**Môn học: Machine Learning**

**Giảng viên:** TS. Lê Thành Sách

**Bài tập lớn số 1** : Bản hiện thực mạng CNN với numpy

*Sinh viên:*

*Nguyễn Hoàng Lộc - MSSV: 1511849*

*Nguyễn Hữu Anh Tiến - MSSV: 1513448*

**Mô tả bài tập lớn:** Nhóm hiện thực một mạng CNN với nhiều layer khác nhau, dựa trên mô hình cấu trúc của thư viện Keras, gồm:

* **Model chính:** class Sequential quản lí việc forward dữ liệu và backward đạo hàm quua list các layers
* **Các layers:** Gồm các loại như Convolution2D, MaxPooling2D, Dropout, Flatten, Dense (Fully Connected)
* **Các hàm Activation:** Gồm hàm Relu, Sigmoid và Softmax (được chứa trong mỗi layer)
* **Hàm Loss :** Cross Entropy (được chứa trong class Sequential)

**Install, import & setup**[**¶**](#gjdgxs)

In [0]:

**import** **numpy** **as** **np**  
**import** **math**  
**import** **numpy**   
**import** **struct**  
**import** **sys**  
**import** **time**  
**import** **keras**  
**from** **sklearn.metrics** **import** accuracy\_score  
**import** **matplotlib.pyplot** **as** **plt**  
plt.style.use('seaborn-whitegrid')  
eps = np.finfo(float).eps  
**from** **google.colab** **import** drive

Using TensorFlow backend.

In [0]:

drive.mount('/content/drive')  
sys.path.append('drive/My Drive/Colab Notebooks/')

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client\_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect\_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response\_type=code  
  
Enter your authorization code:  
··········  
Mounted at /content/drive

**Load data, format & visualize**[**¶**](#30j0zll)

In [0]:

train\_images\_path = 'drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/data/train-images'  
train\_labels\_path = 'drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/data/train-labels'  
test\_images\_path = 'drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/data/t10k-images'  
test\_label\_path = 'drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/data/t10k-labels'  
  
  
**def** read\_idx(filename):  
 **with** open(filename, 'rb') **as** f:  
 zero, data\_type, dims = struct.unpack('>HBB', f.read(4))  
 shape = tuple(struct.unpack('>I', f.read(4))[0] **for** d **in** range(dims))   
 **return** numpy.fromstring(f.read(), dtype=np.uint8).reshape(shape)  
  
**def** load\_mnist\_data():  
 train\_data = [read\_idx(train\_images\_path), read\_idx(train\_labels\_path)]  
 test\_data = [read\_idx(test\_images\_path), read\_idx(test\_label\_path)]  
 **return** train\_data, test\_data

In [0]:

train\_data, test\_data = load\_mnist\_data()  
X\_train = train\_data[0]  
y\_train = train\_data[1]  
X\_test = test\_data[0]  
y\_test = test\_data[1]  
  
X\_train = np.array(X\_train)  
y\_train = np.array(y\_train)  
X\_test = np.array(X\_test)  
y\_test = np.array(y\_test)

/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:11: DeprecationWarning: The binary mode of fromstring is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instead  
 # This is added back by InteractiveShellApp.init\_path()

In [0]:

fig = plt.figure()  
**for** i **in** range(9):  
 plt.subplot(3,3,i+1)  
 plt.tight\_layout()  
 plt.imshow(X\_train[i], cmap='gray', interpolation='none')  
 plt.title("Digit: **{}**".format(y\_train[i]))  
 plt.xticks([])  
 plt.yticks([])

In [0]:

img\_rows = 28  
img\_cols = 28  
*#Chuyển ảnh về format channel first*  
X\_train\_norm = X\_train.reshape(X\_train.shape[0], 1, img\_rows, img\_cols)  
X\_test\_norm = X\_test.reshape(X\_test.shape[0], 1, img\_rows, img\_cols)  
input\_shape = (1, img\_rows, img\_cols)  
  
*#Chuyển ảnh về dạng channel last*  
*# X\_train = X\_train.reshape(X\_train.shape[0], img\_rows, img\_cols, 1)*  
*# X\_test = X\_test.reshape(X\_test.shape[0], img\_rows, img\_cols, 1)*  
*# input\_shape = (img\_rows, img\_cols, 1)*  
  
X\_train\_norm = X\_train\_norm.astype('float32')  
X\_test\_norm = X\_test\_norm.astype('float32')  
X\_train\_norm /= 255  
X\_test\_norm /= 255  
  
*# X\_train\_crop = np.array([x[0, 12:16, 12:16] for x in X\_train\_norm])*  
*# X\_test\_crop = np.array([x[0, 12:16, 12:16] for x in X\_test\_norm])*  
*# X\_train\_crop = X\_train\_crop.reshape(60000, 1, 4, 4)*  
*#X\_test\_crop = X\_test\_crop.reshape(X\_test.shape[0], 1, img\_rows, img\_cols)*  
  
print('X\_train shape:', X\_train.shape)

X\_train shape: (60000, 28, 28)

In [0]:

*#Chuyển tập target về dạng one-hot*  
**def** get\_one\_hot(targets, nb\_classes):  
 res = np.eye(nb\_classes)[np.array(targets).reshape(-1)]  
 **return** res.reshape(list(targets.shape)+[nb\_classes])  
   
num\_category = 10 *#ứng với 10 label từ 0->9*  
y\_train\_norm = get\_one\_hot(y\_train, num\_category)  
y\_test\_norm = get\_one\_hot(y\_test, num\_category)  
print('y\_train shape:', y\_train\_norm.shape)

y\_train shape: (60000, 10)

**--------------------CNN - numpy inplementation--------------------**[**¶**](#1fob9te)

**Model - Sequential**[**¶**](#3znysh7)

Class Sequential đóng vai trò nắm giữ các layers, quản lí việc forward, backward

Có hiện thực các hàm evaluate, hỗ trợ predict và predict\_class

In [0]:

*#Class đóng vai trò là model, quản lí việc forward và backward cho các layers*  
**class** **Sequential**:   
 **def** \_\_init\_\_(self):  
 self.layer\_list = list()  
   
 **def** add(self, new\_layer):  
 self.layer\_list.append(new\_layer)  
   
 **def** compile(self, loss = 'cross\_entropy'):  
 **if** loss == 'cross\_entropy':  
 self.loss = CrossEntropy()  
 **else**:  
 **pass** *#Có thể hiện thực thêm nhiều hàm lỗi khác*  
   
 **def** fit(self, X\_train, y\_train, batch\_size = 128, num\_epoch = 10, validation\_data = **None**):  
 self.X\_train = X\_train  
 self.y\_train = y\_train  
   
 train\_loss\_list = list()  
 val\_loss\_list = list()  
 epoch\_time\_list = list()  
  
 **for** epoch **in** range(num\_epoch):  
 epoch\_loss = list()  
 **for** num\_batch **in** range(X\_train.shape[0] // batch\_size):  
 current\_index = num\_batch\*batch\_size   
 image\_batch = X\_train[current\_index : current\_index + batch\_size] *#Lấy các ảnh ứng với epoch*  
 target\_batch = y\_train[current\_index : current\_index + batch\_size] *#Lấy các target ứng với các ảnh*  
  
 **for** index, image **in** enumerate(image\_batch):  
 output = self.feed\_forward(image) *#Feed từng ảnh qua mạng*  
 self.loss.feed\_forward(output, target\_batch[index]) *#Add ouput của layer cuối cùng vào hàm loss*  
  
 batch\_avg\_loss = self.loss.get\_avg\_loss(batch\_size)  
 epoch\_loss.append(batch\_avg\_loss)  
*# avg\_grad = self.loss.backward(batch\_size)*  
 avg\_grad = target\_batch[index]  
 self.backward(avg\_grad)  
 self.loss.reset()  
   
 epoch\_avg\_loss = sum(epoch\_loss) / len(epoch\_loss)  
  
 train\_loss\_list.append(epoch\_avg\_loss)  
 epoch\_time\_list.append(epoch)  
   
 **if** validation\_data **is** **not** **None**:  
 val\_loss = self.evaluate(validation\_data[0], validation\_data[1])  
 val\_loss\_list.append(val\_loss)  
 print("Epoch ", epoch,"- Training loss: ", np.round(epoch\_avg\_loss, 14), " - Validating loss: ", val\_loss)  
 **else**:  
 print("Epoch ", epoch,"- Training Loss: ", epoch\_avg\_loss)   
   
 plt.plot(epoch\_time\_list, train\_loss\_list)  
 plt.plot(epoch\_time\_list, val\_loss\_list)  
 plt.legend(('Train', 'Val'))  
 plt.xlabel('Epoch:')  
 plt.ylabel('Loss: ')  
 plt.show(block=**False**)  
   
   
 **def** feed\_forward(self, data, isTrain = **True**):  
 layer\_input = data  
 **for** layer **in** self.layer\_list:  
*# print('Feedforward through layer: ',type(layer)) ##*  
 **if** **not** isTrain **and** type(layer) **is** Dropout: *#Biến isTrain để bypass layer Dropout*  
 **continue**  
 layer\_input = layer.feed\_forward(layer\_input)  
*# print("Output :", np.around(layer\_input, decimals = 4), layer\_input.shape) ##*  
  
 *#Sau khi feed qua tất cả layer thì biến layer\_input chứa output của layer cuối cùng*  
 **return** layer\_input   
   
 **def** backward(self, dY, isTrain = **True**):  
 layer\_grad = dY  
 **for** layer **in** reversed(self.layer\_list):  
*# print('Back-forward through layer: ', type(layer)) ##*  
 **if** isTrain == **False** **and** type(layer) **is** Dropout:  
 **continue**  
 layer\_grad = layer.backward(layer\_grad)  
*# print ("Output: ", np.around(layer\_grad, decimals = 4), layer\_grad.shape) ##*  
   
   
 **def** evaluate(self, X\_test, y\_test):  
 output\_list = list()  
 **for** index, image **in** enumerate(X\_test):  
 output = self.feed\_forward(image, isTrain = **False**)   
 output\_list.append(output)  
 self.loss.feed\_forward(output, y\_test[index])  
 avg\_loss = self.loss.get\_avg\_loss(len(y\_test))  
 self.loss.reset()  
 **return** avg\_loss  
   
 **def** predict(self, X\_test):  
 output\_list = list()  
 **for** index, image **in** enumerate(X\_test):  
 output = self.feed\_forward(image, isTrain = **False**)   
 output\_onehot = np.zeros\_like(output)  
 output\_onehot[output.argmax()] = 1  
 output\_list.append(output\_onehot)  
 **return** output\_list  
  
 **def** predict\_class(self, X\_test):  
 output\_list = self.predict(X\_test)  
 output\_list\_class = list()  
 **for** one\_hot **in** output\_list:  
 num\_class, = np.where(one\_hot== 1)[0]  
 output\_list\_class.append(num\_class)  
 **return** output\_list\_class

**Các hàm Activation**[**¶**](#2et92p0)

**ReLu**[**¶**](#tyjcwt)

In [0]:

**class** **ReLU**:  
 **def** \_\_init\_\_(self):  
 self.data = **None**   
   
 **def** feed\_forward(self, data):  
 data[data<0] = 0  
 data = data/ (np.amax(data) + eps)  
 self.data = data  
 **return** data  
   
 **def** backward(self, dY):  
 max\_data = np.amax(self.data)  
 self.data[self.data>0] = 1  
 **return** np.multiply(self.data, dY)\*max\_data

**Sigmoid**[**¶**](#3dy6vkm)

In [0]:

**class** **Sigmoid**:  
 **def** \_\_init\_\_(self):  
 self.data = **None**   
   
 **def** sigmoid(x):  
 **return** 1 / (1 + math.exp(-x))  
   
 **def** feed\_forward(self, data):  
 self.data = self.sigmoid(data)  
 **return** data  
   
 **def** backward(self, dY):  
 **return** np.multiply(np.multiply(self.data, 1- self.data), dY)

**Softmax**[**¶**](#1t3h5sf)

In [0]:

**class** **Softmax**():  
 **def** \_\_init\_\_(self):  
 self.cache = **None**  
   
 **def** feed\_forward(self, X):  
 self.cache = np.exp(X) / np.sum(np.exp(X), axis = 0)  
 **return** self.cache  
   
 **def** backward(self, dY):  
 **return** self.cache - dY

**Hàm Loss**[**¶**](#4d34og8)

In [0]:

**class** **CrossEntropy**():  
 **def** \_\_init\_\_(self):  
 self.reset()  
   
 **def** reset(self): *#Gọi sau mỗi batch*  
 self.gradient = **None** *#Chứa tổng gradient để backward cho mỗi batch*  
 self.total\_loss = **None** *#Chứa tổng loss của mỗi batch*  
   
 **def** feed\_forward(self, output, target):  
 temp = -target\*(1/(np.log(2)\*output))  
 **if** self.gradient **is** **None**:  
 self.gradient = temp  
 **else**:  
 self.gradient += temp   
   
 loss = np.sum(-1\*target \* np.log2(output + eps))  
 **if** self.total\_loss **is** **None**:  
 self.total\_loss = loss  
 **else**:   
 self.total\_loss += loss  
   
 **def** get\_avg\_loss(self, batch\_size):  
 **return** self.total\_loss / batch\_size  
   
 **def** backward(self, batch\_size):  
 **return** self.gradient/batch\_size

**Các Layer**[**¶**](#2s8eyo1)

**Convolution2D**[**¶**](#17dp8vu)

Class Convolution2D có khả năng nhận input nhiều channel, torng nội bộ class có chứa nhiều filter và hỗ trợ padding, stride

Có hiện thực cơ chế im2col và kernel2row để forward và backward dữ liệu

Có hỗ trợ lưu các trọng số thành file và có thể load để tiếp tục sử dụng

Input: width W, height H, Channels C, Stride S, Padding P

Filters: F\_width, F\_height, k filter

= > Output:

* (W - F\_width - 2\*P1) / S1
* (H - F\_height - 2\*P2) / S2
* Channel = k

In [0]:

**class** **Convolution2D**:   
 **def** \_\_init\_\_(self, num\_filter, filter\_size, input\_shape, padding = 0, stride = 1, init\_weights = **None**, init\_bias = **None**, activation = 'relu', learning\_rate = 0.01):  
 self.num\_filter = num\_filter  
 self.filter\_size = filter\_size  
 self.input\_shape = input\_shape  
 self.weights = init\_weights  
*# self.bias = init\_bias*  
 self.padding = padding  
 self.stride = stride  
 **if** activation == 'relu':  
 self.activation = ReLU()  
 **else**:  
 **pass** *#Có thể hiện thực nhiều hàm activation khác*  
 self.lrn\_rate = learning\_rate  
   
 **if** self.weights **is** **None**:  
 rand\_number = math.sqrt(6/(input\_shape[0]+num\_filter))  
 self.weights = np.random.uniform(low = -rand\_number, high = rand\_number, size = (num\_filter, input\_shape[0] ,filter\_size[0], filter\_size[1]))*#Channel first format: param 2 -> số channel input*   
*# if self.bias is None:*  
*# self.bias = np.zeros(num\_filter)*  
  
   
 **def** set\_input(self, data): *#Padding ảnh input với 0*  
 self.input = np.zeros((self.input\_shape[0], self.input\_shape[1] + self.padding\*2, self.input\_shape[2] + self.padding\*2))  
 **for** channel **in** range(self.input\_shape[0]):  
 self.input[channel] = np.pad(data[channel], pad\_width = self.padding, mode = 'constant', constant\_values = 0)  
   
 **def** feed\_forward(self, data):  
 self.set\_input(data)  
 self.convert\_X\_conv()  
 self.convert\_W\_conv()  
 self.convert\_W\_flat()  
 self.output = list()   
*# print ("Weights of Convolution forward", self.weights, self.weights.shape) ##*  
 **for** filter **in** range(self.num\_filter):  
 test = np.matmul(self.W\_flat\_list[filter], self.X\_conv)  
 self.output.append(test.reshape(self.o1, self.o2))   
 **return** self.activation.feed\_forward(np.array(self.output))  
  
 **def** backward(self, dY\_flat):  
 dY\_flat = self.activation.backward(dY\_flat)  
 dX = np.zeros((self.input\_shape[0], self.input.shape[1], self.input.shape[2]))  
 **for** filter **in** range(self.num\_filter):   
 dY\_flatten = dY\_flat[filter, :, :].flatten()  
 *#print(dY\_flatten.shape)*   
 dW = np.matmul(self.X\_conv, dY\_flatten).reshape(self.input\_shape[0], self.filter\_size[0], self.filter\_size[1])   
 dX += np.matmul(dY\_flatten.reshape(1, dY\_flatten.shape[0]) ,self.W\_conv\_list[filter]).reshape((self.input\_shape[0], self.input.shape[1], self.input.shape[2]))   
 self.weights[filter] -= dW\*self.lrn\_rate  
*# print ("Weights of Convolution backward", self.weights, self.weights.shape)##*  
 **return** dX  
   
   
 **def** convert\_W\_flat(self):  
 self.W\_flat\_list = list()  
 **for** filter **in** range(self.num\_filter):  
 W\_flat = **None**  
 **for** channel **in** range(self.input\_shape[0]):  
 temp = self.weights[filter, channel, :, :].flatten()  
 **if** W\_flat **is** **None**:  
 W\_flat = temp  
 **else**:  
 W\_flat = np.concatenate((W\_flat, temp), axis = 0)  
 self.W\_flat\_list.append(W\_flat)  
  
   
 **def** convert\_X\_conv(self):  
 *#im2col cho các channel của input*  
 self.X\_conv = **None**  
 **for** channel **in** range(self.input\_shape[0]):  
 temp = self.im2col(self.input[channel])  
 **if** self.X\_conv **is** **None**:  
 self.X\_conv = temp  
 **else**:  
 self.X\_conv = np.concatenate((self.X\_conv, temp), axis=0) *#Mở rộng chiều rows*  
   
   
 **def** convert\_W\_conv(self):  
 self.W\_conv\_list = list()  
 *#kernel2col cho tất cả các kernel (tất cả channel)*  
 **for** filter **in** range(self.num\_filter):  
 W\_conv = **None**  
 **for** channel **in** range(self.input\_shape[0]):  
 temp = self.kernel2row(self.weights[filter, channel,:,:])  
 **if** W\_conv **is** **None**:  
 W\_conv = temp  
 **else**:  
 W\_conv = np.concatenate((W\_conv, temp), axis=1) *#Ghép theo cột*  
 self.W\_conv\_list.append( W\_conv)  
  
   
 **def** im2col(self, image): *#Chuyển một ma trận 2 chiều (1 channel) về dạng X\_conv*  
 self.o1 = int((self.input.shape[1] - self.filter\_size[0])/self.stride) + 1 *#Số step dịch kernel theo row*  
 self.o2 = int((self.input.shape[2] - self.filter\_size[1])/self.stride) + 1 *#Số step dịch kernel theo column*   
 list\_conv = list()  
 **for** row **in** range(self.o1):  
 row\_start = row \* self.stride  
 row\_end = row\_start + self.filter\_size[0]  
 **for** col **in** range(self.o2):   
 col\_start = col\*self.stride  
 col\_end = col\_start + self.filter\_size[1]   
 temp = image[row\_start:row\_end, col\_start:col\_end]  
 list\_conv.append(np.flip(temp.flatten(),0))  
 X\_conv = np.transpose(np.array(list\_conv))  
 **return** X\_conv  
   
   
 **def** kernel2row(self, kernel): *#Chuyển một kernel về dạng W\_conv*  
 rot\_kernel = np.flip(kernel, 0)  
 rot\_kernel = np.flip(rot\_kernel, 1)  
 list\_w\_conv = list()  
 **for** row **in** range(self.o1):  
 row\_start = row \* self.stride  
 row\_end = row\_start + self.filter\_size[0]  
 **for** col **in** range(self.o2):   
 col\_start = col\*self.stride  
 col\_end = col\_start + self.filter\_size[1]   
 temp = np.zeros((self.input.shape[1], self.input.shape[2]))  
 temp[row\_start:row\_end, col\_start:col\_end] = rot\_kernel  
 list\_w\_conv.append(temp.flatten())  
 W\_conv = np.array(list\_w\_conv)   
 **return** W\_conv  
   
 **def** save\_weights(self, filename):  
 np.save('drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/saved\_model/' + filename + '.npy', self.weights)  
   
 **def** load\_weights(self, filename):  
 self.weights = np.load('drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/saved\_model/' + filename + '.npy')

**MaxPooling2D**[**¶**](#3rdcrjn)

In [0]:

**class** **MaxPooling2D**():  
 **def** \_\_init\_\_(self, pool\_size=(2, 2), stride=**None**):  
 self.X = **None**  
 self.pool\_indices = list()  
 self.pool\_size = pool\_size   
 **if** stride **is** **None**:  
 self.stride = pool\_size  
 **else**:  
 self.stride = stride  
   
 **def** feed\_forward(self, X):  
 D, W, H = X.shape   
 W\_pool = self.pool\_size[0]  
 H\_pool = self.pool\_size[1]   
 W\_S = self.stride[0]  
 H\_S = self.stride[1]   
 WW = 1 + (W - W\_pool)//W\_S  
 HH = 1 + (H - H\_pool)//H\_S  
 DD = D   
 out = np.zeros((DD, HH, WW))   
 **for** depth **in** range(D):  
 **for** row **in** range(HH):  
 **for** col **in** range(WW):  
 out[depth, row, col] = np.max(X[depth, row\*H\_S:row\*H\_S+H\_pool, col\*W\_S:col\*W\_S+W\_pool])   
 self.cache = X   
 **return** np.array(out)  
   
 **def** backward(self, out):  
 D, W, H = self.cache.shape  
 DD, WW, HH = out.shape   
 W\_pool = self.pool\_size[0]  
 H\_pool = self.pool\_size[1]   
 W\_S = self.stride[0]  
 H\_S = self.stride[1]   
 dX = np.zeros(self.cache.shape)  
   
 **for** depth **in** range(D):  
 **for** row **in** range(0, HH):  
 **for** col **in** range(0, WW):  
 x\_pool = self.cache[depth, row\*H\_S:row\*H\_S+H\_pool, col\*W\_S:col\*W\_S+W\_pool]   
 mask = (x\_pool == np.max(x\_pool))   
 dX[depth, row\*H\_S:row\*H\_S+H\_pool, col\*W\_S:col\*W\_S+W\_pool] = mask\* out[depth, row, col]  
 **return** dX

**Dropout**[**¶**](#26in1rg)

In [0]:

**class** **Dropout**():  
 **def** \_\_init\_\_(self, drop\_rate = 0.25):  
 self.drop\_rate = drop\_rate  
 self.data = **None**  
   
 **def** drop(self, element): *#Mỗi phần tử bị set 0 với xác suất là drop\_rate*  
 rand = np.random.ranf(1)  
 **if** rand < self.drop\_rate:  
 **return** 0  
 **else**:  
 **return** element/(1-self.drop\_rate)  
   
*# def feed\_forward(self, data):*  
*# vec = np.vectorize(self.drop) #Map tất cả các phần tử với hàm drop*  
*# self.data = vec(data)*  
*# return self.data*  
   
 **def** feed\_forward(self, data):  
 shape = data.shape  
 flat = data.flatten()  
 **for** i **in** range(len(flat)):  
 flat[i] = self.drop(flat[i])  
 self.data = flat.reshape(shape)  
 **return** self.data  
  
 **def** backward(self, dY):  
 self.data[self.data != 0] = 1  
 **return** np.multiply(self.data, dY)

**Flatten**[**¶**](#lnxbz9)

In [0]:

**class** **Flatten**():  
 **def** \_\_init\_\_(self):  
 self.shape = **None**  
   
 **def** feed\_forward(self, data):  
 self.shape = data.shape  
 **return** data.flatten()  
   
 **def** backward(self, dY):  
 **return** dY.reshape(self.shape)

**Dense**[**¶**](#35nkun2)

Lớp fully connected có hỗ trợ lưu và load lại trọng số từ file

In [0]:

**class** **Dense**():  
 **def** \_\_init\_\_(self, num\_units, init\_weights = **None**, init\_bias = **None**, activation = 'relu', learning\_rate = 0.01):  
 self.num\_units = num\_units  
 self.weights = init\_weights  
 self.bias = init\_bias  
 **if** activation == 'relu':  
 self.activation = ReLU()  
 **elif** activation == 'softmax':  
 self.activation = Softmax()  
 self.lrn\_rate = learning\_rate  
  
 **def** feed\_forward(self, data):  
 self.input = data  
 **if** self.weights **is** **None**:  
 rand\_number = math.sqrt(6/(self.num\_units+data.shape[0]))  
 self.weights = np.random.uniform( low = -rand\_number, high = rand\_number, size = (self.num\_units, data.shape[0]))  
 **if** self.bias **is** **None**:  
 rand\_number = math.sqrt(6/(self.num\_units+data.shape[0]))  
 self.bias = np.random.uniform(low = -rand\_number, high = rand\_number, size = self.num\_units)  
 A = np.matmul(self.weights, data) + self.bias  
 **return** self.activation.feed\_forward(A)  
   
 **def** backward(self, dY):  
 dY = self.activation.backward(dY)  
 dW = np.matmul(dY.reshape(dY.shape[0], 1), self.input.reshape(1, self.input.shape[0]))  
 dX = np.matmul(self.weights.T, dY)  
 self.weights -= dW\*self.lrn\_rate  
 self.bias -= dY\*self.lrn\_rate  
 **return** dX  
   
 **def** save\_weights(self, filename):  
 np.save('drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/saved\_model/' + filename + '.npy', self.weights)  
 np.save('drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/saved\_model/' + filename + '\_bias.npy', self.bias)  
   
 **def** load\_weights(self, filename):  
 self.weights = np.load('drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/saved\_model/' + filename + '.npy')  
 self.bias = np.load('drive/My Drive/Colab Notebooks/Asgmt\_ML\_CNN/saved\_model/' + filename + '\_bias.npy')

**--------------------Testing--------------------**[**¶**](#1ksv4uv)

Xây dựng model theo cấu trúc Conv2D - Conv2D - MaxPooling2D - Dropout - Flatten - Dropout - Dense - Dense

Train trên 10000 ảnh, qua 20 epochs với batch\_size = 20

In [0]:

num\_class = 10  
  
convol2D\_1 = Convolution2D(2, (3, 3), input\_shape=(1,28,28), learning\_rate = 0.05)  
convol2D\_2 = Convolution2D(4, (4, 4), input\_shape=(2,26,26), learning\_rate = 0.05)  
max\_pooling = MaxPooling2D(pool\_size=(2, 2))  
dropout\_1 = Dropout(0.1)  
flatten = Flatten()  
dropout\_2 = Dropout(0.25)  
dense\_1 = Dense(24, activation='relu',learning\_rate = 0.05)  
dense\_2 = Dense(num\_class, activation='softmax', learning\_rate = 0.05)

In [0]:

*# convol2D\_1.load\_weights('convol2D\_1')*  
*# convol2D\_2.load\_weights('convol2D\_2')*  
*# dense\_1.load\_weights('dense\_1')*  
*# dense\_2.load\_weights('dense\_2')*

In [0]:

model = Sequential()  
model.add(convol2D\_1)  
model.add(convol2D\_2)  
model.add(max\_pooling)  
model.add(dropout\_1)  
model.add(flatten)  
model.add(dropout\_2)  
model.add(dense\_1)  
model.add(dense\_2)  
  
model.compile(loss = 'cross\_entropy')  
  
batch\_size = 20  
epochs = 20  
  
model.fit(  
 X\_train\_norm[0:10000],   
 y\_train\_norm[0:10000],  
 batch\_size = batch\_size,  
 num\_epoch = epochs,  
 validation\_data = (X\_test\_norm[0:20], y\_test\_norm[0:20])  
)

Epoch 0 - Training loss: 2.92365631378023 - Validating loss: 2.889496777805659  
Epoch 1 - Training loss: 2.22761676848523 - Validating loss: 2.3044589611493778  
Epoch 2 - Training loss: 1.88353669942595 - Validating loss: 1.9360579906544593  
Epoch 3 - Training loss: 1.66652325945595 - Validating loss: 1.7180017805268606  
Epoch 4 - Training loss: 1.54414263431477 - Validating loss: 1.5891256835366783  
Epoch 5 - Training loss: 1.45903993536859 - Validating loss: 1.4759077819632918  
Epoch 6 - Training loss: 1.40161767819103 - Validating loss: 1.5107993065396212  
Epoch 7 - Training loss: 1.35543781867044 - Validating loss: 1.3885984613429794  
Epoch 8 - Training loss: 1.29635936266021 - Validating loss: 1.2825559801767707  
Epoch 9 - Training loss: 1.27687882133079 - Validating loss: 1.3177467516887087  
Epoch 10 - Training loss: 1.25443171495781 - Validating loss: 1.3202330311432977  
Epoch 11 - Training loss: 1.23273109407053 - Validating loss: 1.2317145807732657  
Epoch 12 - Training loss: 1.1987292532765 - Validating loss: 1.209769903076661  
Epoch 13 - Training loss: 1.17013863824366 - Validating loss: 1.2396785448516332  
Epoch 14 - Training loss: 1.17553892552322 - Validating loss: 1.238631098268376  
Epoch 15 - Training loss: 1.16294906888189 - Validating loss: 1.1677090361705007  
Epoch 16 - Training loss: 1.12969124166967 - Validating loss: 1.195619758834148  
Epoch 17 - Training loss: 1.13579455284543 - Validating loss: 1.1446844309632866  
Epoch 18 - Training loss: 1.10504974330452 - Validating loss: 1.1108574746635018  
Epoch 19 - Training loss: 1.11541749400538 - Validating loss: 1.0964983492651066

In [0]:

convol2D\_1.save\_weights('convol2D\_1')  
convol2D\_2.save\_weights('convol2D\_2')  
dense\_1.save\_weights('dense\_1')  
dense\_2.save\_weights('dense\_2')

In [0]:

predicted = model.predict\_class(X\_test\_norm[0:200])  
target = y\_test[0:200]

In [0]:

print("Accuracy: ",accuracy\_score(predicted, target))  
**from** **sklearn** **import** metrics  
print(metrics.classification\_report(target, predicted))

Accuracy: 0.75  
 precision recall f1-score support  
  
 0 0.94 1.00 0.97 17  
 1 0.92 0.79 0.85 28  
 2 0.67 0.62 0.65 16  
 3 0.65 0.81 0.72 16  
 4 0.79 0.82 0.81 28  
 5 0.81 0.65 0.72 20  
 6 0.79 0.75 0.77 20  
 7 0.89 0.67 0.76 24  
 8 0.40 1.00 0.57 10  
 9 0.69 0.52 0.59 21  
  
avg / total 0.78 0.75 0.75 200

Tiếp tục train thêm 10 epoch

In [0]:

batch\_size = 20  
epochs = 10  
  
model.fit(  
 X\_train\_norm[0:10000],   
 y\_train\_norm[0:10000],  
 batch\_size = batch\_size,  
 num\_epoch = epochs,  
 validation\_data = (X\_test\_norm[0:20], y\_test\_norm[0:20])  
)

Epoch 0 - Training loss: 1.10790770405805 - Validating loss: 1.155222209115932  
Epoch 1 - Training loss: 1.09282723283647 - Validating loss: 1.162309162119335  
Epoch 2 - Training loss: 1.08585056518316 - Validating loss: 1.1040841597355642  
Epoch 3 - Training loss: 1.09057195790592 - Validating loss: 1.1486624710842532  
Epoch 4 - Training loss: 1.07138561907188 - Validating loss: 1.0252644052066695  
Epoch 5 - Training loss: 1.06766938472369 - Validating loss: 1.0321799405416445  
Epoch 6 - Training loss: 1.06629894387216 - Validating loss: 1.0709090977270928  
Epoch 7 - Training loss: 1.0532231673278 - Validating loss: 1.0111623533597647  
Epoch 8 - Training loss: 1.04554922811798 - Validating loss: 1.0543915088628302  
Epoch 9 - Training loss: 1.05152575009738 - Validating loss: 1.0411727615378026

In [0]:

convol2D\_1.save\_weights('convol2D\_1')  
convol2D\_2.save\_weights('convol2D\_2')  
dense\_1.save\_weights('dense\_1')  
dense\_2.save\_weights('dense\_2')

In [0]:

predicted = model.predict\_class(X\_test\_norm[200:400])  
target = y\_test[200:400]  
print("Accuracy: ",accuracy\_score(predicted, target))  
**from** **sklearn** **import** metrics  
print(metrics.classification\_report(target, predicted))

Accuracy: 0.77  
 precision recall f1-score support  
  
 0 0.89 1.00 0.94 16  
 1 0.82 0.93 0.87 29  
 2 0.92 0.79 0.85 28  
 3 0.68 0.79 0.73 19  
 4 0.72 0.72 0.72 18  
 5 0.67 0.55 0.60 22  
 6 0.82 0.64 0.72 14  
 7 0.77 0.59 0.67 17  
 8 0.65 0.88 0.75 17  
 9 0.75 0.75 0.75 20  
  
avg / total 0.77 0.77 0.77 200