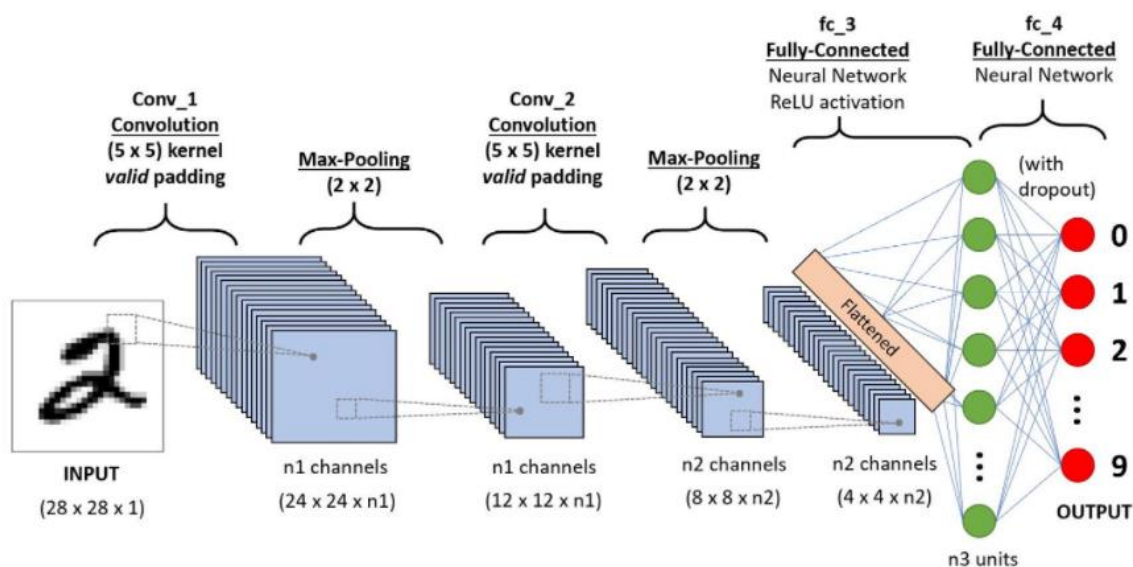


Figure2: ساختار معماری شبکه

همانطور که پایپ لاین بالا نشان میدهد شبکه CNN ما از 3 لایه عمده تشکیل شده است:

1. لایه های کانولوشنی³ بعدی برای استخراج ویژگی های تصویر ورودی
2. لایه های بیشینه ادغام⁴ برای کوچکتر کردن سایز تصاویر
3. لایه های تماماً متصل⁵ برای انجام فرآیند طبقه بندی

مراحل کار پایپ لاین نیز بدین صورت است که تصویر ورودی به شبکه اعمال میشود و در مرحله ی اول با استفاده از لایه های کانولوشنی و لایه های بیشینه مقدار در هر مرحله فیلتر میشود و علاوه بر کوچکتر شدن ابعاد ویژگی های آن نیز استخراج میشوند و در طول شبکه انتشار خواهند یافت.

بعد از عبور تصویر از تعداد مناسب لایه های کانولوشنی و همچنین لایه های بیشینه ادغام خروجی ویژگی ها از فیلتر هموار کننده عبور خواهند کرد و بصورت برداری از ویژگی ها آماده ورود به شبکه تمام متصل خواهند شد.

⁴ Max-pooling

⁵ fully-connected

در مرحله آخر مجموعه ای از لایه های نورونی خواهیم داشت که فرآیند طبقه بندی را انجام خواهند داد.

توجه شود که برای نایل شدن به خروجی های به فرمت one-hot میبایست در لایه آخر از softmax استفاده کنیم و براساس احتمال بیشینه ، احتمال منتظر به هر کلاس خروجی را بصورت one-hot تعیین نماییم.

```
: import zipfile, csv, os, io, string, pickle, math
import numpy as np
import matplotlib.pyplot as plt
import urllib.request as urllib
import pandas as pd
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
from numpy.random import randn
import random
```

```
: df_train= pd.read_csv("mnist_train.csv")
df_test = pd.read_csv("mnist_test.csv")
```

```
: data_train=df_train.to_numpy()
data_test=df_test.to_numpy()
```

```
TEST_RATIO = 0.15
VALIDATION_RATIO = 0.1
MAX_DIM = 1
MAX_LENGTH = 28
NUM_OF_CLASSES = 10
LR = 0.001
LR_DECAY = 0.95
BETA1 = 0.9
BETA2 = 0.999
NUM_OF_EPOCHS = 4
BATCH_SIZE = 100
SAVE_PATH = 'params.pkl'
```

```
X_train = data_train[:,1:]
y_train = data_train[:,0]
X_test = data_test[:,1:]
y_test = data_test[:,0]
```

Some Functions

```
: def function_builder(name, alpha=1, bias=0):
    def func(X):
        if name == 'Relu':
            X[X<=0] = 0
        elif name == 'LeakyRelu':
            X = np.max(alpha * X, X)
        elif name == 'ELU':
            X = alpha*(np.exp(X[X<=0])-1)
        elif name == 'Tanh':
            X = np.tanh(X)
        elif name == 'Linear':
            X = alpha*X + bias
        return X
    return func
```

```
def initializeFilter(size, scale = 1.0):
    stddev = scale/np.sqrt(np.prod(size))
    return np.random.normal(loc = 0, scale = stddev, size = size)

def initializeWeight(size):
    return np.random.standard_normal(size=size) * 0.01

def initializeParams ():
    m = [None]*len(Network)
    v = [None]*len(Network)
    mb = [None]*len(Network)
    vb = [None]*len(Network)

    for i, layer in enumerate(Network):
        if layer[0] == 'maxpool' or layer[0] == 'flatten':
            continue
        m[i] = np.zeros(layer[1].shape)
        v[i] = np.zeros(layer[1].shape)
        mb[i] = np.zeros(layer[2].shape)
        vb[i] = np.zeros(layer[2].shape)

    return [m, v, mb, vb]
```

```
def categoricalCrossEntropy(probs, label):
    return -np.sum(label * np.log(probs))
```

```
def nanargmax(arr):
    idx = np.nanargmax(arr)
    idxs = np.unravel_index(idx, arr.shape)
    return idxs
```

```
def calc_accuracy (X_test,Y_true):

    Y_pred = np.empty((np.shape(Y_true)))
    Y_probs = np.empty((np.shape(Y_true)))
    for i in range(len(X_test)):
        Y_pred[i], Y_probs[i] = predict(X_test[i])
    Y_pred = np.array(Y_pred).astype(int)
    return np.sum((Y_pred==Y_true).astype(int))/len(Y_true)*100, Y_pred
```

Layer Builders

```
def convolution(data, weights, bias, stride):
    filter_x, filter_y, filter_depth, filter_num = np.shape(weights)
    data_x, data_y, data_depth = np.shape(data)
    result = np.zeros((data_x, data_y, filter_num))
    pad_size_x = filter_x // 2
    pad_size_y = filter_y // 2
    padded_data = np.pad(data, ((pad_size_x, pad_size_x), (pad_size_y, pad_size_y), (0, 0)), 'constant', constant_values=0)
    for n in range(filter_num):
        kernel_dy = pad_size_y
        result_y = 0
        while kernel_dy < data_y:
            kernel_dx = pad_size_x
            result_x = 0
            while kernel_dx < data_x:
                result[result_x, result_y, n] = np.sum(np.multiply(weights[:, :, :, n], padded_data[kernel_dx:kernel_dx+filter_x,
                kernel_dy:kernel_dy+filter_y, :]))
                kernel_dx += stride
                result_x += 1
            kernel_dy += stride
            result_y += 1
        return result
```

```

def maxpool(data, pool_f): #pool_f is tuple (2, 1), pool_stride
    (data_x, data_y, data_depth) = np.shape(data)
    dim_x = math.ceil(data_x / pool_f[0])
    dim_y = math.ceil(data_y / pool_f[1])
    downsampled = np.zeros((dim_x, dim_y, data_depth))
    pad_size_x = pool_f[0] - data_x % pool_f[0]
    pad_size_y = pool_f[1] - data_y % pool_f[1]
    padded_data = np.pad(data, ((0, pad_size_x), (0, pad_size_y), (0, 0)), 'constant', constant_values=0)
    for i in range(data_depth):
        kernel_dy = 0
        result_y = 0
        while kernel_dy + pool_f[1] < np.shape(padded_data)[1]:
            kernel_dx = 0
            result_x = 0
            while kernel_dx + pool_f[0] < np.shape(padded_data)[0]:
                downsampled[result_x, result_y, i] = np.max(padded_data[kernel_dx:kernel_dx+pool_f[0], kernel_dy:kernel_dy+pool_f[1], i])
                kernel_dx += pool_f[0]
                result_x += 1
            kernel_dy += pool_f[1]
            result_y += 1
        return downsampled

```

```

def convolutionBackward(dconv_prev, conv_in, filt, s):
    (f_dim_x, f_dim_y, f_depth, num_f) = filt.shape
    (orig_dim_x, orig_dim_y, _) = conv_in.shape

    dout = np.zeros(conv_in.shape)
    dfilt = np.zeros(filt.shape)
    dbias = np.zeros((num_f, 1))

    for curr_f in range(num_f):
        curr_y = out_y = 0
        while curr_y + f_dim_y <= orig_dim_y:
            curr_x = out_x = 0
            while curr_x + f_dim_x <= orig_dim_x:
                dfilt[:, :, :, curr_f] += dconv_prev[out_x, out_y, curr_f] * conv_in[curr_x:curr_x + f_dim_x, curr_y:curr_y + f_dim_y, :]
                dout[curr_x:curr_x + f_dim_x, curr_y:curr_y + f_dim_y, :] += dconv_prev[out_x, out_y, curr_f] * filt[:, :, :, curr_f]
                curr_x += s
                out_x += 1
            curr_y += s
            out_y += 1

        dbias[curr_f] = np.sum(dconv_prev[:, :, :, curr_f])

    return dout, dfilt, dbias

```

```

def softmax(raw_preds):
    out = np.exp(raw_preds)
    return out/np.sum(out)

```

Design Network

```
# conv, maxpool, fullyconn, flatten, softmax
Network = []
Network.append(['conv', initializeFilter((5, 5, 1, 8)), np.zeros((8,1)), 'Relu']) #####
Network.append(['conv', initializeFilter((5, 5, 8, 8)), np.zeros((8,1)), 'Relu'])
Network.append(['maxpool', (2, 2)])

##Network.append(['conv', initializeFilter((5, 5, 8, 16)), np.zeros((16,1)), 'Relu']) #####
#Network.append(['conv', initializeFilter((5, 5, 16, 16)), np.zeros((16,1)), 'Relu'])
#Network.append(['maxpool', (2, 2)])

#Network.append(['conv', initializeFilter((5, 5, 16, 32)), np.zeros((32,1)), 'Relu']) #####
#Network.append(['conv', initializeFilter((5, 5, 32, 32)), np.zeros((32,1)), 'Relu'])
#Network.append(['maxpool', (2, 2)])

# Network.append(['conv', initializeFilter((3, 1, 64, 32)), np.zeros((32,1)), 'Relu'])
Network.append(['flatten',])

Network.append(['fullyconn', initializeWeight((200, 1568)), np.zeros((200,1)), 'Relu'])
Network.append(['fullyconn', initializeWeight((10, 200)), np.zeros((10,1)), 'Softmax'])
```

```
data_dim_x = MAX_LENGTH
data_dim_y = 1
data_depth = MAX_DIM
f_dim1 = 3
f_dim2 = 1
num_filt1 = 64
num_filt2 = 64
num_neurons = 128
flatten_size = 1600
```

Implementing Network

```
def CNN_forward (data):
    results = [None]*(len(Network)+1)
    results[0] = data
    for i, layer in enumerate(Network):
        if (layer[0] == 'conv'):
            results[i+1] = convolution(results[i], layer[1], layer[2], 1)
            results[i+1] = function_builder(layer[3])(results[i+1])
        elif (layer[0] == 'maxpool'):
            results[i+1] = maxpool(results[i], layer[1])
        elif (layer[0] == 'flatten'):
            dim1, dim2, dim3 = np.shape(results[i])
            results[i+1] = results[i].reshape((dim1*dim2*dim3, 1))
        elif (layer[0] == 'fullyconn'):
            results[i+1] = layer[1].dot(results[i]) + layer[2]
            if layer[3] == 'Softmax':
                results[i+1] = softmax(results[i+1])
                break
            results[i+1] = function_builder(layer[3])(results[i+1])
    return results
```



```
def CNN_backward (results, label):
    dNetwork = [[None, None]]*len(Network)
    dresults = [None]*(len(Network)+1)
    dresults[-1] = results[-1] - label
    for i, layer in reversed(list(enumerate(Network))):
        if (layer[0] == 'conv'):
            dresults[i], df, db = convolutionBackward(dresults[i+1], results[i], layer[1], 1)
            dNetwork[i] = [df, db]
            if i != 0 and Network[i-1][0] == 'conv':
                dresults[i][results[i]<=0] = 0
        elif (layer[0] == 'maxpool'):
            dresults[i] = maxpoolBackward(dresults[i+1], results[i], layer[1])
            dresults[i][results[i]<=0] = 0
        elif (layer[0] == 'flatten'):
            dresults[i] = dresults[i+1].reshape(np.shape(results[i]))
        elif (layer[0] == 'fullyconn'):
            dw = dresults[i+1].dot(results[i].T)
            db = np.sum(dresults[i+1], axis=1).reshape(layer[2].shape)
            dNetwork[i] = [dw, db]
            dresults[i] = layer[1].T.dot(dresults[i+1])
            if Network[i-1][0] == 'flatten':
                continue
            dresults[i][results[i]<=0] = 0
    return dNetwork
```

```
def CNN (data, label):
    results = CNN_forward (data)
    dNetwork = CNN_backward(results, label)

    loss = categoricalCrossEntropy(results[-1], label)
    grads = dNetwork

    return grads, loss
```

Solver

```
def momentumGD(dNetwork, lr, params):
    [m, v, mb, vb] = params
    for i, layer in enumerate(Network):
        if layer[0] == 'maxpool' or layer[0] == 'flatten':
            continue
        v[i] = GAMMA*v[i] + lr*(dNetwork[i][0]/BATCH_SIZE)
        layer[1] -= v[i]

        vb[i] = GAMMA*vb[i] + lr*(dNetwork[i][1]/BATCH_SIZE)
        layer[2] -= vb[i]

    return [m, v, mb, vb]
```

```
def adamGD (dNetwork, lr, params):
    [m, v, mb, vb] = params
    for i, layer in enumerate(Network):
        if layer[0] == 'maxpool' or layer[0] == 'flatten':
            continue
        m[i] = BETA1*m[i] + (1-BETA1)*dNetwork[i][0]/BATCH_SIZE
        v[i] = BETA2*v[i] + (1-BETA2)*(dNetwork[i][0]/BATCH_SIZE)**2
        m_hat = m[i]/(1-BETA1)
        v_hat = v[i]/(1-BETA2)
        layer[1] -= lr * m_hat/(np.sqrt(v_hat)+1e-7)

        mb[i] = BETA1*mb[i] + (1-BETA1)*dNetwork[i][1]/BATCH_SIZE
        vb[i] = BETA2*vb[i] + (1-BETA2)*(dNetwork[i][1]/BATCH_SIZE)**2
        mb_hat = mb[i]/(1-BETA1)
        vb_hat = vb[i]/(1-BETA2)
        layer[2] -= lr * mb_hat/(np.sqrt(vb_hat)+1e-7)

    return [m, v, mb, vb]
```

```

def solver(batch, lr, params, cost, optimizer='ADAM'):
    |
    X = batch[:,0:-1]
    Y = batch[:,1]

    X = X.reshape(len(batch),MAX_LENGTH,MAX_LENGTH,MAX_DIM)

    cost_ = 0

    dNetwork = [[0, 0]]*len(Network)

    for i in range(len(X)):
        x = X[i]
        y = np.eye(NUM_OF_CLASSES)[int(Y[i])].reshape(NUM_OF_CLASSES, 1)

        grads, loss = CNN(x, y)
        for j, grad in enumerate(grads):
            if grad[0] is None:
                continue
            if i == 0:
                dNetwork[j] = [np.zeros(grad[0].shape), np.zeros(grad[1].shape)]
            dNetwork[j][0] += grad[0]
            dNetwork[j][1] += grad[1]
        cost_ += loss

    if optimizer == 'ADAM':
        new_params = adamGD(dNetwork, lr, params)
    elif optimizer == 'MOMENTUM':
        new_params = momentumGD(dNetwork, lr, params)

    cost_ = cost_/BATCH_SIZE
    cost.append(cost_)

    return new_params, cost

params = initializeParams()

m = 5000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
X=(X-mean)/std
trainData = np.hstack((X,y_dash))
cost = []
print("LR:"+str(LR)+" , Batch Size:"+str(BATCH_SIZE))
for epoch in range(NUM_OF_EPOCHS):
    np.random.shuffle(trainData)
    batches = np.array([trainData[k:k + BATCH_SIZE] for k in range(0, trainData.shape[0], BATCH_SIZE)])
    t = tqdm(batches)
    for x,batch in enumerate(t):
        params, cost = solver(batch, LR*(LR_DECAY**x), params, cost, 'ADAM')
        t.set_description("Cost: %.6f" % (cost[-1]))
with open(SAVE_PATH, 'wb') as file:
    pickle.dump(Network, file)

```

Loading Parameters from Directory

```
Network = pickle.load(open('Network.plk', 'rb'))
```

Test

```

m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X=(X-mean)/std
X = X.reshape(len(X),28,28,1)

acc, Y_pred = calc_accuracy(X,y_dash)

```

Image Generation

```
def CNN_backward (results, label):
    dNetwork = [[None, None]]*len(Network)
    dresults = [None]*(len(Network)+1)
    dresults[-1] = results[-1] - label
    for i, layer in reversed(list(enumerate(Network))):
        if (layer[0] == 'conv'):
            dresults[i], df, db = convolutionBackward(dresults[i+1], results[i], layer[1], 1)
            dNetwork[i] = [df, db]
            if i != 0 and Network[i-1][0] == 'conv':
                dresults[i][results[i]<=0] = 0
        elif (layer[0] == 'maxpool'):
            dresults[i] = maxpoolBackward(dresults[i+1], results[i], layer[1])
            dresults[i][results[i]<=0] = 0
        elif (layer[0] == 'flatten'):
            dresults[i] = dresults[i+1].reshape(np.shape(results[i]))
        elif (layer[0] == 'fullyconn'):
            dw = dresults[i+1].dot(results[i].T)
            db = np.sum(dresults[i+1], axis=1).reshape(layer[2].shape)
            dNetwork[i] = [dw, db]
            dresults[i] = layer[1].T.dot(dresults[i+1])
            if Network[i-1][0] == 'flatten':
                continue
            dresults[i][results[i]<=0] = 0
    return dNetwork, dresults[0]
```

```
def CNN (data, label):
    results = CNN_forward (data)
    dNetwork, dresults = CNN_backward(results, label)

    loss = categoricalCrossEntropy(results[-1], label)
    grads = dNetwork

    return dresults, loss
```

```
our_noise=255 * np.random.rand(28, 28, 1)

image=np.zeros((28, 28, 1))
beta1 = 0.9
beta2 = 0.999
lr = 0.001
print(mean)
print(std)
mimage=np.mean(image)
simage=np.std(image)

image=image-mean
image=image/std

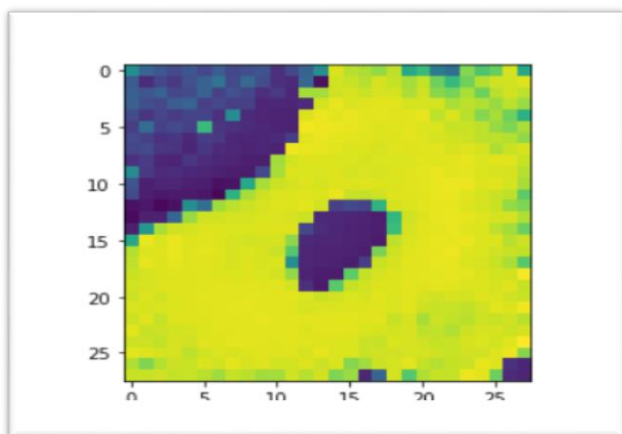
v1 = np.zeros(image.shape)
s1 = np.zeros(image.shape)

for i in range(100):
    label=9
    y = np.eye(10)[int(label)].reshape(10, 1)
    dresults, loss=CNN(image,y)
    print(dresults.shape)
    v1 = beta1*v1 + (1-beta1)*dresults
    s1 = beta2*s1 + (1-beta2)*(dresults)**2

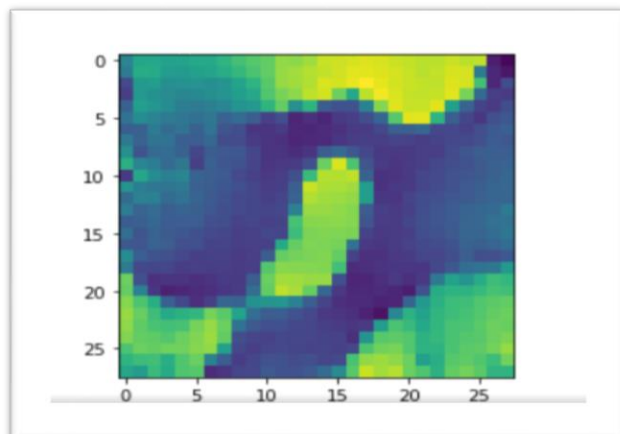
    v1_hat = v1/(1-beta1)
    s1_hat = s1/(1-beta2)
    image= lr * v1_hat/(np.sqrt(s1_hat)*1e-7)
    print(loss)

plt.imshow(image)
```

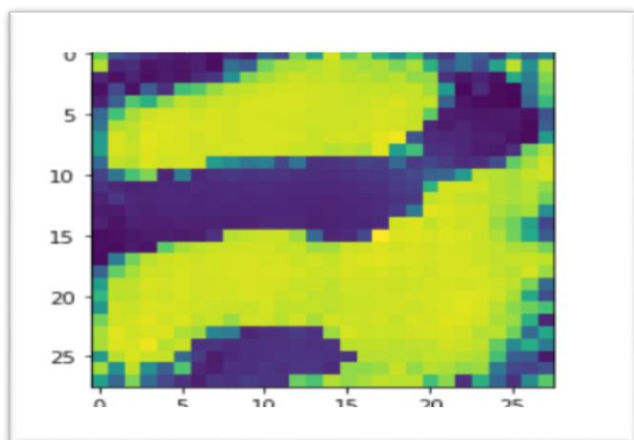
در ادامه خروجی های مربوط به عکس های تولید شده از هر کلاس آورده شده است توجه شود که مراحل کار و راهکارهای بهبود کیفیت در قسمت GAN به همراه خروجی های آنها آورده شده اند.



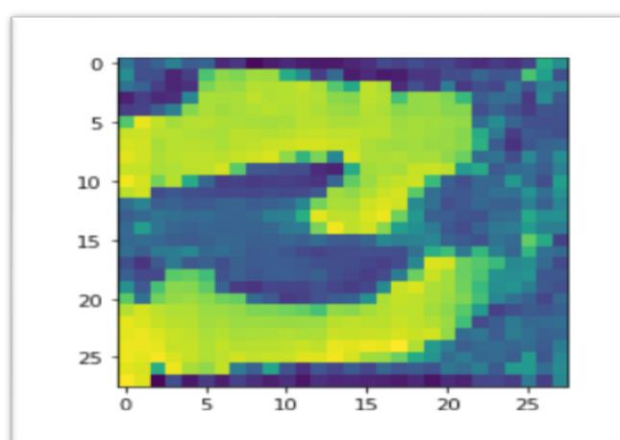
خروجی کلاس [0]



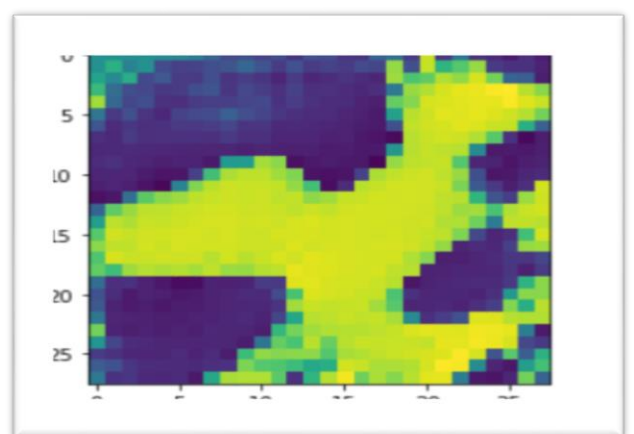
خروجی کلاس [1]



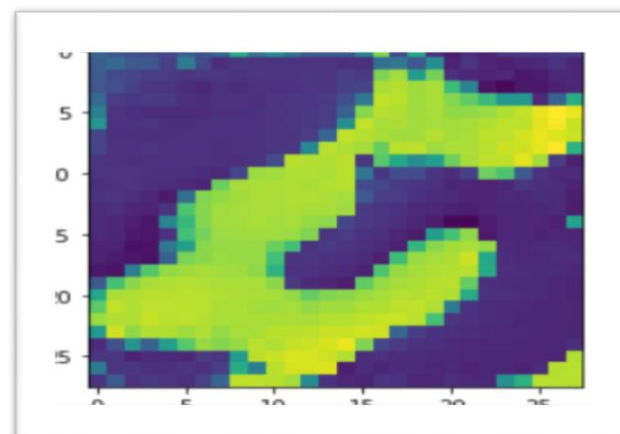
خروجی کلاس [2]



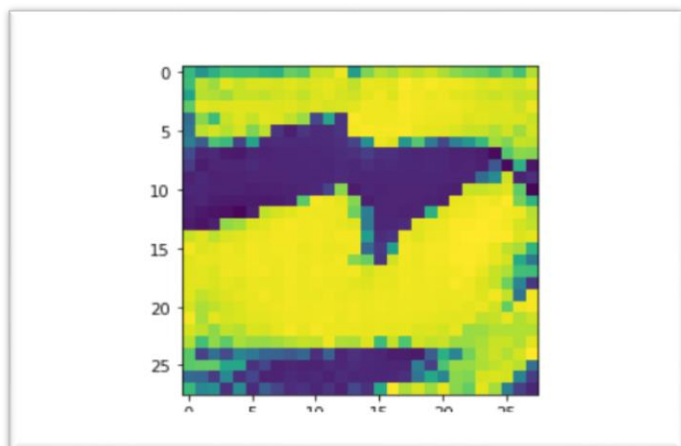
خروجی کلاس [3]



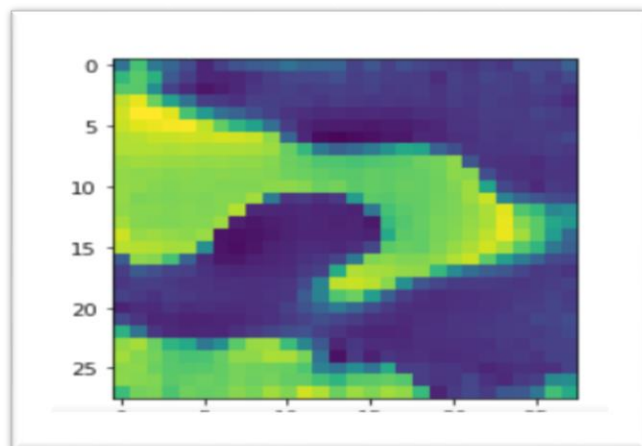
خروجی کلاس [4]



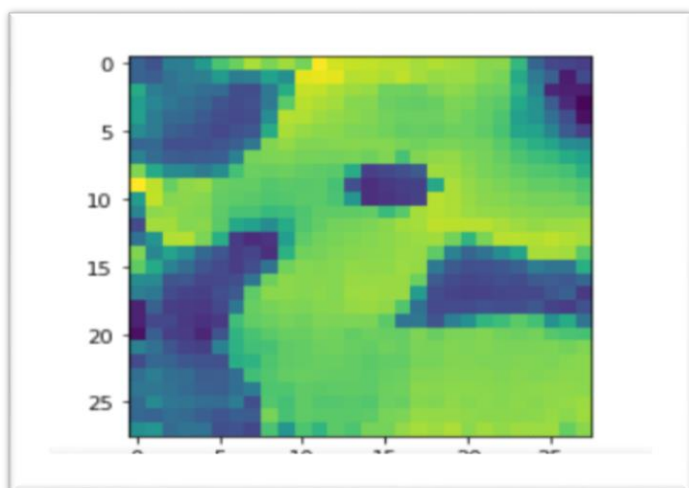
خروجی کلاس [5]



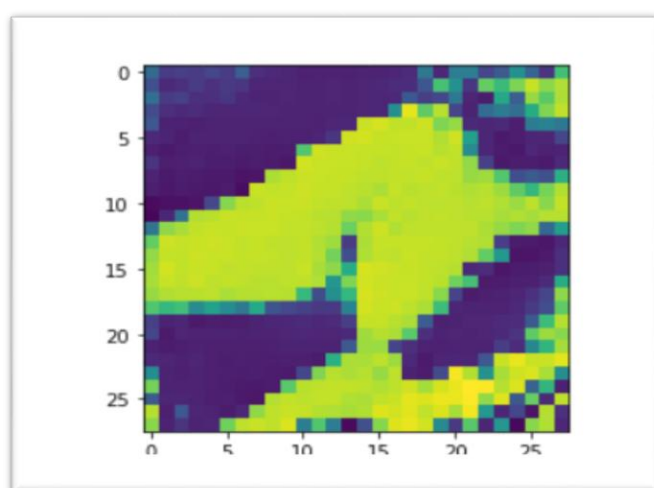
خروجی کلاس [6]



خروجی کلاس [7]



خروجی کلاس [8]



خروجی کلاس [9]

تحلیل کامل بخش های فوق در ادامه بصورت مفصل خواهد آمد و راهکارهای عملی در راستای افزایش کیفیت آنها نیز ارایه خواهد شد.

همانطور که در توضیحات اولیه این بخش نیز اشاره شد هسته اصلی این قسمت مربوطه به آموزش شبکه پیچشی برای طبقه بندی داده های ورودی است.

در کدهای بالا همانطوری که مشاهده می نمایید در وهله اول دیتاست مفروض در محیط پایتون خوانده می شود و بعد از پیش پردازش های اولیه و جداسازی داده های تست و آموزش ، با استفاده از پارامترهای اولیه مسئله یعنی تعداد 200 نورون در لایه مخفی⁶ شبکه تماما متصل و استفاده از 2 لایه پیچشی⁷ و همچنین یک لایه حداکثر ادغام⁸ به استخراج ویژگی ها خواهیم پرداخت و در نهایت در طبقه بند تماما متصل به طبقه بندی دادگان خواهیم پرداخت.

در اینجا نیز قبل از نمایش نتایج آموزش شبکه توجه به پارامترهای مورد استفاده ضروری است و ذکر این نکته لازم است که در مواردی مانند پارامترهای مربوط به بهینه ساز مسئله ما سعی کرده ایم تا حد امکان این پارامترها را با تست مقادیر مختلف بدست آوریم ولی در مواردی نیز همانند پارامترهای بهینه ساز مفروض با مراجعه به مقالات موجود و نتایج کار آنها این پارامترها مورد استفاده قرار گرفته اند که به حداکثر دقت در فرایند طبقه بندی منجر شود.

```
TEST_RATIO = 0.15
VALIDATION_RATIO = 0.1
MAX_DIM = 1
MAX_LENGTH = 28
NUM_OF_CLASSES = 10
LR = 0.001
LR_DECAY = 0.95
BETA1 = 0.9
BETA2 = 0.999
NUM_OF_EPOCHS = 4
BATCH_SIZE = 100
SAVE_PATH = 'params.pkl'
```

```
data_dim_x = MAX_LENGTH
data_dim_y = 1
data_depth = MAX_DIM
f_dim1 = 3
f_dim2 = 1
num_filt1 = 64
num_filt2 = 64
num_neurons = 200
flatten_size = 1600
```

Figure3: پارامترهای مورد استفاده

برای استخراج ویژگی ها در بهترین حالت ممکن و متناسب با اندازه تصاویر ورودی تعداد 64 فیلتر برای کانال های اولیه و ثانویه در نظر گرفته شده است.

همچنین طبق تست های انجام شده با انجام 4 تکرار روی داده های آموزش به دقت معقولی خواهیم رسید.

⁶ Hidden-layer

⁷ Convolutional layer

⁸ Max-pooling layer

همچنین تعداد نورون ها نیز مطابق خواسته های سوال درقسمت اول برابر 200 نورون لحاظ شده است.

در ادامه نتایج آموزش با این مجموعه پارامترها ضمیمه خواهد شد:

LR:0.001, Batch Size:100

Cost: 0.567548: 100%	50/50 [26:27<00:00, 31.76s/it]
Cost: 0.336795: 100%	50/50 [25:47<00:00, 30.95s/it]
Cost: 0.173551: 100%	50/50 [24:55<00:00, 29.90s/it]
Cost: 0.244342: 100%	50/50 [23:56<00:00, 28.73s/it]

Figure4: آموزش شبکه در 4 تکرار

Test

```
m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X=(X-mean)/std
X = X.reshape(len(X),28,28,1)
acc, Y_pred = calc_accuracy(X,y_dash)
print("accuracy is equal to:",acc)
```

accuracy is equal to: 91.0

Figure5: تست شبکه بر روی داده های تست

توجه شود که باتوجه به زمان زیادی که برای آموزش با استفاده از بیش از 5000 دیتا موردنیاز بود تصمیم براین شد که با 5000 داده آموزش را انجام دهیم.

همانطور که ملاحظه می نمایید دقت 91 درصد دراین حالت بدست می آید که حاکی از آموزش دقیق شبکه ما خواهد بود.

در ادامه قصد داریم مطابق خواسته سوال تعداد نورون های شبکه را تغییر دهیم و نتیجه را بررسی نماییم لذا دراین راستا شبکه را بامقادیر $n=100,200,300,400,500$ آموزش خواهیم داد و اثر این تغییر را تحلیل خواهیم کرد.

تعداد نورون ها نیز در این قسمت برابر 100 نورون لحاظ شده است.

در ادامه نتایج آموزش با این مجموعه پارامترها ضمیمه خواهد شد:

LR:0.001, Batch Size:100

Cost: 1.961660: 100%	10/10 [04:26<00:00, 26.64s/it]
Cost: 0.818847: 100%	10/10 [04:54<00:00, 29.46s/it]
Cost: 0.624897: 100%	10/10 [04:36<00:00, 27.69s/it]
Cost: 0.710733: 100%	10/10 [04:32<00:00, 27.27s/it]

Figure6: آموزش شبکه در 4 تکرار

Test

```
m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean = int(np.mean(X))
std = int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X = (X - mean) / std
X = X.reshape(len(X), 28, 28, 1)
acc, Y_pred = calc_accuracy(X, y_dash)
print("accuracy is equal to:", acc)
```

accuracy is equal to: 81.0

Figure7: تست شبکه بر روی داده های تست

همانطور که ملاحظه میشود با کاهش تعداد نورون های لایه مخفی دقت آموزش ما کاهش می یابد که تاثیر بسزایی در پایین آمدن کیفیت عکس های تولیدی در مرحله بعد نیز خواهد داشت.

تعداد نورون ها نیز در این قسمت برابر 200 نورون لحاظ شده است.

در ادامه نتایج آموزش با این مجموعه پارامترها ضمیمه خواهد شد:

LR:0.001, Batch Size:100

Cost: 1.834560: 100%	10/10 [06:13<00:00, 37.37s/it]
Cost: 0.642432: 100%	10/10 [05:02<00:00, 30.27s/it]
Cost: 0.448097: 100%	10/10 [04:23<00:00, 26.30s/it]
Cost: 0.576957: 100%	10/10 [04:24<00:00, 26.43s/it]

Figure8: آموزش شبکه در 4 تکرار

Test

```
m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X=(X-mean)/std
X = X.reshape(len(X),28,28,1)
acc, Y_pred = calc_accuracy(X,y_dash)
print("accuracy is equal to:",acc)

accuracy is equal to: 88.0
```

Figure9: تست شبکه بر روی داده های تست

تعداد نورون ها نیز در این قسمت برابر 300 نورون لحاظ شده است.

در ادامه نتایج آموزش با این مجموعه پارامترها ضمیمه خواهد شد:

```
LR:0.001, Batch Size:100
Cost: 1.738840: 100%| 10/10 [06:22<00:00, 38.20s/it]
Cost: 0.780353: 100%| 10/10 [05:42<00:00, 34.28s/it]
Cost: 0.475196: 100%| 10/10 [04:32<00:00, 27.27s/it]
Cost: 0.453892: 100%| 10/10 [07:01<00:00, 42.17s/it]
```

Figure10: آموزش شبکه در 4 تکرار

Test

```
m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X=(X-mean)/std
X = X.reshape(len(X),28,28,1)
acc, Y_pred = calc_accuracy(X,y_dash)
print("accuracy is equal to:",acc)

accuracy is equal to: 86.0
```

Figure11: تست شبکه بر روی داده های تست

تعداد نورون ها نیز در این قسمت برابر 400 نورون لحاظ شده است.

در ادامه نتایج آموزش با این مجموعه پارامترها ضمیمه خواهد شد:

```
LR:0.001, Batch Size:100
Cost: 1.655759: 100%| 10/10 [06:27<00:00, 38.80s/it]
Cost: 0.716172: 100%| 10/10 [05:34<00:00, 33.46s/it]
Cost: 0.507406: 100%| 10/10 [07:09<00:00, 42.90s/it]
Cost: 0.301088: 100%| 10/10 [06:33<00:00, 39.40s/it]
```

Figure12: آموزش شبکه در 4 تکرار

```
m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X=(X-mean)/std
X = X.reshape(len(X),28,28,1)
acc, Y_pred = calc_accuracy(X,y_dash)
print("accuracy is equal to:",acc)

accuracy is equal to: 90.5
```

Figure13: تست شبکه بر روی داده های تست

تعداد نورون ها نیز در این قسمت برابر 500 نورون لحاظ شده است.

در ادامه نتایج آموزش با این مجموعه پارامترها ضمیمه خواهد شد:

```
0%| 0/10 [00:00<?, ?it/s]
LR:0.001, Batch Size:100
Cost: 1.512167: 100%| 10/10 [06:41<00:00, 40.14s/it]
Cost: 0.420923: 100%| 10/10 [04:36<00:00, 27.68s/it]
Cost: 0.188395: 100%| 10/10 [05:45<00:00, 34.57s/it]
Cost: 0.303492: 100%| 10/10 [05:39<00:00, 34.00s/it]
```

Figure14: آموزش شبکه در 4 تکرار

Test

```
m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X=(X-mean)/std
X = X.reshape(len(X),28,28,1)
acc, Y_pred = calc_accuracy(X,y_dash)
print("accuracy is equal to:",acc)

accuracy is equal to: 87.5
```

Figure15: تست شبکه بر روی داده های تست

تعداد نورون ها نیز در این قسمت برابر 600 نورون لحاظ شده است.

در ادامه نتایج آموزش با این مجموعه پارامترها ضمیمه خواهد شد:

LR:0.001, Batch Size:100

Cost: 1.209839: 100%	10/10 [04:48<00:00, 28.88s/it]
Cost: 0.707566: 100%	10/10 [08:13<00:00, 49.38s/it]
Cost: 0.436083: 100%	10/10 [05:32<00:00, 33.21s/it]
Cost: 0.298203: 100%	10/10 [05:43<00:00, 34.39s/it]

Figure16: آموزش شبکه در 4 تکرار

Test

```
m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X=(X-mean)/std
X = X.reshape(len(X),28,28,1)
acc, Y_pred = calc_accuracy(X,y_dash)
print("accuracy is equal to:",acc)
```

accuracy is equal to: 93.5

Figure17: تست شبکه بر روی داده های تست

همانطور که ملاحظه می نمایید افزایش تعداد نورون های هر لایه بصورت صریح موجب افزایش یا کاهش دقت طبقه بند خواهند شد و لذا نمیتوان حکم کلی درمورد اثرات آنها داشت اما چیزی که مسلم است این است که همانطور که در لکچرهای درس نیز ارایه شد تعداد نورون های لایه مخفی به عنوان یک هایپرپارامتر در صورت افزایش از یکجایی به بعد ممکن است منجر به کاهش دقت شود و لذا نمیتوان حکم کلی داد و این پارامترها می بایست با استفاده از تست حالات مختلف در طول ارزیابی بدست بیایند.

در بخش بعدی قصد داریم که با ثابت نگه داشتن مقادیر نورون ها تعداد لایه های پیچشی^۹ را تغییر دهیم و بررسی کنیم که چه تاثیری بر دقت شبکه ما و همچنین تولید عکس های جدید به عنوان هدف اصلی این قسمت خواهد گذاشت:

تغییرات تعداد لایه های پیچشی^{۱۰} را یکبار در حالت افزایش تعداد لایه های پیچشی بررسی میکنیم و بار دیگر این تغییر را با کاهش تعداد لایه های پیچشی تحلیل خواهیم کرد:

Design Network

```
# conv, maxpool, fullyconn, flatten, softmax
Network = []
Network.append(['conv', initializeFilter((5, 5, 1, 8)), np.zeros((8,1)), 'Relu']) #####
Network.append(['conv', initializeFilter((5, 5, 8, 8)), np.zeros((8,1)), 'Relu'])
Network.append(['maxpool', (2, 2)])

Network.append(['conv', initializeFilter((5, 5, 8, 16)), np.zeros((16,1)), 'Relu']) #####
Network.append(['conv', initializeFilter((5, 5, 16, 16)), np.zeros((16,1)), 'Relu'])
Network.append(['maxpool', (2, 2)])

Network.append(['conv', initializeFilter((5, 5, 16, 32)), np.zeros((32,1)), 'Relu']) #####
Network.append(['conv', initializeFilter((5, 5, 32, 32)), np.zeros((32,1)), 'Relu'])
Network.append(['maxpool', (2, 2)])

Network.append(['conv', initializeFilter((3, 1, 32, 32)), np.zeros((32,1)), 'Relu'])
Network.append(['flatten',])

Network.append(['fullyconn', initializeWeight((200, 512)), np.zeros((200,1)), 'Relu'])
Network.append(['fullyconn', initializeWeight((10, 200)), np.zeros((10,1)), 'Softmax'])
```

Figure18: طراحی شبکه با افزایش لایه های پیچشی

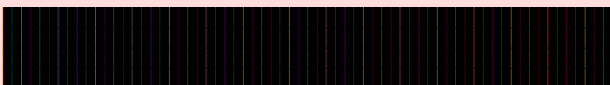
0%			0/10 [00:00<?, ?it/s]
LR:0.001, Batch Size:100			
Cost: 2.302599: 100%		10/10	[11:16<00:00, 67.70s/it]
Cost: 2.302767: 100%		10/10	[10:09<00:00, 60.94s/it]
Cost: 2.306021: 100%		10/10	[08:42<00:00, 52.28s/it]
Cost: 2.139975: 100%		10/10	[08:38<00:00, 51.85s/it]

Figure19: آموزش شبکه در 4 تکرار

⁹ Convolutional Layer

¹⁰ Convolutional Layer

Test

```
m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X=(X-mean)/std
X = X.reshape(len(X),28,28,1)
acc, Y_pred = calc_accuracy(X,y_dash)
print("accuracy is equal to:",acc)
```

accuracy is equal to: 12.0

Figure20: تست شبکه بر روی داده های تست

همانطور که ملاحظه می نمایید باافزایش لایه های کانولوشنی دقت ما بشدت کاهش می یابد و این حاکی از انتخاب درست معماری شبکه به عنوان یک امر مهم خواهد بود لذا نباید انتظار داشته باشیم که باافزایش تعداد لایه های پیچشی همواره به دقت و عملکرد بهتر خواهیم رسید.

همانطور که ملاحظه کردید بااین افت دقت عملاً قادر نیز نخواهیم بود که عمل تولید تصویر را انجام دهیم زیرا شرط اساسی در این امر آموزش درست به همراه معماری درست برای شبکه ما خواهد بود.

GAN 1.1

همانطور که در قسمت های قبل نیز اشاره شد علی رغم تلاش های زیادی که برای افزایش کیفیت عکس های تولید شده به روش گفته شده در صورت سوال گردید بدلیل اینکه شبکه ما فقط شامل یک مدل پیچشی^{۱۱} است امکان گرفتن کیفیت بیش از این با ساختار مفروض وجود ندارد.

لذا طبق بررسی های انجام شده درحوزه مدل های مولد درتولید عکس بااستفاده از شبکه های عصبی ما متوجه شدیم که روش های گوناگونی برای این امر وجود دارد منتها این ساختارها همگی درقسمتی مشترک خواهند بود و آن حضور و تعامل دو شبکه پیچشی^{۱۲} درآنها خواهد بود که درادامه قصدداریم معروف ترین این الگوریتم ها را پیاده سازی نماییم تا بتوانیم کیفیت حداکثری برروی عکس های تولیدی داشته باشیم.

شبکه GAN:

ساختار این شبکه ی مولد بصورت زیر خواهد بود:

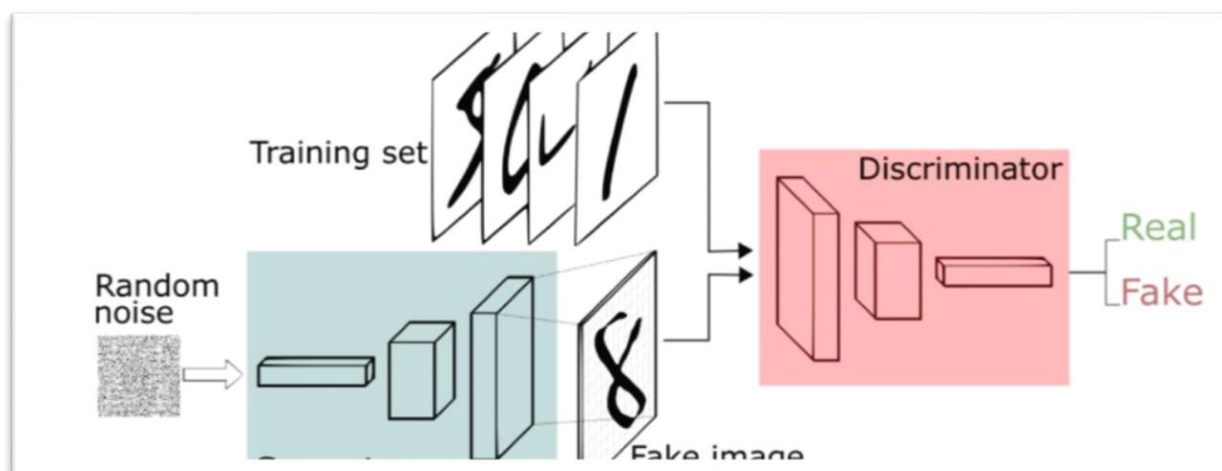


Figure21: تست شبکه بر روی داده های تست

همانطور که مشاهده مینمایید این ساختار از دو شبکه بگونه ای استفاده میکند که میتواند به کمک شبکه مولد بیاید و کیفیت عکس ها را بطور قابل ملاحظه ای افزایش دهد.

درادامه درمورد جزییات این امر بیشتر صحبت خواهیم کرد:

¹¹ CNN

¹² CNN

شبکه مفروض شامل یک شبکه پیچشی مولد می باشد که این شبکه وظیفه تولید عکس های غیرواقعی را برعهده دارد درواقع عکس هایی که تولید میکند همان عکس های کلاسی میباشند که ما قصد تولید آنها را خواهیم داشت منتها نکته قابل توجه دراختار این شبکه این است که شبکه دومی آموزش دیده شده است تا درکنار شبکه ی قبلی کار کند و به ما بگوید که آیا عکس تولید شده ای که به ورودی این شبکه اعمال شده است یک عکس حقیقی است یا ازمجموعه عکس های تولیدی است. درواقع دراین ساختار بایک مجموعه روبه رو هستیم که باتعامل باهم میتوانند عکس هایی با کیفیت بهتر را تولید نمایند.

شبکه دوم^{۱۳} برحسب مجموعه ای ازداده ها آموزش دیده است که تعیین کند آیاعکس مفروض یک عکس واقعی است یا یک عکس جعلی خواهد بود درنهایت بادادن فیدبک به شبکه مولد^{۱۴} این ساختار انقدر انجام میشود تا زمانی که شبکه حقیقی یاب نتواند تشخیص دهد که داده ای که که شبکه مولد تولید کرده است جعلی است و آن زمان این خروجی به عنوان خروجی مطلوب به خروجی پاس داده خواهد شد.

همانگونه که ملاحظه می فرمایید از توضیحات ارایه شده نیز کاملاً مشهود است که این شبکه قدرت فوق العاده بیشتری درامر تولید عکس خواهد داشت و لذا به کیفیت بی نظیری دراین امر دست خواهیم یافت.

* همانطور که توضیح دادیم علی رغم اینکه ما تمامی پارامترهای طراحی شبکه اولیه را تغییر دادیم نتوانستیم به کیفیت بی نظیر دست یابیم لذا برآن شدیم که یک شبکه مولد بااین روش مرسوم طراحی نماییم و عکس ها را باکیفیت بسیار بالانیز تولید نماییم که حتی فراتر از خواسته سوال را برآورده کرده باشیم.

درادامه ابتدا مراحل که برای بهبود عملکرد مدل قبلی انجام شد را بیان میداریم و درانتها نیز به سراغ پیاده سازی GAN ازمعروف ترین شبکه های مولد برای تولید عکس خواهیم پرداخت و نتایج آن را به تصویر خواهیم کشید.

همانطور که دردید اولیه نیز ممکن است به ذهن برسد یکی از دلایل اصلی که مدل تک شبکه ای نمی تواند به کیفیت عالی دست یابد این است که ما درآنجا تنها از گرادیان گرفتن نسبت به عکس ورودی و اپدیت کردن این مراحل تا رسیدن به همگرایی یعنی رسیدن به هزینه صفر استفاده خواهیم کرد.

درواقع درفرایند طبقه بندی ما تنها قصد داریم تفاوت بین این 10 کلاس را بیابیم و لذا مدل ما بعد از آموزش کافی برحسب الگوهای تفاوتی ضعیفی که بین این کلاس ها پیدا کرده است به انجام فرایند طبقه بندی اقدام میکند درحالی که این مدل فقط و فقط داده هایی از نوع این ارقام دیده است و مجموعه مورد بررسی آن داده گان با واریانس کم خواهند بود و لذا برای مثال مدل ما فقط به تفاوت های کوچک بین اینها حساس خواهد بود و الگوهای ضعیفی را خواهد آموخت که منجر میشود درمرحله گرادیان گیری و علی رغم اینکه مدل ما به خوبی تابع هزینه را کاهش میدهد و احتمالات کلاس مفروض نیز در روندی درست افزایش می یابد ولی کیفیت عکس خروجی پایین خواهد بود چرا که همانطور که توضیح دادیم مدل ما فقط الگوهای ضعیف تفاوتی بین دادگان را آموخته و چیزی را که به عنوان خروجی پاس میدهد به خیال خود عکسی است که ما

¹³ Discriminator

¹⁴ Generator

کلاس آن را به مولد داده ایم که البته چنین کیفیت های عکسی که در قسمت های قبلی ضمیمه شد با این ساختار و معماری ضعیف شبکه که در این سوال از ما خواسته شده از آن استفاده کنیم بسیار ستودنی و بسیار عالی است زیرا الگوهای رفتاری اعداد کاملاً در خروجی این مولد عکس آشکار خواهد بود ولی همانطور که اشاره شد از آنجا که این مدل عکس های متفاوت با این دادگان را ندیده است در امر کیفیت ضعیف ظاهر میشود.

راهکار اول: بعد از بررسی های متعدد و دادن ورودی های مختلف به مولد طراحی شده متوجه شدیم که علی رغم استفاده از بهترین و موثر ترین بهینه ساز یعنی بهینه ساز ADAM ولی بدلیل موارد گفته شده در بعضی مواقع الگوریتم در کمینه های محلی گیر میکرد که موجب میشد خروجی عکس ما ایده آل ترین حالت ممکن برای این بهینه ساز نباشد و در واقع علی رغم یکسان بودن تابع هزینه در تکرار آخر ولی به دو مقدار متفاوت همگرا میشد لذا راهکاری که ما اندیشیدیم این بود که بازای چندین بردار نویز مختلف الگوریتم را اجرا نماییم و در انتها از مجموعه پیکسل های این تکرارهای مختلف میانگین گیری نماییم تا به کیفیت بهتری دست یابیم.

راهکار دوم: این راهکار به نوعی پیاده سازی شبکه گفته شده در الگوریتم GAN خواهد بود که میتواند کیفیت کار ما را بهبود دهد بدین صورت که ما یک کلاس 11 ام تحت عنوان کلاس داده های غیر شبیه اضافه میکنیم و در این صورت و در واقع با آموزش روی داده های غیر واقعی که خودمان به مجموعه داده های آموزش اضافه میکنیم مدل مان بگونه ای تقویت میشود و مجبور خواهد بود که تفاوت های عمیق تری را بین داده های ارقام و داده های غیر مربوط بیاموزد و در این صورت مدل ما در فرایند تشخیصی دیگر فقط به یکسری الگوهای تفاوتی کلی نگاه نخواهد کرد و در واقع مجموعه ای از الگوهای قوی را خواهد آموخت که میتواند جایگزین شبکه دوم^{۱۵} مورد استفاده در GAN گردد.

در ادامه به پیاده سازی یک شبکه GAN مبتنی بر دیتاست MNIST میپردازیم تا اثر افزایش کیفیت را به وضوح ملاحظه نماییم:

¹⁵ Discriminator

در ادامه کدهای مربوطه ضمیمه خواهند شد: (توجه شود این قسمت به عنوان فعالیت امتیازی در این پروژه انجام شده است)

```
# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1)):
    model = Sequential()
    # normal
    model.add(Conv2D(64, (3,3), padding='same', input_shape=in_shape))
    model.add(LeakyReLU(alpha=0.2))
    # downsample
    model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # downsample
    model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # downsample
    model.add(Conv2D(256, (3,3), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # classifier
    model.add(Flatten())
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
```

تعریف مدل Discriminator

```
# define the standalone generator model
def define_generator(latent_dim):
    model = Sequential()
    # foundation for 7x7 image
    n_nodes = 256 * 7 * 7
    model.add(Dense(n_nodes, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((7, 7, 256)))
    # upsample to 14x14
    model.add(Conv2DTranspose(128, (7,7), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # upsample to 28x28
    model.add(Conv2DTranspose(32, (7,7), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # output layer
    model.add(Conv2D(1, (3,3), activation='tanh', padding='same'))
    return model
```

تعریف مدل مولد تصویر

ساختار حقیقت یاب با چهار لایه که ورودی آن تصاویر سیاه و سفید 28×28 بوده و خروجی آن یک مقدار بله و خیر با این عنوان که عکس ورودی ساختگی بوده و یا نه می باشد.

مدل مولد^{۱۶} که یک فضای برداری تصادفی با اندازه ی 100 را دریافت کرده و با یک شبکه ی تماماً متصل به 256 لایه 7*7 تبدیل کرده سپس در دو لایه عکس کانولوشن باگام 2 در هر مرحله طول و عرض تصویر دو برابر شده تا به اندازه ی تصویر ورودی یعنی 28*28 برسیم.

```
# define the combined generator and discriminator model, for updating the generator
def define_gan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect them
    model = Sequential()
    # add generator
    model.add(g_model)
    # add the discriminator
    model.add(d_model)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model
```

تعریف مدل GAN

در اینجا باید توجه داشت که مدل حقیقت یاب نباید در مدل کامل GAN آموزش ببیند بلکه در هر مرحله با تعداد برابر داده های واقعی و ساختگی تصادفی آموزش داده می شود.

```
# load and prepare MNIST training images
def load_real_samples():
    # load MNIST dataset
    (trainX, _), (_, _) = load_data()
    # convert from unsigned ints to floats
    x = trainX.astype('float32')
    # scale from [0,255] to [-1,1]
    x = (x - 127.5) / 127.5
    return x

# select real samples
def generate_real_samples(dataset, n_samples):
    # choose random instances
    ix = randint(0, dataset.shape[0], n_samples)
    # retrieve selected images
    x = dataset[ix]
    # generate 'real' class labels (1)
    y = ones((n_samples, 1))
    return x, y
```

انتخاب تعدادی از داده های دیتاست اصلی برای آموزش

¹⁶ generator

و مقدار خروجی مورد انتظار discriminator که در اینجا 1 می باشد چون داده ها واقعی هستند.

```
# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(g_model, latent_dim, n_samples):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    X = g_model.predict(x_input)
    # create 'fake' class labels (0)
    y = zeros((n_samples, 1))
    return X, y
```

ایجاد بردار تصادفی و استفاده از شبکه مولد برای ایجاد تصاویر ساختگی

مقدار خروجی حقیقت یاب در اینجا صفر بوده چون داده ها جعلی و تولیدی مولد می باشند.

```
# train the generator and discriminator
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=10, n_batch=128):
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)
    # manually enumerate epochs
    for i in range(n_epochs):
        # enumerate batches over the training set
        for j in range(bat_per_epo):
            # get randomly selected 'real' samples
            X_real, y_real = generate_real_samples(dataset, half_batch)
            # update discriminator model weights
            d_loss1, _ = d_model.train_on_batch(X_real, y_real)
            # generate 'fake' examples
            X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
            # update discriminator model weights
            d_loss2, _ = d_model.train_on_batch(X_fake, y_fake)
            # prepare points in latent space as input for the generator
            X_gan = generate_latent_points(latent_dim, n_batch)
            # create inverted labels for the fake samples
            y_gan = ones((n_batch, 1))
            # update the generator via the discriminator's error
            g_loss = gan_model.train_on_batch(X_gan, y_gan)
            # summarize loss on this batch
            print('>%d, %d/%d, d1=%.3f, d2=%.3f g=%.3f' %
                  (i+1, j+1, bat_per_epo, d_loss1, d_loss2, g_loss))
    # evaluate the model performance, sometimes
```

آموزش شبکه GAN

برای آموزش در هر تکرار^{۱۷} به تعداد نیمه از داده های برداشت شده را از داده گان اصلی برای آموزش شبکه پیچشی حقیقت یاب استفاده خواهیم کرد و تلف آن را در قالب d_loss1 نمایش می دهیم نیمه دیگر را عکس جعلی است توسط شبکه ی مولد تولید کرده و مجدداً برای آموزش حقیقت یاب استفاده می کنیم که تابع هزینه این قسمت را به عنوان d_loss2 نشان می دهیم. حال برای آموزش مولد به تعداد کل تیکه های داده بردار تصادفی تولید کرده و مقدار خروجی مورد انتظار را یک قرار میدهم چرا که انتظار داریم داده های تولید مولد^{۱۸} واقعا در حد داده های اصلی دیتاست باشند و حال کل مدل به هم پیوسته را با این داده ها آموزش می دهیم.

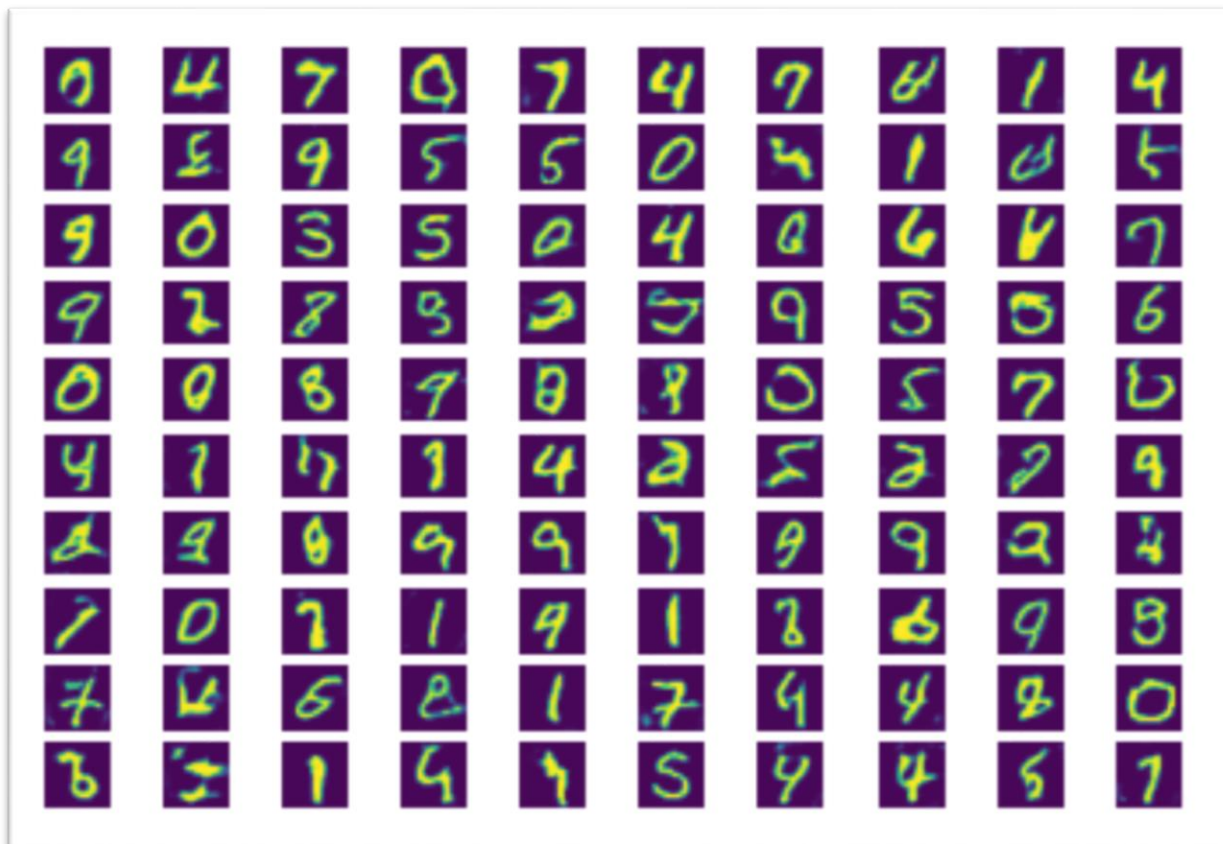
>3,	3/468,	d1=0.718,	d2=0.677	g=0.963
>3,	4/468,	d1=0.759,	d2=0.632	g=0.922
>3,	5/468,	d1=0.695,	d2=0.700	g=0.873
>3,	6/468,	d1=0.639,	d2=0.673	g=0.913
>3,	7/468,	d1=0.694,	d2=0.647	g=0.893
>3,	8/468,	d1=0.678,	d2=0.674	g=0.887
>3,	9/468,	d1=0.667,	d2=0.691	g=0.890
>3,	10/468,	d1=0.619,	d2=0.663	g=0.900
>3,	11/468,	d1=0.647,	d2=0.676	g=0.894
>3,	12/468,	d1=0.604,	d2=0.701	g=0.889
>3,	13/468,	d1=0.602,	d2=0.662	g=0.960
>3,	14/468,	d1=0.696,	d2=0.621	g=0.990
>3,	15/468,	d1=0.686,	d2=0.618	g=0.982
>3,	16/468,	d1=0.660,	d2=0.654	g=0.989
>3,	17/468,	d1=0.715,	d2=0.650	g=0.901
>3,	18/468,	d1=0.672,	d2=0.668	g=0.896
>3,	19/468,	d1=0.654,	d2=0.698	g=0.888
>3,	20/468,	d1=0.640,	d2=0.689	g=0.877
>3,	21/468,	d1=0.632,	d2=0.719	g=0.872
>3,	22/468,	d1=0.647,	d2=0.674	g=0.830
>3,	23/468,	d1=0.612,	d2=0.698	g=0.893
>3,	24/468,	d1=0.660,	d2=0.659	g=0.926
>3,	25/468,	d1=0.669,	d2=0.700	g=0.964
>3,	26/468,	d1=0.770,	d2=0.639	g=0.942
>3,	27/468,	d1=0.703,	d2=0.664	g=0.903
>3,	28/468,	d1=0.684,	d2=0.696	g=0.876
>3,	29/468,	d1=0.645,	d2=0.633	g=0.869
>3,	30/468,	d1=0.657,	d2=0.674	g=0.878
>3,	31/468,	d1=0.693,	d2=0.703	g=0.875
>3,	32/468,	d1=0.676,	d2=0.711	g=0.873
>3,	33/468,	d1=0.705,	d2=0.682	g=0.874
>3,	34/468,	d1=0.630,	d2=0.687	g=0.906
>3,	35/468,	d1=0.622,	d2=0.638	g=0.948
>3,	36/468,	d1=0.715,	d2=0.578	g=0.950

بخشی از توابع هزینه ها در هنگام آموزش

¹⁷ epoch

¹⁸ Generator

با انجام پروسه آموزش تا بیست تکرار به نتایج نسبتاً رضایت بخشی رسیدیم.



100 نمونه تصویر تولید شده توسط شبکه GAN

همانطور که مشاهده می‌نمایید کیفیت عکس‌های تولیدی بطور قابل ملاحظه‌ای افزایش یافته است.

بخش دوم: آنالیز معنایی متن

1.2 توضیحات اولیه مسئله

آنالیز معنایی متن^{۱۹}. در این مسئله ما با یک مجموعه داده^{۲۰} واقعی مواجه هستیم که شامل تعداد جمله به همراه یک برچسب است. جملات، بازخورد هایی^{۲۱} هستند که توسط مشتریان سه سایت آمازون^{۲۲}، آی‌ام‌دی‌بی^{۲۳} و یِلپ^{۲۴} در مورد یک فیلم، محصول و یا رستوران نوشته شده اند. برچسبها^{۲۵} هم یک عدد 0 یا 1 هستند که 0 نشان دهنده بازخورد منفی و 1 نشان دهنده بازخورد مثبت است. هدف ما این است که به کمک شبکه های عصبی^{۲۶}، به تحلیل مثبت/منفی بودن بازخورد بپردازیم؛ یعنی به کمک این شبکه، دسته بندی متن را به صورت نظارت شده^{۲۷} انجام دهیم.

روشی که برای حل مسائل متنی با کمک شبکه های عصبی مورد استفاده قرار می گیرد، یادگیری انتقالی^{۲۸} است. اساس کار این روش یادگیری این است که در آموزش یک شبکه جدید، از داده هایی استفاده نماییم که قبلاً توسط شبکه های دیگری بدست آمده اند. در این مسئله ما از مجموعه داده گِلُو^{۲۹} استفاده کردیم که توسط دانشگاه استنفورد تهیه شده است. گِلُو خود یک الگوریتم یادگیری بدون نظارت^{۳۰} است که روی دادگان بزرگی اجرا شده و کلمات را به یک فضای د-بعدی برده که در آن فضا فاصله کلمات مرتبط به هم کم است و هر کلمه با یک بردار د-عضوی توصیف می شود. در مجموعه داده هایی که بر روی پایگاه داده قرار گرفته است فایل هایی با مقادیر مختلف d موجود است: 50, 100, 200, 300. پس از بررسی هایی که انجام شد ما به این نتیجه رسیدیم که در صورت استفاده از داده های 300 بعدی می توانیم به دقت بهتری در طبقه بندی داده های دست پیدا کنیم که در ادامه به این موضوع پرداخته خواهد شد.

از آن جایی که رایانه زبان طبیعی نگارش و محاوره را نمی فهمد ما باید راهکاری این زبان طبیعی را به اعدادی تبدیل کنیم که برای رایانه قابل فهم شود. این کار با استفاده از بردارهای موجود در داده گان گِلُو که پیش تر در مورد آنها گفته شد انجام می شود. در ادامه ایده و راه حل به کار رفته در حل این مسئله شرح داده می شود.

همانطور که در بالا اشاره شد ما به کمک دادگان گِلُو می توانیم هر کلمه را به یک بردار د-بعدی از اعداد تبدیل کنیم بنابراین اگر جملاتی که در مجموعه داده بازخوردها موجود است را ابتدا به کلمات تفکیک کرده و سپس به جای هر کلمه بردار

¹⁹ Text sentiment analysis

²⁰ Dataset

²¹ Feedbacks

²² Amazon

²³ imdb

²⁴ Yelp

²⁵ Labels

²⁶ Neural networks

²⁷ Supervised

²⁸ Transfer learning

²⁹ Global vector (GloVe)

³⁰ Unsupervised learning

د-بعدی اش را قرار دهیم یک ماتریس 3 بعدی خواهیم داشت. ایده ای که در ابتدا مطرح می شود این است که این ماتریس را به یک بردار تک بعدی تبدیل کنیم و آن را به یک شبکه عصبی تماماً متصل³¹ بدهیم. اما شاید راهکار بهتری هم وجود داشته باشد. از آنجایی که ساختار توصیف شده برای جملات شباهت زیادی با ساختار یک تصویر دارد، می توان یک جمله را مانند یک تصویر در نظر گرفت که طول آن برابر تعداد کلمات جمله، عرض آن برابر 1 و همچنین عمق آن برابر تعداد اعضای برداری است که بیانگر هر کلمه است (برای مثال 300 * 1 * طول جمله). اکنون ایده ای که مطرح می شود این است که مانند طبقه بندی در روش آنالیز تصاویر، برای طبقه بندی متون هم از شبکه عصبی پیچشی³² استفاده کنیم. این کار یک برتری مهم دیگر هم نسبت به استفاده از شبکه تماماً متصل دارد و آن اهمیت پیدا کردن محل نسبی کلمات در جمله است؛ به عبارت بهتر در شبکه تمام متصل ما صرفاً می توانیم ویژگی های مربوط به وجود و عدم وجود کلمات مختلف در جمله را بررسی کنیم، در حالی که شبکه های عصبی پیچشی دارای هسته هایی³³ هستند که وظیفه آن ها استخراج اطلاعات از ورودی بدون تغییر ساختار آن است و در نتیجه به ما کمک می کند که به خوبی محل قرارگیری کلمات در هر جمله نسبت به هم و اینکه چه کلماتی در کنار هم به کار رفته اند را تحلیل کنیم. در ادامه به توضیح بخش های مختلف پیاده سازی این شبکه می پردازیم.

tupe می دهیم و در بخش توضیح شبکه های عصبی پیچشی به درباره هر تابع توضیح داده خواهد شد.

1. لغت نامه³⁴: در این قسمت فایل فشرده گلو را دریافت می کنیم که شامل کلمات و بردار هایشان هستند. در واقع این قسمت بیانگر یک مترجم است که هر کلمه را به برداری از اعداد ترجمه می کند. این لغت نامه شامل 4 فایل با عمق های 300، 200، 100، 50 است که ما برای بهبود عملکرد از عمق 300 استفاده کرده ایم.
2. خواندن جملات: در این قسمت هم فایل مجموعه داده بازخوردها را دریافت می کنیم. سپس هر جمله را از برچسبش جدا کرده و در یک تاپل³⁵ دوتای ذخیره می کنیم. همه این تاپل ها را در یک بردار نگهداری می کنیم.
3. جداسازی داده ها برای آموزش، ارزیابی و اعتبارسنجی³⁶: ابتدا 15٪ از داده ها را برای آزمون و بقیه را برای آموزش قرار می دهیم. همچنین 10٪ از داده آموزش را هم برای اعتبارسنجی جدا می کنیم.
4. آماده سازی جملات متنی برای ورودی به شبکه عصبی: همانطور که پیشتر نیز ذکر شد، برای تحلیل جملات متنی باید ابتدا آن ها را به ماتریسی از اعداد تبدیل کرد.

در نتیجه اولین اقدام، تفکیک همه جملات به کلماتشان است. علاوه بر این باید توجه کنیم که چون علائم نگارشی در معنای جمله اثرگذار هستند باید آنها را هم به عنوان یک کلمه در نظر بگیریم.

قدم بعد تبدیل کلمات تفکیک شده به بردار اعداد به کمک لغت نامه است.

نکته حائز اهمیت این است که در ادامه باید هر یک از این جملات تفکیک شده را به ورودی شبکه عصبی بدهیم و واضح است که ابعاد ورودی شبکه مقدار مشخص است پس باید همه جملات دارای ابعاد یکسان باشند. این شرط در

³¹ Fully connected

³² Convolutional

³³ Kernels

³⁴ Dictionary

³⁵ Tuple

³⁶ Validation

رابطه با عرض و عمق جملات صادق است ولی از آنجایی که طول خطوط یکسان نیست باید روشی برای یکسان کردن طول همه جملات پیدا کنیم. با یک بررسی کوتاه از جملات موجود در مجموعه داده، واضح است که طول اکثر جملات کمتر یا نهایتاً مساوی با 50 کلمه است بنابراین ابتدا این محدودیت را اعمال می کنیم که اندازه هیچ جمله ای نباید بیشتر از 50 کلمه باشد و اگر بود، فقط 50 کلمه ابتدایی آن را در بردار نهایی قرار می دهیم. حال نوبت به افزایش طول جملات کوتاه تر از 50 کلمه می رسد راهکاری که در صورت پروژه پیشنهاد شده است پر کردن این فاصله با لایه گذاری صفر³⁷ است. اما مسئله مهم دیگر این است که در موقع یکنواخت سازی³⁸ داده ها، برای استفاده شبکه عصبی، ممکن است این افزایش طول با کمک لایه گذاری صفر مشکل ساز شود پس بهتر است روش های دیگری راه هم امتحان کنیم.

روش هایی که برای افزایش طول جملات کوتاه استفاده شد:

1. محاسبه میانگین و انحراف معیار³⁹ داده ها روی هر بعد (میانگین و انحراف معیار هر کدام بردارهایی با طول 300 می شوند) و سپس یکنواخت سازی و در نهایت افزودن صفر به انتهای جملات
 2. افزودن صفر به انتهای جملات و سپس محاسبه میانگین و انحراف معیار و یکنواخت سازی
 3. تکرار کلمات هر جمله تا رسیدن به طول مناسب و سپس محاسبه میانگین و انحراف معیار و یکنواخت سازی. اساس این روش این است که با تکرار جمله تغییری در معنای آن ایجاد نمی شود.⁴⁰
- پیاده سازی روش اول دشوار است و پس از بررسی های صورت گرفته، از نظر دقت و هزینه رویداد های ارزیابی برتری چندانی ندارد. روش سوم هم باعث می شود تا طول جملات که ممکن است در معنای جمله، به عنوان یک ویژگی، اثرگذار باشد از بین می رود. ما از روش دوم در حل مسئله استفاده کردیم.
5. تعریف ساختار شبکه: برای حل مسئله باید ساختاری را برای شبکه تعریف کنیم. ما از ساختار ساده زیر استفاده کردیم:
 - یک لایه شامل دو فیلتر دو بعدی پیچشی با هسته های 3 در 1 و عمق 64، و یک لایه ادغامی⁴¹ با اندازه 2 در 1 است
 - پس این لایه، یک لایه صاف کننده⁴² و دو لایه تماماً متصل 128 نورونی
 - بین فیلترهای پیچشی و لایه های تماماً متصل از تابع غیرخطی رلو⁴³ استفاده می کنیم
 - در آخر برای تعیین طبقه از یک لایه سافت ماکس⁴⁴ استفاده می کنیم.

³⁷ Zero padding

³⁸ Normalization

³⁹ Standard deviation

⁴⁰ Wrap padding

⁴¹ Max pooling

⁴² Flatten

⁴³ Relu

⁴⁴ Softmax

6. آموزش شبکه: بعد از آماده شدن داده های ورودی ، آن ها را وارد شبکه عصبی کرده و وزن ها و سوگیری ها⁴⁵ برای لایه ها را آموزش می دهیم. روش آموزش به این صورت است که ابتدا در مسیر پیشرو مقادیر نهایی را محاسبه می کنیم و سپس در مسیر عقبگرد تغییرات وزن ها و سوگیری ها را بدست آورده و با استفاده از آنها و یک تابع حل کننده شبکه را به روز رسانی می کنیم.
7. در نهایت پس از اجرای چند تکرار⁴⁶ خروجی بدست آمده را روی داده های ارزیابی می آزمایشیم و نتایج را نمایش می دهیم.

1.3 بررسی نتایج بدست آمده

بررسی تاثیر نرخ یادگیری

نرخ یادگیری⁴⁷ را در چهار مقدار مختلف برای مشخصات زیر بررسی کردیم:

- تابع حل کننده: آدام⁴⁸
- بتا 1: 0.9
- بتا 2: 0.999⁴⁹
- معماری شبکه: فیلتر دو بعدی با هسته 3 در 1 و عمق 64 ، یک لایه ادغامی با ابعاد 2 در 1 ، دو لایه تماما متصل با تعداد نورون به ترتیب 1600 و 128 ، تابع فعالساز غیرخطی رلو
- اندازه بسته ها: 100
- تعداد تکرار: 8

نرخ یادگیری 0.1: به سرعت واگرا می شود.

نرخ یادگیری 0.01:

⁴⁵ Bias

⁴⁶ Epoch

⁴⁷ Learning rate

⁴⁸ ADAM

⁴⁹ این مقادیر از مقاله Kingma, D, Ba, J, Adam: a method for stochastic optimization برداشت شده است.

```

LR:0.01, Batch Size:100
Cost: 0.409259: 100%|██████████| 22/22 [10:02<00:00, 27.37s/it]
73.39055793991416
Cost: 0.379228: 100%|██████████| 22/22 [10:01<00:00, 27.32s/it]
75.9656652360515
Cost: 0.745437: 100%|██████████| 22/22 [09:55<00:00, 27.06s/it]
72.1030042918455
Cost: 0.060200: 100%|██████████| 22/22 [09:51<00:00, 26.88s/it]
78.11158798283262
Cost: 0.125045: 100%|██████████| 22/22 [09:52<00:00, 26.92s/it]
73.39055793991416
Cost: 0.273563: 100%|██████████| 22/22 [10:00<00:00, 27.31s/it]
78.11158798283262
Cost: 0.796329: 100%|██████████| 22/22 [09:59<00:00, 27.23s/it]
78.54077253218884
Cost: 1.670754: 100%|██████████| 22/22 [09:53<00:00, 26.97s/it]
78.96995708154506

```

شکل (22) جزئیات زمان اجرا



شکل (23) نحوه تغییرات دقت روی داده های آموزش و اعتبارسنجی و همچنین هزینه

دقت روی داده های ارزیابی: 78.16 %

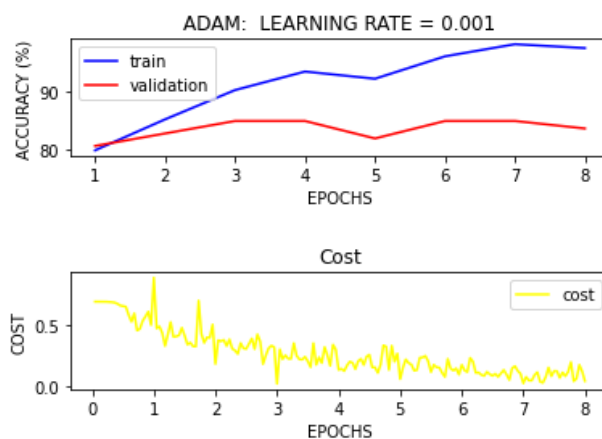
نرخ یادگیری 0.001 :

```

LR:0.001, Batch Size:100
Cost: 0.889309: 100%|██████████| 22/22 [10:02<00:00, 27.37s/it]
80.68669527896995
Cost: 0.184416: 100%|██████████| 22/22 [09:58<00:00, 27.20s/it]
82.83261802575107
Cost: 0.022044: 100%|██████████| 22/22 [09:58<00:00, 27.19s/it]
84.97854077253218
Cost: 0.136340: 100%|██████████| 22/22 [10:01<00:00, 27.32s/it]
84.97854077253218
Cost: 0.060064: 100%|██████████| 22/22 [09:58<00:00, 27.20s/it]
81.97424892703863
Cost: 0.176335: 100%|██████████| 22/22 [09:58<00:00, 27.22s/it]
84.97854077253218
Cost: 0.025915: 100%|██████████| 22/22 [10:04<00:00, 27.46s/it]
84.97854077253218

```

شکل (24) جزئیات زمان اجرا



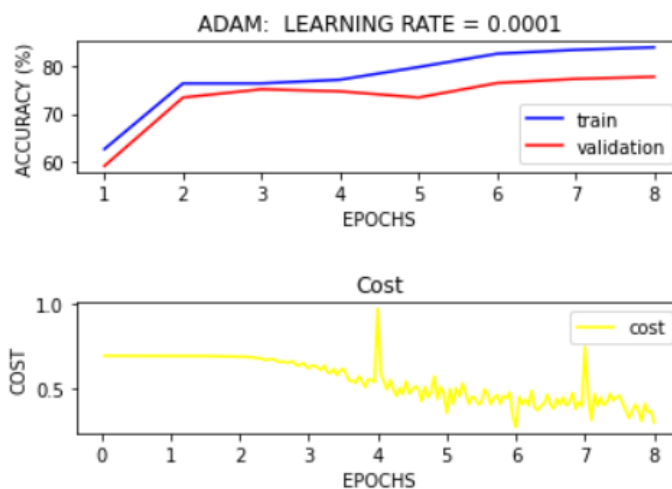
شکل (25) نحوه تغییرات دقت روی داده های آموزش و اعتبارسنجی و همچنین هزینه

دقت روی داده های ارزیابی: 85.19 %

نرخ یادگیری 0.0001 :

```
LR:0.0001, Batch Size:100
Cost: 0.692654: 100%|██████████| 22/22 [09:47<00:00, 26.70s/it]
59.227467811158796
Cost: 0.687766: 100%|██████████| 22/22 [09:47<00:00, 26.72s/it]
73.39055793991416
Cost: 0.618368: 100%|██████████| 22/22 [09:43<00:00, 26.53s/it]
75.10729613733905
Cost: 0.972799: 100%|██████████| 22/22 [09:39<00:00, 26.35s/it]
74.67811158798283
Cost: 0.354071: 100%|██████████| 22/22 [09:39<00:00, 26.35s/it]
73.39055793991416
Cost: 0.271476: 100%|██████████| 22/22 [09:39<00:00, 26.32s/it]
76.39484978540773
Cost: 0.746175: 100%|██████████| 22/22 [09:39<00:00, 26.33s/it]
77.25321888412017
Cost: 0.299600: 100%|██████████| 22/22 [09:38<00:00, 26.30s/it]
77.6824034334764
```

شکل () جزئیات زمان اجرا



شکل (26) نحوه تغییرات دقت روی داده های آموزش و اعتبارسنجی و همچنین هزینه

دقت روی داده های ارزیابی: 83.25 %

همانطور که از نتایج بدست آمده در برمی آید، نرخ یادگیری 0.1 بسیار بزرگ است و به سرعت واگرا می شود، نرخ یادگیری 0.01 هم به اندازه کافی کوچک نیست و به یک دقت نه چندان خوب همگرا می شود. نرخ یادگیری 0.0001 هم بیش از اندازه کوچک است و سرعت بسیار کمی دارد. نرخ یادگیری 0.001 از سه نرخ دیگر دقت بالاتری را روی داده های ارزیابی ثبت کرد و سرعت خوبی هم داشت بنابراین این نرخ را به عنوان نرخ یادگیری مناسب انتخاب کرده و برای قسمت های بعدی هم از این نرخ استفاده می کنیم. (در مقاله یادشده هم 0.001 برای نرخ یادگیری الگوریتم آدام توصیه شده است).

بررسی دو روش بهینه سازی مختلف در محاسبه گرادیان

در قسمت قبلی الگوریتم حل کننده آدام بود، در این قسمت به بررسی الگوریتم گرادیان تصادفی کاهشی با تکانه^{۵۰} می پردازیم. با توجه به اهمیت نرخ یادگیری در همگرایی این روش، ابتدا از نرخ 0.001 که در قسمت قبل بدست آورده بودیم استفاده می کنیم. از همان مشخصات قسمت قبل استفاده می کنیم و پارامتر گاما^{۵۱} که مختص همین الگوریتم است، 0.9 قرار می دهیم.^{۵۲}

⁵⁰ Stochastic gradient descent with momentum

⁵¹ Gamma

⁵² طبق پیشنهاد مقاله Ruder, S, An overview of gradient descent optimization algorithms

تئوری الگوریتم آدام:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1 - \beta_1) \nabla_w L^{(t)}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1 - \beta_2) (\nabla_w L^{(t)})^2$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1 - \beta_1}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1 - \beta_2}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

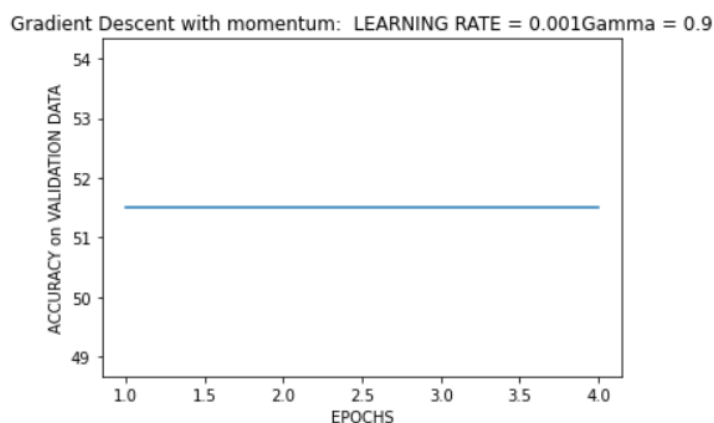
پارامترهای m ، v ، \hat{m} و \hat{v} مربوط به الگوریتم بوده و در ابتدا برابر صفر مقدار دهی می‌شوند. بالانویس t هم شماره تکرار را نشان می‌دهد.

تئوری الگوریتم تکانه:

$$w := w - \alpha \nabla Q_i(w) + \gamma \Delta w$$

ایده الگوریتم این است که برای به‌روزرسانی وزن‌ها از تغییرات قبلی با یک ضریب استفاده کنیم.

نتایج اجرای الگوریتم به صورت زیر است:



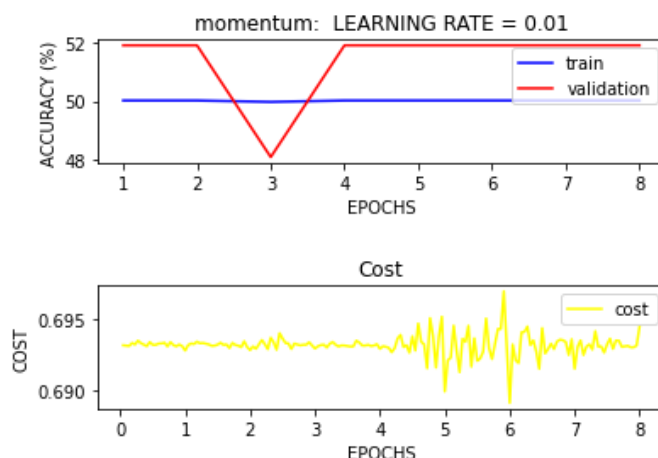
شکل (27) حوه تغییرات دقت روی داده های اعتبارسنجی

دقت روی داده های تست: 48.54 %

از روی نتایج بالا واضح است که الگوریتم با این نرخ یادگیری سرعت بسیار کمی دارد بنابراین باید نرخ یادگیری را افزایش دهیم. این بار نرخ یادگیری را 0.01 قرار می دهیم.

```
LR:0.01, Batch Size:100
Cost: 0.692789: 100%|██████████| 22/22 [09:29<00:00, 25.89s/it]
51.931330472103
Cost: 0.692826: 100%|██████████| 22/22 [09:30<00:00, 25.95s/it]
51.931330472103
Cost: 0.692928: 100%|██████████| 22/22 [09:29<00:00, 25.87s/it]
48.06866952789699
Cost: 0.692907: 100%|██████████| 22/22 [09:26<00:00, 25.75s/it]
51.931330472103
Cost: 0.689969: 100%|██████████| 22/22 [09:26<00:00, 25.74s/it]
51.931330472103
Cost: 0.689174: 100%|██████████| 22/22 [09:28<00:00, 25.84s/it]
51.931330472103
Cost: 0.691536: 100%|██████████| 22/22 [09:26<00:00, 25.73s/it]
51.931330472103
Cost: 0.694432: 100%|██████████| 22/22 [09:26<00:00, 25.77s/it]
51.931330472103
```

شکل (28) جزئیات زمان اجرا



شکل (29) نحوه تغییرات دقت روی داده های آموزش و اعتبارسنجی و همچنین هزینه

دقت روی داده های ارزیابی: 51.7 %

همانطور که واضح است دقت اندکی افزایش یافته است اما باز هم قابل مقایسه با دقت بسیار بالای الگوریتم آدام نیست. بنابراین الگوریتم مناسب برای حل این مسئله، آدام است و در قسمت های بعدی از آن استفاده می کنیم.

اثر تغییر تعداد لایه‌های میانی و تعداد نورون‌ها

در این قسمت قصد داریم تا اندکی معماری شبکه را تغییر دهیم. در ابتدا تاثیر تعداد نورون‌های لایه‌های تماماً متصل را بررسی می‌کنیم و در ادامه تعداد لایه‌های شبکه پیچشی را افزایش می‌دهیم و اثر آن را هم بررسی می‌کنیم.

- تاثیر تغییر تعداد نورون‌ها

حالت اول

دو فیلتر دو بعدی با هسته 3 در 1 و عمق 64، یک لایه ادغامی با ابعاد 2 در 1

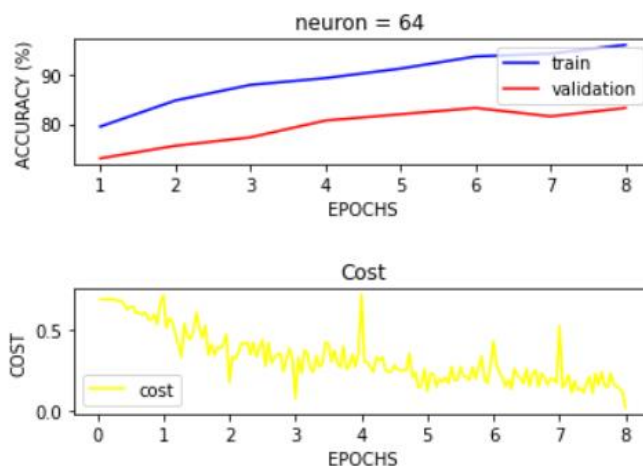
دو لایه تماماً متصل با تعداد نورون به ترتیب 1600 و 64

تابع فعالساز غیرخطی رلو

نتایج به صورت زیر است:

```
LR:0.001, Batch Size:100
Cost: 0.717913: 100%|██████████| 22/22 [10:22<00:00, 28.28s/it]
72.96137339055794
Cost: 0.174768: 100%|██████████| 22/22 [10:20<00:00, 28.23s/it]
75.53648068669528
Cost: 0.071974: 100%|██████████| 22/22 [10:19<00:00, 28.17s/it]
77.25321888412017
Cost: 0.722991: 100%|██████████| 22/22 [10:29<00:00, 28.61s/it]
80.68669527896995
Cost: 0.120475: 100%|██████████| 22/22 [10:33<00:00, 28.78s/it]
81.97424892703863
Cost: 0.428749: 100%|██████████| 22/22 [10:26<00:00, 28.49s/it]
83.2618025751073
Cost: 0.526849: 100%|██████████| 22/22 [10:25<00:00, 28.42s/it]
81.54506437768241
Cost: 0.012877: 100%|██████████| 22/22 [10:36<00:00, 28.93s/it]
83.2618025751073
```

شکل (30) جزئیات زمان اجرا



شکل (31) نحوه تغییرات دقت روی داده های آموزش و اعتبارسنجی و همچنین هزینه

دقت روی داده های ارزیابی: 85.19 %.

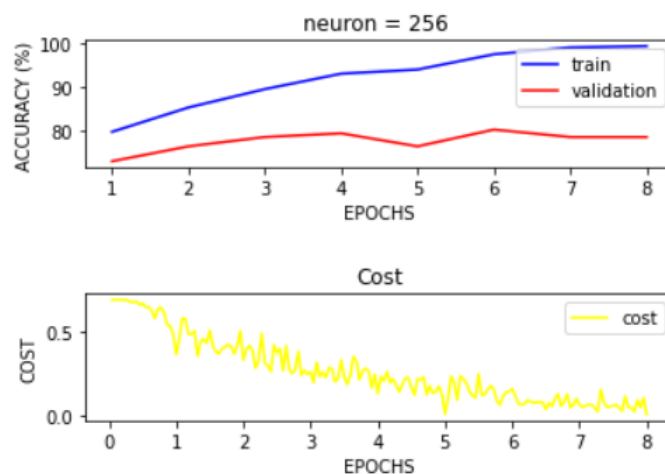
حالت دوم

دو فیلتر دو بعدی با هسته 3 در 1 و عمق 64 ، یک لایه ادغامی با ابعاد 2 در 1
دو لایه تماماً متصل با تعداد نورون به ترتیب 1600 و 256
تابع فعالساز غیرخطی رلو

نتایج به صورت زیر است:

```
LR:0.001, Batch Size:100
Cost: 0.363636: 100%|██████████| 22/22 [09:54<00:00, 27.02s/it]
72.96137339055794
Cost: 0.330262: 100%|██████████| 22/22 [09:55<00:00, 27.07s/it]
76.39484978540773
Cost: 0.259214: 100%|██████████| 22/22 [10:00<00:00, 27.32s/it]
78.54077253218884
Cost: 0.128648: 100%|██████████| 22/22 [09:53<00:00, 26.98s/it]
79.39914163090128
Cost: 0.006223: 100%|██████████| 22/22 [09:54<00:00, 27.03s/it]
76.39484978540773
Cost: 0.157072: 100%|██████████| 22/22 [09:54<00:00, 27.01s/it]
80.25751072961373
Cost: 0.062973: 100%|██████████| 22/22 [10:08<00:00, 27.65s/it]
78.54077253218884
Cost: 0.002026: 100%|██████████| 22/22 [09:57<00:00, 27.16s/it]
78.54077253218884
```

شکل (32) جزئیات زمان اجرا



شکل (33) نحوه تغییرات دقت روی داده های آموزش و اعتبارسنجی و همچنین هزینه

دقت روی داده های ارزیابی: 85.68 %

حالت سوم

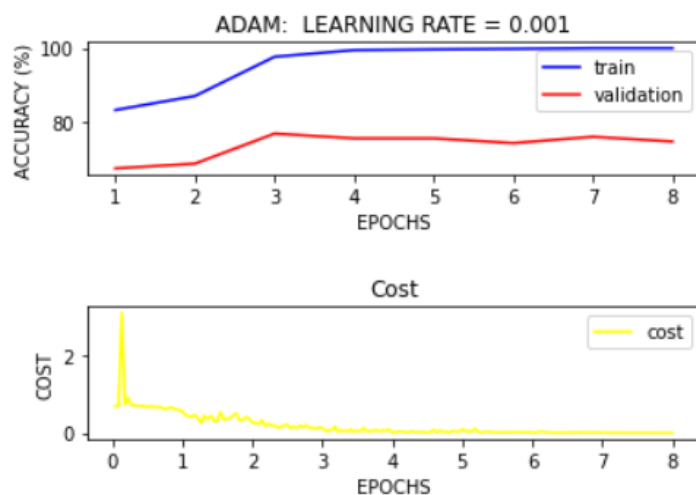
سه لایه تماماً متصل با تعداد نورون به ترتیب 15000 و 1024 و 1024

تابع فعالساز غیرخطی رلو

نتایج به صورت زیر است:

```
LR:0.001, Batch Size:100
Cost: 0.564814: 100%|██████████| 22/22 [03:29<00:00, 9.52s/it]
67.38197424892704
Cost: 0.262745: 100%|██████████| 22/22 [03:31<00:00, 9.59s/it]
68.6695278969957
Cost: 0.126920: 100%|██████████| 22/22 [03:28<00:00, 9.49s/it]
76.82403433476395
Cost: 0.009405: 100%|██████████| 22/22 [03:26<00:00, 9.37s/it]
75.53648068669528
Cost: 0.089783: 100%|██████████| 22/22 [03:24<00:00, 9.31s/it]
75.53648068669528
Cost: 0.000892: 100%|██████████| 22/22 [03:23<00:00, 9.25s/it]
74.2489270386266
Cost: 0.000075: 100%|██████████| 22/22 [03:23<00:00, 9.23s/it]
75.9656652360515
Cost: 0.000123: 100%|██████████| 22/22 [03:23<00:00, 9.26s/it]
74.67811158798283
```

شکل (34) جزئیات زمان اجرا



شکل (35) نحوه تغییرات دقت روی داده های آموزش و اعتبارسنجی و همچنین هزینه

دقت روی داده های ارزیابی: 73.79٪

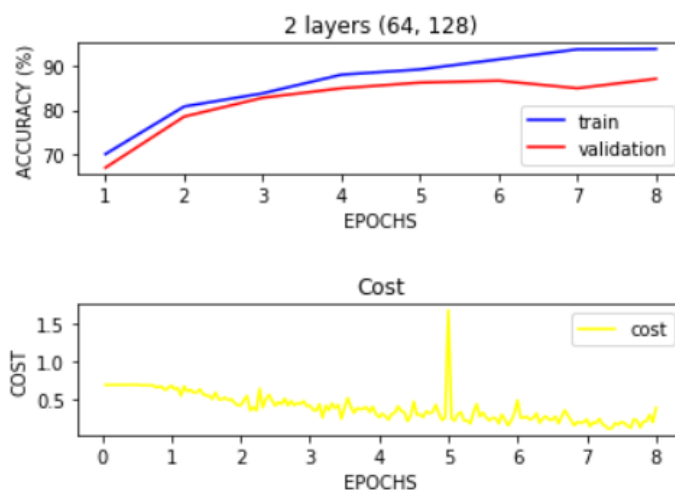
- تاثیر تغییر تعداد لایه ها

- دو فیلتر دو بعدی با هسته 3 در 1 و عمق 64 ، یک لایه ادغامی با ابعاد 2 در 1
- دو فیلتر دو بعدی با هسته 3 در 1 و عمق 128 ، یک لایه ادغامی با ابعاد 2 در 1
- دو لایه تماما متصل با تعداد نورون به ترتیب 1664 و 128
- تابع فعالساز غیرخطی رلو

نتایج به صورت زیر است:

```
LR:0.001, Batch Size:100
Cost: 0.683981: 100%|██████████| 22/22 [16:48<00:00, 45.84s/it]
66.95278969957081
Cost: 0.421220: 100%|██████████| 22/22 [16:41<00:00, 45.51s/it]
78.54077253218884
Cost: 0.422884: 100%|██████████| 22/22 [16:28<00:00, 44.94s/it]
82.83261802575107
Cost: 0.268456: 100%|██████████| 22/22 [16:27<00:00, 44.90s/it]
84.97854077253218
Cost: 1.678896: 100%|██████████| 22/22 [16:30<00:00, 45.01s/it]
86.26609442060087
Cost: 0.489896: 100%|██████████| 22/22 [16:29<00:00, 44.97s/it]
86.69527896995707
Cost: 0.240530: 100%|██████████| 22/22 [16:32<00:00, 45.13s/it]
84.97854077253218
Cost: 0.389782: 100%|██████████| 22/22 [16:40<00:00, 45.46s/it]
87.1244635193133
```

شکل (36) جزئیات زمان اجرا



شکل (37) نحوه تغییرات دقت روی داده های آموزش و اعتبارسنجی و همچنین هزینه

دقت روی داده های ارزیابی: 83.74 %.

تحلیل:

با توجه به اینکه پایین ترین دقت مربوط به شبکه بدون لایه پیچشی است نتیجه می گیریم که انتخاب ما در استفاده از لایه های پیچشی برای حل این مسئله ، انتخاب درستی بوده است. همچنین به نظر می رسد که افزودن نورون بیشتر باعث افزایش چشمگیری در دقت مدل نشده است که احتمالاً به دلیل بیش برآزش^{۵۳} باشد. پس شاید افزودن نورون بیشتر از 64 ضروری نبوده باشد و با توجه به دقتی که بر روی داده ها داشته است انتخاب به صرفه از نظر زمان آموزش برای یک لایه بوده است. افزودن تعداد لایه ها تاثیر مطلوبی بر روی افزایش دقت داده های اعتبارسنجی داشته و دقت داده های اعتبارسنجی را به داده های آموزش نزدیک کرده است. ولی به نظر می رسد که با توجه به افزایش زمان آموزش ، همان یک لایه کافی بوده باشد.

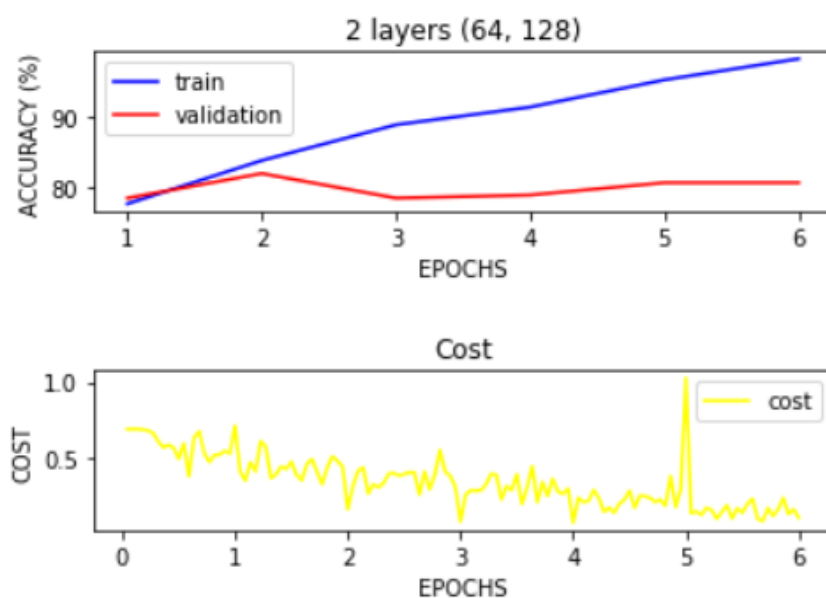
⁵³ Over fitting

بررسی دو تابع فعالساز خطی و غیرخطی

در قسمت پیش به بررسی تابع فعالسازی غیر خطی با شبکه ذکر شده پرداختیم. در این قسمت از تابع فعالساز خطی $y = 1x + 0$ استفاده می‌کنیم.

```
LR:0.001, Batch Size:100
Cost: 0.715054: 100%|██████████| 22/22 [09:36<00:00, 26.19s/it]
78.54077253218884
Cost: 0.169493: 100%|██████████| 22/22 [09:29<00:00, 25.87s/it]
81.97424892703863
Cost: 0.088887: 100%|██████████| 22/22 [09:27<00:00, 25.79s/it]
78.54077253218884
Cost: 0.080550: 100%|██████████| 22/22 [09:25<00:00, 25.71s/it]
78.96995708154506
Cost: 1.029080: 100%|██████████| 22/22 [09:25<00:00, 25.73s/it]
80.68669527896995
Cost: 0.113493: 100%|██████████| 22/22 [09:31<00:00, 25.99s/it]
80.68669527896995
```

شکل (38) جزئیات زمان اجرا



شکل (39) نحوه تغییرات دقت روی داده های آموزش و اعتبارسنجی و همچنین هزینه

در شکل بالا وقوع بیش‌برازش کاملاً مشخص است. دقت روی داده‌های ارزیابی: 48.82٪.

همانطور که انتظار داشتیم، تابع فعالساز خطی نسبت به تابع فعالساز غیرخطی عملکرد ضعیف‌تری داشته است که این موضوع از دقت روی داده های آموزش و همچنین نمودار دقت برای داده های آموزش و صحت سنجی مشهود است.

بخش سوم: شبکه پیچشی

همانطور که قبل تر نیز اشاره شد هسته اصلی هردو بخش از این پروژه متعلق است به طراحی شبکه پیچشی^{۵۴} لذا بر آن شدیم که بخش مجزایی را به آن اختصاص دهیم که در آن درمورد تمامی توابع موجوده نوشته شده توضیح دهیم توجه شود که در این بخش تمامی توابع از پایه و بدون استفاده از هرگونه کتابخانه ای پیاده سازی شده اند توجه کنید که این توابع تماما بصورت پارامتری نوشته شده اند و این امکان را بما می دهند که هر شبکه ای را با تعداد لایه های دلخواه و همچنین پارامترهای مفروض تست نماییم. در ادامه ابتدا پایه اساسی کدهای این شبکه ضمیمه خواهد شد و در انتها نیز توضیح مختصری درمورد هریک از توابع مورد استفاده ضمیمه خواهیم کرد:

```
: import zipfile, csv, os, io, string, pickle, math
import numpy as np
import matplotlib.pyplot as plt
import urllib.request as urllib
import pandas as pd
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
from numpy.random import randn
import random
```

```
: df_train= pd.read_csv("mnist_train.csv")
df_test = pd.read_csv("mnist_test.csv")
```

```
: data_train=df_train.to_numpy()
data_test=df_test.to_numpy()
```

```
TEST_RATIO = 0.15
VALIDATION_RATIO = 0.1
MAX_DIM = 1
MAX_LENGTH = 28
NUM_OF_CLASSES = 10
LR = 0.001
LR_DECAY = 0.95
BETA1 = 0.9
BETA2 = 0.999
NUM_OF_EPOCHS = 4
BATCH_SIZE = 100
SAVE_PATH = 'params.pkl'
```

```
X_train = data_train[:,1:]
y_train = data_train[:,0]
X_test = data_test[:,1:]
y_test = data_test[:,0]
```

⁵⁴ Convolutional Neural Network

Some Functions

```
def function_builder(name, alpha=1, bias=0):
    def func(X):
        if name == 'Relu':
            X[X<=0] = 0
        elif name == 'LeakyRelu':
            X = np.max(alpha * X, X)
        elif name == 'ELU':
            X = alpha*(np.exp(X[X<=0])-1)
        elif name == 'Tanh':
            X = np.tanh(X)
        elif name == 'Linear':
            X = alpha*X + bias
        return X
    return func
```

```
def initializeFilter(size, scale = 1.0):
    stddev = scale/np.sqrt(np.prod(size))
    return np.random.normal(loc = 0, scale = stddev, size = size)

def initializeWeight(size):
    return np.random.standard_normal(size=size) * 0.01

def initializeParams():
    m = [None]*len(Network)
    v = [None]*len(Network)
    mb = [None]*len(Network)
    vb = [None]*len(Network)

    for i, layer in enumerate(Network):
        if layer[0] == 'maxpool' or layer[0] == 'flatten':
            continue
        m[i] = np.zeros(layer[1].shape)
        v[i] = np.zeros(layer[1].shape)
        mb[i] = np.zeros(layer[2].shape)
        vb[i] = np.zeros(layer[2].shape)

    return [m, v, mb, vb]
```

```
def categoricalCrossEntropy(probs, label):
    return -np.sum(label * np.log(probs))
```

```
def nanargmax(arr):
    idx = np.nanargmax(arr)
    idxs = np.unravel_index(idx, arr.shape)
    return idxs
```

```
def calc_accuracy (X_test,Y_true):
    Y_pred = np.empty((np.shape(Y_true)))
    Y_probs = np.empty((np.shape(Y_true)))
    for i in range(len(X_test)):
        Y_pred[i], Y_probs[i] = predict(X_test[i])
    Y_pred = np.array(Y_pred).astype(int)
    return np.sum((Y_pred==Y_true).astype(int))/len(Y_true)*100, Y_pred
```

Layer Builders

```
def convolution(data, weights, bias, stride):
    filter_x, filter_y, filter_depth, filter_num = np.shape(weights)
    data_x, data_y, data_depth = np.shape(data)
    result = np.zeros((data_x, data_y, filter_num))
    pad_size_x = filter_x // 2
    pad_size_y = filter_y // 2
    padded_data = np.pad(data, ((pad_size_x, pad_size_x), (pad_size_y, pad_size_y), (0, 0)), 'constant', constant_values=0)
    for n in range(filter_num):
        kernel_dy = pad_size_y
        result_y = 0
        while kernel_dy < data_y:
            kernel_dx = pad_size_x
            result_x = 0
            while kernel_dx < data_x:
                result[result_x, result_y, n] = np.sum(np.multiply(weights[:, :, :, n], padded_data[kernel_dx:kernel_dx+filter_x,
                kernel_dy:kernel_dy+filter_y, :]), axis=(0, 1))
                kernel_dx += stride
                result_x += 1
            kernel_dy += stride
            result_y += 1
    return result
```

```
def maxpool(data, pool_f): #pool_f is tuple (2, 1), pool_stride
    (data_x, data_y, data_depth) = np.shape(data)
    dim_x = math.ceil(data_x / pool_f[0])
    dim_y = math.ceil(data_y / pool_f[1])
    downsampled = np.zeros((dim_x, dim_y, data_depth))
    pad_size_x = pool_f[0] - data_x % pool_f[0]
    pad_size_y = pool_f[1] - data_y % pool_f[1]
    padded_data = np.pad(data, ((0, pad_size_x), (0, pad_size_y), (0, 0)), 'constant', constant_values=0)
    for i in range(data_depth):
        kernel_dy = 0
        result_y = 0
        while kernel_dy + pool_f[1] < np.shape(padded_data)[1]:
            kernel_dx = 0
            result_x = 0
            while kernel_dx + pool_f[0] < np.shape(padded_data)[0]:
                downsampled[result_x, result_y, i] = np.max(padded_data[kernel_dx:kernel_dx+pool_f[0], kernel_dy:kernel_dy+pool_f[1], i])
                kernel_dx += pool_f[0]
                result_x += 1
            kernel_dy += pool_f[1]
            result_y += 1
    return downsampled
```

```
def convolutionBackward(dconv_prev, conv_in, filt, s):

    (f_dim_x, f_dim_y, f_depth, num_f) = filt.shape
    (orig_dim_x, orig_dim_y, _) = conv_in.shape

    dout = np.zeros(conv_in.shape)
    dfilt = np.zeros(filt.shape)
    dbias = np.zeros((num_f,1))

    for curr_f in range(num_f):

        curr_y = out_y = 0
        while curr_y + f_dim_y <= orig_dim_y:
            curr_x = out_x = 0
            while curr_x + f_dim_x <= orig_dim_x:
                dfilt[:, :, :, curr_f] += dconv_prev[out_x, out_y, curr_f] * conv_in[curr_x:curr_x + f_dim_x, curr_y:curr_y + f_dim_y, :]
                dout[curr_x:curr_x + f_dim_x, curr_y:curr_y + f_dim_y, :] += dconv_prev[out_x, out_y, curr_f] * filt[:, :, :, curr_f]
                curr_x += s
                out_x += 1
            curr_y += s
            out_y += 1

        dbias[curr_f] = np.sum(dconv_prev[:, :, :, curr_f])

    return dout, dfilt, dbias
```

```
def softmax(raw_preds):
    out = np.exp(raw_preds)
    return out/np.sum(out)
```

Design Network

```
# conv, maxpool, fullyconn, flatten, softmax
Network = []
Network.append(['conv', initializeFilter((5, 5, 1, 8)), np.zeros((8,1)), 'Relu']) #####
Network.append(['conv', initializeFilter((5, 5, 8, 8)), np.zeros((8,1)), 'Relu'])
Network.append(['maxpool', (2, 2)])

##Network.append(['conv', initializeFilter((5, 5, 8, 16)), np.zeros((16,1)), 'Relu']) #####
#Network.append(['conv', initializeFilter((5, 5, 16, 16)), np.zeros((16,1)), 'Relu'])
#Network.append(['maxpool', (2, 2)])

#Network.append(['conv', initializeFilter((5, 5, 16, 32)), np.zeros((32,1)), 'Relu']) #####
#Network.append(['conv', initializeFilter((5, 5, 32, 32)), np.zeros((32,1)), 'Relu'])
#Network.append(['maxpool', (2, 2)])

# Network.append(['conv', initializeFilter((3, 1, 64, 32)), np.zeros((32,1)), 'Relu'])
Network.append(['flatten',])

Network.append(['fullyconn', initializeWeight((200, 1568)), np.zeros((200,1)), 'Relu'])
Network.append(['fullyconn', initializeWeight((10, 200)), np.zeros((10,1)), 'Softmax'])
```

```
data_dim_x = MAX_LENGTH
data_dim_y = 1
data_depth = MAX_DIM
f_dim1 = 3
f_dim2 = 1
num_filt1 = 64
num_filt2 = 64
num_neurons = 128
flatten_size = 1600
```


Implementing Network

```
def CNN_forward (data):
    results = [None]*(len(Network)+1)
    results[0] = data
    for i, layer in enumerate(Network):
        if (layer[0] == 'conv'):
            results[i+1] = convolution(results[i], layer[1], layer[2], 1)
            results[i+1] = function_builder(layer[3])(results[i+1])
        elif (layer[0] == 'maxpool'):
            results[i+1] = maxpool(results[i], layer[1])
        elif (layer[0] == 'flatten'):
            dim1, dim2, dim3 = np.shape(results[i])
            results[i+1] = results[i].reshape((dim1*dim2*dim3, 1))
        elif (layer[0] == 'fullyconn'):
            results[i+1] = layer[1].dot(results[i]) + layer[2]
            if layer[3] == 'Softmax':
                results[i+1] = softmax(results[i+1])
                break
            results[i+1] = function_builder(layer[3])(results[i+1])
    return results
```

```
def CNN_backward (results, label):
    dNetwork = [[None, None]]*len(Network)
    dresults = [None]*(len(Network)+1)
    dresults[-1] = results[-1] - label
    for i, layer in reversed(list(enumerate(Network))):
        if (layer[0] == 'conv'):
            dresults[i], df, db = convolutionBackward(dresults[i+1], results[i], layer[1], 1)
            dNetwork[i] = [df, db]
            if i != 0 and Network[i-1][0] == 'conv':
                dresults[i][results[i]<=0] = 0
        elif (layer[0] == 'maxpool'):
            dresults[i] = maxpoolBackward(dresults[i+1], results[i], layer[1])
            dresults[i][results[i]<=0] = 0
        elif (layer[0] == 'flatten'):
            dresults[i] = dresults[i+1].reshape(np.shape(results[i]))
        elif (layer[0] == 'fullyconn'):
            dw = dresults[i+1].dot(results[i].T)
            db = np.sum(dresults[i+1], axis=1).reshape(layer[2].shape)
            dNetwork[i] = [dw, db]
            dresults[i] = layer[1].T.dot(dresults[i+1])
            if Network[i-1][0] == 'flatten':
                continue
            dresults[i][results[i]<=0] = 0
    return dNetwork
```

```
def CNN (data, label):
    results = CNN_forward (data)
    dNetwork = CNN_backward(results, label)

    loss = categoricalCrossEntropy(results[-1], label)
    grads = dNetwork

    return grads, loss
```

Solver

```
def momentumGD(dNetwork, lr, params):
    [m, v, mb, vb] = params
    for i, layer in enumerate(Network):
        if layer[0] == 'maxpool' or layer[0] == 'flatten':
            continue
        v[i] = GAMMA*v[i] + lr*(dNetwork[i][0]/BATCH_SIZE)
        layer[1] -= v[i]

        vb[i] = GAMMA*vb[i] + lr*(dNetwork[i][1]/BATCH_SIZE)
        layer[2] -= vb[i]

    return [m, v, mb, vb]
```

```
def adamGD (dNetwork, lr, params):
    [m, v, mb, vb] = params
    for i, layer in enumerate(Network):
        if layer[0] == 'maxpool' or layer[0] == 'flatten':
            continue
        m[i] = BETA1*m[i] + (1-BETA1)*dNetwork[i][0]/BATCH_SIZE
        v[i] = BETA2*v[i] + (1-BETA2)*(dNetwork[i][0]/BATCH_SIZE)**2
        m_hat = m[i]/(1-BETA1)
        v_hat = v[i]/(1-BETA2)
        layer[1] -= lr * m_hat/(np.sqrt(v_hat)+1e-7)

        mb[i] = BETA1*mb[i] + (1-BETA1)*dNetwork[i][1]/BATCH_SIZE
        vb[i] = BETA2*vb[i] + (1-BETA2)*(dNetwork[i][1]/BATCH_SIZE)**2
        mb_hat = mb[i]/(1-BETA1)
        vb_hat = vb[i]/(1-BETA2)
        layer[2] -= lr * mb_hat/(np.sqrt(vb_hat)+1e-7)

    return [m, v, mb, vb]
```

```
def solver(batch, lr, params, cost, optimizer='ADAM'):
    |
    X = batch[:,0:-1]
    Y = batch[:,1]

    X = X.reshape(len(batch),MAX_LENGTH,MAX_LENGTH,MAX_DIM)

    cost_ = 0

    dNetwork = [[0, 0]]*len(Network)

    for i in range(len(X)):
        x = X[i]
        y = np.eye(NUM_OF_CLASSES)[int(Y[i])].reshape(NUM_OF_CLASSES, 1)

        grads, loss = CNN(x, y)
        for j, grad in enumerate(grads):
            if grad[0] is None:
                continue
            if i == 0:
                dNetwork[j] = [np.zeros(grad[0].shape), np.zeros(grad[1].shape)]
            dNetwork[j][0] += grad[0]
            dNetwork[j][1] += grad[1]
        cost_ += loss

    if optimizer == 'ADAM':
        new_params = adamGD(dNetwork, lr, params)
    elif optimizer == 'MOMENTUM':
        new_params = momentumGD(dNetwork, lr, params)

    cost_ = cost_/BATCH_SIZE
    cost.append(cost_)

    return new_params, cost
```

```
params = initializeParams()

m = 5000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
X=(X-mean)/std
trainData = np.hstack((X,y_dash))
cost = []
print("LR:"+str(LR)+" , Batch Size:"+str(BATCH_SIZE))
for epoch in range(NUM_OF_EPOCHS):
    np.random.shuffle(trainData)
    batches = np.array([trainData[k:k + BATCH_SIZE] for k in range(0, trainData.shape[0], BATCH_SIZE)])
    t = tqdm(batches)
    for x,batch in enumerate(t):

        params, cost = solver(batch, LR*(LR_DECAY**x), params, cost, 'ADAM')
        t.set_description("Cost: %.6f" % (cost[-1]))
with open(SAVE_PATH, 'wb') as file:
    pickle.dump(Network, file)
```

Loading Parameters from Directory

```
Network = pickle.load(open('Network.plk', 'rb'))
```

Test

```
m = 1000
X = X_train[0:m,:]
len(X)
y_dash = y_train[0:m].reshape(m,1)
mean=int(np.mean(X))
std= int(np.std(X))
#####
m = 200
X = X_test[0:m,:]
y_dash = y_test[0:m]
X=(X-mean)/std
X = X.reshape(len(X),28,28,1)

acc, y_pred = calc_accuracy(X,y_dash)
```

در ادامه توضیحات مربوط به هریک از توابع به پیوست خواهد آمد:

1. تابع `function builder`: در این قسمت تابع های فعالساز مختلف خطی و غیر خطی تعریف شده اند.
2. تابع برای مقدار دهی اولیه: در این قسمت 3 تابع مختلف برای مقدار دهی اولیه وزن ها و اعداد ثابت برای هسته های لایه های پیچشی و نورون ها و همچنین مقدار دهی اولیه پارامتر های توابعی مانند تابع گرادیان نزولی به همراه مومنتوم و همچنین تابع آدام که به کمک آن ها مقادیر وزن ها و اعداد ثابت آن ها به روز رسانی می شوند.
3. تابع `categoricalCrossEntropy`: این تابع هزینه به کمک احتمالات خروجی برای هر کلاس و همچنین با در اختیار داشتن برچسب هر داده، مقدار هزینه (loss) را برای هر داده محاسبه می کند.
4. تابع `nanargmax`: این تابع `index` بزرگترین مقدار غیر `NAN` یک آرایه را بر میگرداند
5. تابع `calc_accuracy`: با توجه به اینکه شبکه عصبی به صورت سراسری (global) تعریف شده است، این تابع با در اختیار داشتن یک مجموعه داده به همراه برچسب آن، دقت شبکه را خروجی می دهد.
6. تابع `convolution`: این تابع با در اختیار داشتن داده ورودی و همچنین وزن ها و اعداد ثابت برای هر هسته پیچشی و همچنین طول گام آن، خروجی پیچشی (convolution) را محاسبه می کند. عمق هسته در این تابع باید با عمق ورودی یکسان باشد. این تابع با حرکت دادن هسته بر روی داده، خروجی پیچشی آن را محاسبه می کند.
7. تابع `maxpool`: عامل ادغام بر اساس بیشینه معمولاً بعد از لایه پیچشی اعمال می شود و با در اختیار داشتن مقدار ثابت از بین ورودی های مجاور هم، بیشینه آن ها را انتخاب می کند. در واقع این تابع مانند یک فیلتر عمل می کند که باید با حرکت بر روی ورودی، بیشینه مقدار را از چندین سلول کنار هم انتخاب کند.
8. تابع `convolutionBackward`: عمل `back propagation` را برای لایه های `conv2D` انجام می دهد
9. تابع `maxpoolBackward`: عمل `back propagation` را برای لایه های `maxpool` انجام می دهد

10. تابع softmax: این تابع با در اختیار داشتن مقادیر احتمال هر کلاس برای یک داده، آن ها را به صورت یک تابع نمایی نرمال می کند.
11. تابع CNN_forward: مقادیر ورودی را به صورت sequential در لایه های مختلف مدل وارد کرده و عمل forwarding را انجام میدهد
12. تابع CNN_Backward: با استفاده از توابع convolutionBackward و maxpoolBackward عمل back propagation را در طول تمام لایه ها انجام می دهد.
13. تابع CNN: این تابع شامل دو تابع CNN_forward و CNN_Backward است؛ یعنی با ورود هر جمله کد شده به این تابع، نتایج محاسبات تا انتها محاسبه شده و انتشار می یابد، مقدار هزینه به کمک تابع categoricalCrossEntropy محاسبه می شود.
14. تابع momentumGD: gradient descent را به روش momentum انجام می دهد.
15. تابع adamGD: gradient descent را با روش adam انجام می دهد.
16. تابع solver: با گرفتن پارامتر ها و lr و نوع solver عمل بروزرسانی پارامتر های شبکه را انجام می دهد.
17. تابع predic: در این تابع از تابع CNN_forward استفاده شده است؛ یعنی داده تست از ورودی به خروجی انتقال داده می شود و مقادیر خروجی شبکه محاسبه می شود.

نکات انتهایی:

درانتها از دکتر رشاد حسینی استاد محترم درس و سازماندهی کننده محترم دوره جناب آقای پسند کمال قدردانی را خواهیم داشت و همچنین از آقایان محمد حسین وعیدی، محمدرضا توکلی، سید محمد متین آل محمد و محمد حیدری تشکر ویژه میکنیم که بدون همکاری آنها به پایان رساندن این پروژه بی معنی می بود. همچنین تلاش آقای سالار نوری در راستای طراحی پروژه با بالاترین کیفیت ممکن و استانداردهای آموزشی قابل ستایش خواهد بود.

ارجاعات:

- [1] <https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/>
- [2] <https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcd8a0f>
- [3] <https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/>
- [4] <https://quick-adviser.com/how-do-i-load-a-cifar-10-dataset/>
- [5] <https://www.geeksforgeeks.org/python-randint-function/>
- [6] <https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/>
- [7] <https://towardsdatascience.com/gans-from-scratch-8f5da17b3fb4>
- [8] <https://www.kaggle.com/oddrational/mnist-in-csv/metadata>
- [9] <https://towardsdatascience.com/the-end-to-all-blurry-pictures-f27e49f23588>