Deep Learning Lab

Lab-1 Learning XOR Problem

Date- 11-10-2020

```
import numpy as np
import pandas as pd
-
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[1],[1],[0]])
-
epochs = 100
lr = 0.1
-
def sigmoid (x):
    return 1/(1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)
-
hidden_weights = np.random.uniform(size=(2,2))
hidden_bias =np.random.uniform(size=(1,2))
output_weights = np.random.uniform(size=(2,1))
output_bias = np.random.uniform(size=(1,1))
-
print(*hidden_weights)
print(*hidden_bias)
print(*output_weights)
print(*output_bias)
```

Expected Output:-

```
[0.83691084 0.18954863] [0.56091971 0.75676505]
[0.56202789 0.30508382]
[0.8789262] [0.59486526]
[0.24976975]
```

```
for _ in range(epochs):
  hidden_layer_activation = np.dot(inputs,hidden_weights)
  hidden_layer_activation += hidden_bias
  hidden_layer_output = sigmoid(hidden_layer_activation)
  output_layer_activation = np.dot(hidden_layer_output,output_weights)
  output_layer_activation += output_bias
  predicted_output = sigmoid(output_layer_activation)
  error = expected_output - predicted_output
  d_predicted_output = error * sigmoid_derivative(predicted_output)
  error_hidden_layer = d_predicted_output.dot(output_weights.T)
  d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)
  output_weights += hidden_layer_output.T.dot(d_predicted_output) * lr
```

```
  output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * lr
  hidden_weights += inputs.T.dot(d_hidden_layer) * lr
  hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * lr
print(output_weights)
print(output_bias)
print(hidden_weights)
print(hidden_bias)
```

Expected output:-

```
[[-0.12751425]
 [ 0.06263322]]
[[0.0722675]]
[[0.24403306 0.51725332]
 [0.89044608 0.86264739]]
[[0.56084468 0.04947946]]
```

Image Classification using CNN -- 11/12/2020 Lab2

Dataset -- https://www.kaggle.com/fanconic/skin-cancer-malignant-vs-benign

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import os
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob
import seaborn as sns
from PIL import Image
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
-

from google.colab import drive
drive.mount('/content/drive')

-

benign_train = '/content/drive/My Drive/archive/data/train/benign'
malignant_train = '/content/drive/My Drive/archive/data/train/malignant'

benign_test = '/content/drive/My Drive/archive/data/test/benign'
malignant_test = '/content/drive/My Drive/archive/data/test/malignant'
```

Deep Learning Lab

-

```
read = lambda imname: np.asarray(Image.open(imname).convert("RGB"))

# Load in training pictures
ims_benign = [read(os.path.join(benign_train, filename)) for filename in os.listdir(b
enign_train)]
X_benign = np.array(ims_benign, dtype='uint8')
ims_malignant = [read(os.path.join(malignant_train, filename)) for filename in os.lis
tdir(malignant_train)]
X_malignant = np.array(ims_malignant, dtype='uint8')


ims_benign = [read(os.path.join(benign_test, filename)) for filename in os.listdir(be
nign_test)]
X_benign_test = np.array(ims_benign, dtype='uint8')
ims_malignant = [read(os.path.join(malignant_test, filename)) for filename in os.list
dir(malignant_test)]
X_malignant_test = np.array(ims_malignant, dtype='uint8')
```

-

```
# Create labels
y_benign = np.zeros(X_benign.shape[0])
y_malignant = np.ones(X_malignant.shape[0])

y_benign_test = np.zeros(X_benign_test.shape[0])
y_malignant_test = np.ones(X_malignant_test.shape[0])
```

-

```
# Merge data
X_train = np.concatenate((X_benign, X_malignant), axis = 0)
y_train = np.concatenate((y_benign, y_malignant), axis = 0)

X_test = np.concatenate((X_benign_test, X_malignant_test), axis = 0)
y_test = np.concatenate((y_benign_test, y_malignant_test), axis = 0)
```

-

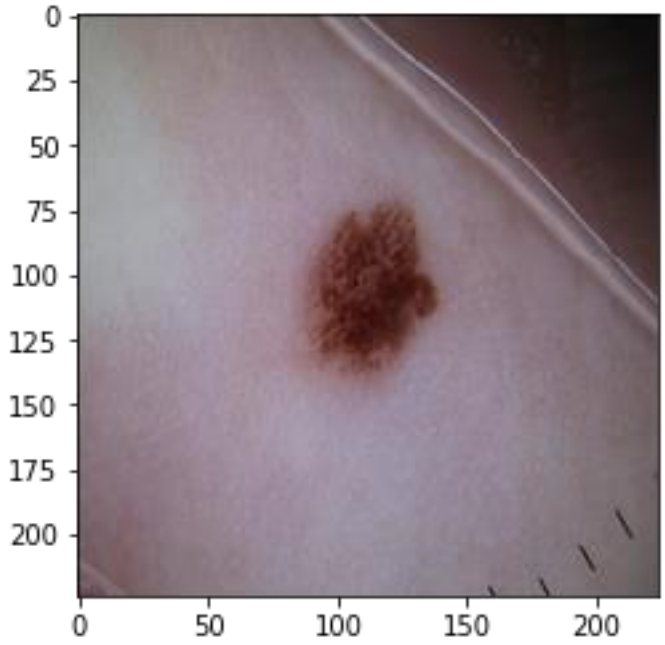
```
s = np.arange(X_train.shape[0])
np.random.shuffle(s)
X_train = X_train[s]
y_train = y_train[s]

s = np.arange(X_test.shape[0])
```

```
np.random.shuffle(s)
X_test = X_test[s]
y_test = y_test[s]
```

-

```
plt.imshow(X_test[1], interpolation='nearest')
plt.show()
```

-



-

```
X_train = X_train/255
X_test = X_test/255
import tensorflow as tf
X_Train = tf.keras.utils.normalize(X_train)
X_Test = tf.keras.utils.normalize(X_test)
```

-

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(128,(3,3), input_shape = X_Train.shape[1:] ,activati
on = tf.nn.relu ))
model.add(tf.keras.layers.MaxPool2D(pool_size=(3,3),strides=None))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64,activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Dense(32,activation=tf.nn.relu))
```

```
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Dense(2,activation=tf.nn.softmax))
from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
from keras.layers import Dense, Dropout, Activation, Flatten
model.compile(optimizer="adam",loss="sparse_categorical_crossentropy",metrics=["accur
acy"])
earlystop=EarlyStopping(monitor='val_loss',min_delta=0,
                        patience=5,
                        verbose=1,
                        restore_best_weights=True)

callbacks=[earlystop]
model.fit(X_Train, y_train, epochs = 50,callbacks=callbacks, shuffle=True,batch_size=
50, validation_split = 0.1)
```

-

```
y_pred = model.predict(X_Test)
yp =[]
for i in range(0,660):
    if y_pred[i][0] >= 0.5:
        yp.append(0)
    else:
        yp.append(1)
print(accuracy_score(y_test, yp))
```

-

Building a deep learning model 11/13/2020 lab 3

Dataset -- https://www.kaggle.com/fanconic/skin-cancer-malignant-vs-benign

benign_train = r'C:\Users\dmsss\Downloads\archive\data\train\benign'

malignant_train = r'C:\Users\dmsss\Downloads\archive\data\train\malignant'

benign_test = r'C:\Users\dmsss\Downloads\archive\data\test\benign'

malignant_test = r'C:\Users\dmsss\Downloads\archive\data\test\malignant'

-

X_test,test_y=[],[]

-

RGB images as input¶

import glob

Deep Learning Lab

```
import cv2
import numpy as np
for imageName in glob.glob(r'C:\Users\dmsss\Downloads\archive\data\train\benign\*.JPG'):
    a=cv2.imread(imageName)
    a=cv2.resize(a,(48,48))
    #a = cv2.cvtColor(a, cv2.COLOR_BGR2GRAY)
    a=a.reshape((48,48,3))
    #print(a)
    X_train.append(np.array(a,'float32'))
    train_y.append(0)


-
import glob
import cv2
import numpy as np
from IPython.display import Image, display
for imageName in glob.glob(r'C:\Users\dmsss\Downloads\archive\data\train\malignant\*.JPG'):
    a=cv2.imread(imageName)
    a=cv2.resize(a,(48,48))
    #a = cv2.cvtColor(a, cv2.COLOR_BGR2GRAY)
    a=a.reshape((48,48,3))
    #print(a)
    X_train.append(np.array(a,'float32'))
    train_y.append(1)
-
import glob
import cv2
import numpy as np
from IPython.display import Image, display
for imageName in glob.glob(r'C:\Users\dmsss\Downloads\archive\data\test\benign\*.JPG'):
    a=cv2.imread(imageName)
```

```
    a=cv2.resize(a,(48,48))
    #a = cv2.cvtColor(a, cv2.COLOR_BGR2GRAY)
    a=a.reshape((48,48,3))
    #print(a)
    X_test.append(np.array(a,'float32'))
    test_y.append(0)
import glob
import cv2
import numpy as np
from IPython.display import Image, display
for imageName in glob.glob(r'C:\Users\dmsss\Downloads\archive\data\test\malignant\*.JPG'):
    a=cv2.imread(imageName)
    a=cv2.resize(a,(48,48))
    #a = cv2.cvtColor(a, cv2.COLOR_BGR2GRAY)
    a=a.reshape((48,48,3))
    #print(a)
    X_test.append(np.array(a,'float32'))
    test_y.append(1)
-
from keras.models import Sequential
from keras.layers import Activation, Dropout, Flatten, Dense
import pandas as pd
import cv2
from keras.utils import np_utils
from PIL import Image, ImageEnhance
import os
import sys
import numpy as np
from PIL import Image
import glob
from keras.optimizers import RMSprop,SGD,Adam
```

Deep Learning Lab

```
from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Conv2D, MaxPooling2D, BatchNormalization,AveragePooling2D

from keras.losses import categorical_crossentropy

from keras.optimizers import Adam

from keras.utils import np_utils

from keras.models import model_from_json

from keras.preprocessing import image


num_features = 64

num_labels = 2

batch_size = 64

epochs = 50

width, height = 48, 48



X_train2 = np.array(X_train,'float32')

train_y2 = np.array(train_y,'float32')

X_test2 = np.array(X_test,'float32')

test_y2 = np.array(test_y,'float32')


train_y2=np_utils.to_categorical(train_y2, num_classes=num_labels)

test_y2=np_utils.to_categorical(test_y2, num_classes=num_labels)



X_train2 -= np.mean(X_train2, axis=0)

X_train2 /= np.std(X_train2, axis=0)


X_test2 -= np.mean(X_test2, axis=0)

X_test2 /= np.std(X_test2, axis=0)
```

```
X_train2 = X_train2.reshape(X_train2.shape[0], 48, 48, 3)


X_test2 = X_test2.reshape(X_test2.shape[0], 48, 48, 3)



model = Sequential()
#ist layer

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(X_train2.shape[1:])))
# model.add(Conv2D(32,kernel_size= (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

#########################

# #2nd convolution layer
model.add(Conv2D(64, (3, 3), activation='relu'))
# model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

#########################
model.add(Conv2D(128, (3, 3), activation='relu'))
# model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

#########################
```

Deep Learning Lab

```
########################


model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))



model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(2, activation='sigmoid'))

# model.summary()

#Compliling the model
model.compile(loss='binary_crossentropy',
          optimizer=Adam(lr=0.001),
          metrics=['accuracy'])
##
earlystop=EarlyStopping(monitor='val_loss',min_delta=0,
                  patience=5,
                  verbose=1,
                  restore_best_weights=True)
##
# reduce_lr=ReduceLROnPlateau(monitor='var_loss',
#                       factor=0.2,
#                       patience=3,
#                       verbose=1,
```

```
#                       min_delta=0.0001)
##
callbacks=[earlystop]
#Training the model
model.fit(X_train2, train_y2,
        batch_size=batch_size,
        epochs=epochs,
        callbacks=callbacks,
        validation_data=(X_test2, test_y2),
        shuffle=True)



#Saving the  model to  use it later on
fer_json = model.to_json()
with open("benvsmaligcancer.json", "w") as json_file:
   json_file.write(fer_json)
model.save_weights("benvsmaligcancer.h5")


-
from keras.utils import np_utils


num_labels = 2
width, height = 48, 48
pu_d = np.array(X_test,'float32')
pu_d_y = np.array(test_y,'float32')
pu_d_y=np_utils.to_categorical(pu_d_y, num_classes=num_labels)
pu_d -= np.mean(pu_d, axis=0)
pu_d /= np.std(pu_d, axis=0)
pu_d = pu_d.reshape(pu_d.shape[0], 48, 48, 3)
-
from keras.models import model_from_json
```

Deep Learning Lab

```
model = model_from_json(open("benvsmaligcancer.json", "r").read())
#load weights
model.load_weights('benvsmaligcancer.h5')
pu_d_o=[]
for i in pu_d:
   pu_d_o.append(np.argmax(model.predict(i.reshape(1,48,48,3))))
pu_d_o2=np_utils.to_categorical(pu_d_o, num_classes=num_labels)
import sklearn
from sklearn.metrics import accuracy_score

print(sklearn.metrics.accuracy_score(pu_d_y, pu_d_o2))
```

--

Grayscale as input

```
X_train,train_y=[],[]
X_test,test_y=[],[]
```

--

```
import glob
import cv2
import numpy as np
from IPython.display import Image, display
for imageName in glob.glob(r'C:\Users\dmsss\Downloads\archive\data\train\benign\*.JPG'):
   a=cv2.imread(imageName)
   a=cv2.resize(a,(48,48))
   a = cv2.cvtColor(a, cv2.COLOR_BGR2GRAY)
   a=a.reshape((48,48))
   #print(a)
   X_train.append(np.array(a,'float32'))
   train_y.append(0)
```

Deep Learning Lab

```
import glob
import cv2
import numpy as np
from IPython.display import Image, display
for imageName in glob.glob(r'C:\Users\dmsss\Downloads\archive\data\train\malignant\*.JPG'):
    a=cv2.imread(imageName)
    a=cv2.resize(a,(48,48))
    a = cv2.cvtColor(a, cv2.COLOR_BGR2GRAY)
    a=a.reshape((48,48))
    #print(a)
    X_train.append(np.array(a,'float32'))
    train_y.append(1)
--
import glob
import cv2
import numpy as np
from IPython.display import Image, display
for imageName in glob.glob(r'C:\Users\dmsss\Downloads\archive\data\test\benign\*.JPG'):
    a=cv2.imread(imageName)
    a=cv2.resize(a,(48,48))
    a = cv2.cvtColor(a, cv2.COLOR_BGR2GRAY)
    a=a.reshape((48,48))
    #print(a)
    X_test.append(np.array(a,'float32'))
    test_y.append(0)
import glob
import cv2
import numpy as np
from IPython.display import Image, display
for imageName in glob.glob(r'C:\Users\dmsss\Downloads\archive\data\test\malignant\*.JPG'):
    a=cv2.imread(imageName)
```

```
    a=cv2.resize(a,(48,48))
    a = cv2.cvtColor(a, cv2.COLOR_BGR2GRAY)
    a=a.reshape((48,48))
    #print(a)
    X_test.append(np.array(a,'float32'))
    test_y.append(1)
--
from keras.models import Sequential
from keras.layers import Activation, Dropout, Flatten, Dense
import pandas as pd
import cv2
from keras.utils import np_utils
from PIL import Image, ImageEnhance
import os
import sys
import numpy as np
from PIL import Image
import glob
from keras.optimizers import RMSprop,SGD,Adam
from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization,AveragePooling2D
from keras.losses import categorical_crossentropy
from keras.optimizers import Adam
from keras.utils import np_utils
from keras.models import model_from_json
from keras.preprocessing import image



num_features = 64
num_labels = 2
```

Deep Learning Lab

```
batch_size = 64

epochs = 50

width, height = 48, 48



X_train2 = np.array(X_train,'float32')

train_y2 = np.array(train_y,'float32')

X_test2 = np.array(X_test,'float32')

test_y2 = np.array(test_y,'float32')


train_y2=np_utils.to_categorical(train_y2, num_classes=num_labels)

test_y2=np_utils.to_categorical(test_y2, num_classes=num_labels)



X_train2 -= np.mean(X_train2, axis=0)

X_train2 /= np.std(X_train2, axis=0)


X_test2 -= np.mean(X_test2, axis=0)

X_test2 /= np.std(X_test2, axis=0)


X_train2 = X_train2.reshape(X_train2.shape[0], 48, 48, 1)


X_test2 = X_test2.reshape(X_test2.shape[0], 48, 48, 1)


model = Sequential()

#ist layer


model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(X_train2.shape[1:])))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))

model.add(Dropout(0.5))
```

Deep Learning Lab

```
########################

# #2nd convolution layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

########################
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

########################

########################

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))



model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(2, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
```

```
        optimizer=Adam(lr=0.001),
        metrics=['accuracy'])
##
earlystop=EarlyStopping(monitor='val_loss',min_delta=0,
                patience=5,
                verbose=1,
                restore_best_weights=True)

callbacks=[earlystop]
model.fit(X_train2, train_y2,
      batch_size=batch_size,
      epochs=epochs,
      callbacks=callbacks,
      validation_data=(X_test2, test_y2),
      shuffle=True)



#Saving the  model to  use it later on
fer_json = model.to_json()
with open("benvsmaligcancerbw.json", "w") as json_file:
   json_file.write(fer_json)
model.save_weights("benvsmaligcancerbw.h5")

-


from keras.utils import np_utils
num_labels = 2
width, height = 48, 48
pu_d = np.array(X_test,'float32')
pu_d_y = np.array(test_y,'float32')
pu_d_y=np_utils.to_categorical(pu_d_y, num_classes=num_labels)
```

```
pu_d -= np.mean(pu_d, axis=0)
pu_d /= np.std(pu_d, axis=0)
pu_d = pu_d.reshape(pu_d.shape[0], 48, 48, 1)
```

-

```
from keras.models import model_from_json

model = model_from_json(open("benvsmaligcancerbw.json", "r").read())
#load weights
model.load_weights('benvsmaligcancerbw.h5')
pu_d_o=[]
for i in pu_d:
    pu_d_o.append(np.argmax(model.predict(i.reshape(1,48,48,1))))
pu_d_o2=np_utils.to_categorical(pu_d_o, num_classes=num_labels)
import sklearn
from sklearn.metrics import accuracy_score

print(sklearn.metrics.accuracy_score(pu_d_y, pu_d_o2))
```

-

Using 2 Layer Model

```
from keras.models import Sequential
from keras.layers import Activation, Dropout, Flatten, Dense
import pandas as pd
import cv2
from keras.utils import np_utils
from PIL import Image, ImageEnhance
import os
import sys
```

Deep Learning Lab

```
import numpy as np
from PIL import Image
import glob
from keras.optimizers import RMSprop,SGD,Adam
from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization,AveragePooling2D
from keras.losses import categorical_crossentropy
from keras.optimizers import Adam
from keras.utils import np_utils
from keras.models import model_from_json
from keras.preprocessing import image



num_features = 64
num_labels = 2
batch_size = 64
epochs = 50
width, height = 48, 48



X_train2 = np.array(X_train,'float32')
train_y2 = np.array(train_y,'float32')
X_test2 = np.array(X_test,'float32')
test_y2 = np.array(test_y,'float32')

train_y2=np_utils.to_categorical(train_y2, num_classes=num_labels)
test_y2=np_utils.to_categorical(test_y2, num_classes=num_labels)



X_train2 -= np.mean(X_train2, axis=0)
```

```
X_train2 /= np.std(X_train2, axis=0)


X_test2 -= np.mean(X_test2, axis=0)

X_test2 /= np.std(X_test2, axis=0)


X_train2 = X_train2.reshape(X_train2.shape[0], 48, 48, 1)


X_test2 = X_test2.reshape(X_test2.shape[0], 48, 48, 1)


model = Sequential()

#ist layer


model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(X_train2.shape[1:])))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))

model.add(Dropout(0.5))


#########################


# #2nd convolution layer

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))

model.add(Dropout(0.5))


#########################

# model.add(Conv2D(128, (3, 3), activation='relu'))

# model.add(BatchNormalization())

# model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))

# model.add(Dropout(0.5))
```

Deep Learning Lab

```
########################

########################

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))



model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(2, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
          optimizer=Adam(lr=0.001),
          metrics=['accuracy'])
##
earlystop=EarlyStopping(monitor='val_loss',min_delta=0,
                  patience=5,
                  verbose=1,
                  restore_best_weights=True)

callbacks=[earlystop]
history=model.fit(X_train2, train_y2,
        batch_size=batch_size,
        epochs=epochs,
        callbacks=callbacks,
        validation_data=(X_test2, test_y2),
        shuffle=True)
```

Deep Learning Lab

```
#Saving the  model to  use it later on
fer_json = model.to_json()
with open("benvsmaligcancerbw2layer.json", "w") as json_file:
    json_file.write(fer_json)
model.save_weights("benvsmaligcancerbw2layer.h5")


-
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Deep Learning Lab

Data Augmentation lab --11/20/2020

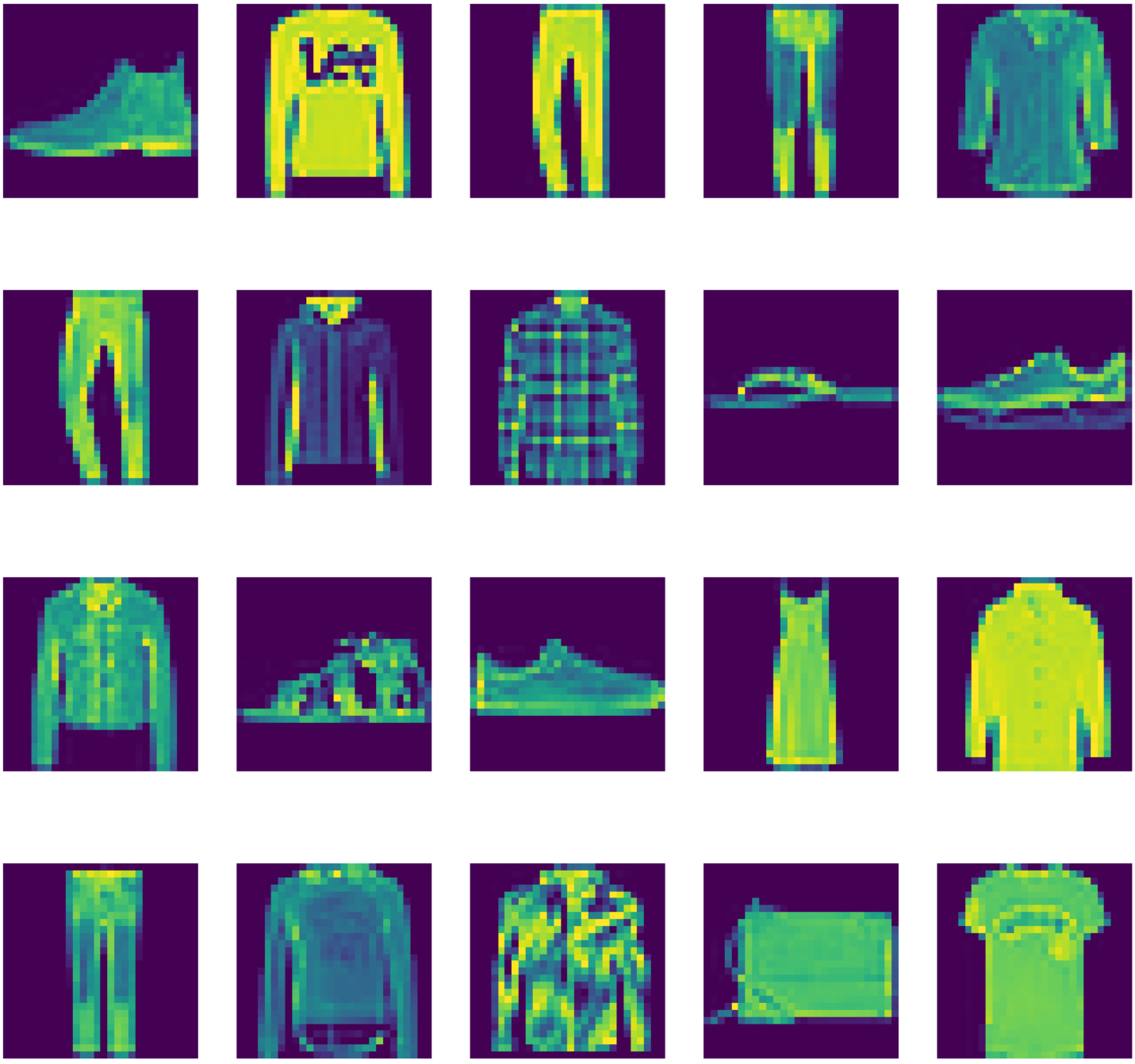
```
import numpy as np
from keras.utils import np_utils
from keras.datasets import fashion_mnist
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

--

```
plt.figure(figsize=(20, 20))
for i in range(20):
    plt.subplot(4, 5, i+1)
    plt.imshow(X_test[i])
    plt.axis('off')
plt.show()
```

-

Deep Learning Lab



-

X_train=X_train.astype('float32')

X_test=X_test.astype('float32')

X_train/=255

X_test/=255

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')

```
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')


y_train = np_utils.to_categorical(y_train, 10)

y_test = np_utils.to_categorical(y_test, 10)

--

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten, Reshape

from keras.layers import Convolution2D, MaxPooling2D

model = Sequential()

model.add(Convolution2D(32, 3, 3, input_shape=(28, 28, 1)))

model.add(Activation('relu'))

model.add(Dropout(0.25))

model.add(Convolution2D(32, 3, 3))

model.add(Activation('relu'))

model.add(Convolution2D(32, 3, 3))

model.add(Activation('relu'))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128))

model.add(Dense(128))

model.add(Activation('relu'))

model.add(Dense(10))

model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X_train, y_train, batch_size=32,epochs=10,validation_data=(X_test, y_test))

--


from keras.preprocessing.image import ImageDataGenerator

from matplotlib import pyplot as plt

gena = ImageDataGenerator(

   rotation_range=50,
```

```
    width_shift_range=0.01,
    height_shift_range=0.01)
gena.fit(X_train)
gen = gena.flow(X_train[1:2], batch_size=1)
--


plt.figure(figsize=(20, 20))
for i in range(20):
    plt.subplot(4, 5, i+1)
    plt.imshow(X_test[i])
    plt.axis('off')
    plt.imshow(gen.next().squeeze())
    plt.plot()
plt.show()
```

Deep Learning Lab

--

```
model.fit(X_train, y_train, batch_size=32,epochs=10, validation_data=(X_test, y_test))

print('Test accuracy:', model.evaluate(X_test, y_test))
```

--

Deep Learning Lab

Implementation of RNN

```
import logging

import pandas as pd

import numpy as np

from numpy import random

import gensim

import nltk

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

from nltk.corpus import stopwords

import re

from bs4 import BeautifulSoup

import pandas as pd

import os

import re

import spacy

from gensim.models.phrases import Phrases, Phraser

from time import time

import multiprocessing

from gensim.models import Word2Vec

import bokeh.plotting as bp

from bokeh.models import HoverTool, BoxSelectTool

from bokeh.plotting import figure, show, output_notebook

from sklearn.manifold import TSNE

from sklearn.model_selection import train_test_split

import numpy as np

from sklearn.preprocessing import scale

import keras

from keras.models import Sequential, Model
```

Deep Learning Lab

```
from keras import layers
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, Input, Embedding
from keras.layers.merge import Concatenate
from sklearn.feature_extraction.text import TfidfVectorizer
from wordcloud import WordCloud
from nltk.tokenize import RegexpTokenizer
from sklearn.metrics import confusion_matrix
--

X=df[['text']]
y=df[['target']]
description_list = df['text'].tolist()
text=np.array(df['target'].tolist())

--

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import pickle
count_vect = CountVectorizer()
#count_vect._validate_vocabulary()
x_train_counts = count_vect.fit_transform(description_list)

tfidf_transformer = TfidfTransformer()
x_train_tfidf = tfidf_transformer.fit_transform(x_train_counts)

# Save the vectorizer
vec_file = 'vectorizer.pickle'
pickle.dump(count_vect, open(vec_file, 'wb'))
```

```
   # Save the model
# mod_file = 'classification.model'
# pickle.dump(model, open(mod_file, 'wb'))
--


#building a simple RNN model


   model = Sequential()
   model.add(keras.layers.InputLayer(input_shape=(15,1)))
   keras.layers.embeddings.Embedding(nb_words, 15, weights=[embedding_matrix], input_length=15,
   trainable=False)


   model.add(keras.layers.recurrent.SimpleRNN(units = 100, activation='relu',
   use_bias=True))
   model.add(keras.layers.Dense(units=1000, input_dim = 2000, activation='sigmoid'))
   model.add(keras.layers.Dense(units=500, input_dim=1000, activation='relu'))
   model.add(keras.layers.Dense(units=2, input_dim=500,activation='softmax'))
   model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Bidirectional LSTM -- 11/24/2020

Dataset- https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import sys, os, re, csv, codecs, numpy as np, pandas as pd


from keras.preprocessing.text import Tokenizer
```

Deep Learning Lab

```
from keras.preprocessing.sequence import pad_sequences

from keras.layers import RNN,Dense, Input, LSTM, Embedding, Dropout, Activation

from keras.layers import Bidirectional, GlobalMaxPool1D

from keras.models import Model

from keras import initializers, regularizers, constraints, optimizers, layers

from subprocess import check_output

--



EMBEDDING_FILE=r"C:\Users\dmsss\Downloads\deep learning lab\glove.6B.100d.txt\glove.6B.100d.txt"

TRAIN_DATA_FILE=r'C:\Users\dmsss\Downloads\deep learning lab\toxic classification\train\train.csv'

TEST_DATA_FILE=r'C:\Users\dmsss\Downloads\deep learning lab\toxic classification\test\test.csv'

--



embed_size = 100

max_features = 25000

maxlen = 100

train = pd.read_csv(TRAIN_DATA_FILE)

test = pd.read_csv(TEST_DATA_FILE)

a=EMBEDDING_FILE.read()

---


train.head()
```

Deep Learning Lab

--

```
list_sentences_train = train["comment_text"].fillna("_na_").values
list_classes = ["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]
y = train[list_classes].values
list_sentences_test = test["comment_text"].fillna("_na_").values
```

--

```
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(list_sentences_train))
list_tokenized_train = tokenizer.texts_to_sequences(list_sentences_train)
list_tokenized_test = tokenizer.texts_to_sequences(list_sentences_test)
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
```

--

```
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.strip().split()) for o in
open(EMBEDDING_FILE,encoding='utf8'))
```

--

```
all_embs = np.stack(embeddings_index.values())
emb_mean,emb_std = all_embs.mean(), all_embs.std()
```

emb_mean,emb_std

--

```
word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index))
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector
```

--

```
inp = Input(shape=(maxlen,))
x = Embedding(max_features, embed_size, weights=[embedding_matrix])(inp)
x = Bidirectional(LSTM(100, return_sequences=True, dropout=0.25,
recurrent_dropout=0.1))(x)
x = GlobalMaxPool1D()(x)
x = Dense(100, activation="relu")(x)
x = Dropout(0.25)(x)
x = Dense(6, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_t, y, batch_size=32, epochs=10) # validation_split=0.1);
```

--

Deep Learning Lab

LSTM -- Dec 1 2020

Dataset-- https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews

```
from google.colab import drive
drive.mount('/content/drive')
from tensorflow.keras.preprocessing import text_dataset_from_directory

# Assumes you're in the root level of the dataset directory.
# If you aren't, you'll need to change the relative paths here.
train_data = text_dataset_from_directory("/content/drive/My Drive/movie-reviews-dataset/train")
test_data = text_dataset_from_directory("/content/drive/My Drive/movie-reviews-dataset/test")
```

--

```
from tensorflow.keras.preprocessing import text_dataset_from_directory
from tensorflow.strings import regex_replace

def prepareData(dir):
  data = text_dataset_from_directory(dir)
  return data.map(
    lambda text, label: (regex_replace(text, '<br />', ' '), label),
  )

train_data = prepareData('/content/drive/My Drive/movie-reviews-dataset/train')
test_data = prepareData('/content/drive/My Drive/movie-reviews-dataset/test')
```

-

```
for text_batch, label_batch in train_data.take(1):
  print(text_batch.numpy()[0])
  print(label_batch.numpy()[0]) # 0 = negative, 1 = positive
```

Deep Learning Lab

-

```
from tensorflow.keras.models import Sequential
from tensorflow.keras import Input

model = Sequential()
model.add(Input(shape=(1,), dtype="string"))
```

-

```
from tensorflow.keras.layers.experimental.preprocessing import TextVectori
zation

max_tokens = 1000
max_len = 100
vectorize_layer = TextVectorization(

  max_tokens=max_tokens,

  output_mode="int",

  output_sequence_length=max_len,
)
```

-

```
train_texts = train_data.map(lambda text, label: text)
vectorize_layer.adapt(train_texts)
model.add(vectorize_layer)
```

--

```
from tensorflow.keras.layers import Embedding
max_tokens = 1000
#model.add(vectorize_layer)
```

Deep Learning Lab

```
model.add(Embedding(max_tokens + 1, 128))
from tensorflow.keras.layers import LSTM
model.add(LSTM(64))
from tensorflow.keras.layers import Dense

model.add(Dense(64, activation="relu"))
model.add(Dense(1, activation="sigmoid"))


--



model.compile(
  optimizer='adam',
  loss='binary_crossentropy',
  metrics=['accuracy'],
)
model.fit(train_data, epochs=10)


--



print(model.predict([
  "i loved it! highly recommend it to anyone and everyone looking for a gr
eat movie to watch.",
]))

# Should print a very low score like 0.01.
print(model.predict([
  "this was awful! i hated it so much, nobody should watch this. the actin
g was terrible, the music was terrible, overall it was just bad.",
]))
```

Expected output—1 0

Deep Learning Lab

Variational Auto Encoders @ 12/3/2020

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data')
```

--

```
#import tensorflow.compat.v1 as tf
#tf.disable_v2_behavior()
tf.reset_default_graph()

batch_size = 64

X_in = tf.placeholder(dtype=tf.float32, shape=[None, 28, 28], name='X')
Y    = tf.placeholder(dtype=tf.float32, shape=[None, 28, 28], name='Y')
Y_flat = tf.reshape(Y, shape=[-1, 28 * 28])
keep_prob = tf.placeholder(dtype=tf.float32, shape=(), name='keep_prob')

dec_in_channels = 1
n_latent = 8

reshaped_dim = [-1, 7, 7, dec_in_channels]
inputs_decoder = (49 * dec_in_channels) // 2


def lrelu(x, alpha=0.3):
    return tf.maximum(x, tf.multiply(x, alpha))
```

--

```
def encoder(X_in, keep_prob):
    activation = lrelu
    with tf.variable_scope("encoder", reuse=None):
        X = tf.reshape(X_in, shape=[-1, 28, 28, 1])
        x = tf.layers.conv2d(X, filters=64, kernel_size=4, strides=2, padding='same',
 activation=activation)
        x = tf.nn.dropout(x, keep_prob)
```

```
        x = tf.layers.conv2d(x, filters=64, kernel_size=4, strides=2, padding='same',
 activation=activation)
        x = tf.nn.dropout(x, keep_prob)
        x = tf.layers.conv2d(x, filters=64, kernel_size=4, strides=1, padding='same',
 activation=activation)
        x = tf.nn.dropout(x, keep_prob)
        x = tf.layers.flatten(x)
        mn = tf.layers.dense(x, units=n_latent)
        sd       = 0.5 * tf.layers.dense(x, units=n_latent)
        epsilon = tf.random_normal(tf.stack([tf.shape(x)[0], n_latent]))
        z  = mn + tf.multiply(epsilon, tf.exp(sd))

        return z, mn, sd
```

--

```
def decoder(sampled_z, keep_prob):
    with tf.variable_scope("decoder", reuse=None):
        x = tf.layers.dense(sampled_z, units=inputs_decoder, activation=lrelu)
        x = tf.layers.dense(x, units=inputs_decoder * 2 + 1, activation=lrelu)
        x = tf.reshape(x, reshaped_dim)
        x = tf.layers.conv2d_transpose(x, filters=64, kernel_size=4, strides=2, paddi
ng='same', activation=tf.nn.relu)
        x = tf.nn.dropout(x, keep_prob)
        x = tf.layers.conv2d_transpose(x, filters=64, kernel_size=4, strides=1, paddi
ng='same', activation=tf.nn.relu)
        x = tf.nn.dropout(x, keep_prob)
        x = tf.layers.conv2d_transpose(x, filters=64, kernel_size=4, strides=1, paddi
ng='same', activation=tf.nn.relu)

        x = tf.layers.flatten(x)
        x = tf.layers.dense(x, units=28*28, activation=tf.nn.sigmoid)
        img = tf.reshape(x, shape=[-1, 28, 28])
        return img
```

--

```
sampled, mn, sd = encoder(X_in, keep_prob)
dec = decoder(sampled, keep_prob)
```

--

Deep Learning Lab

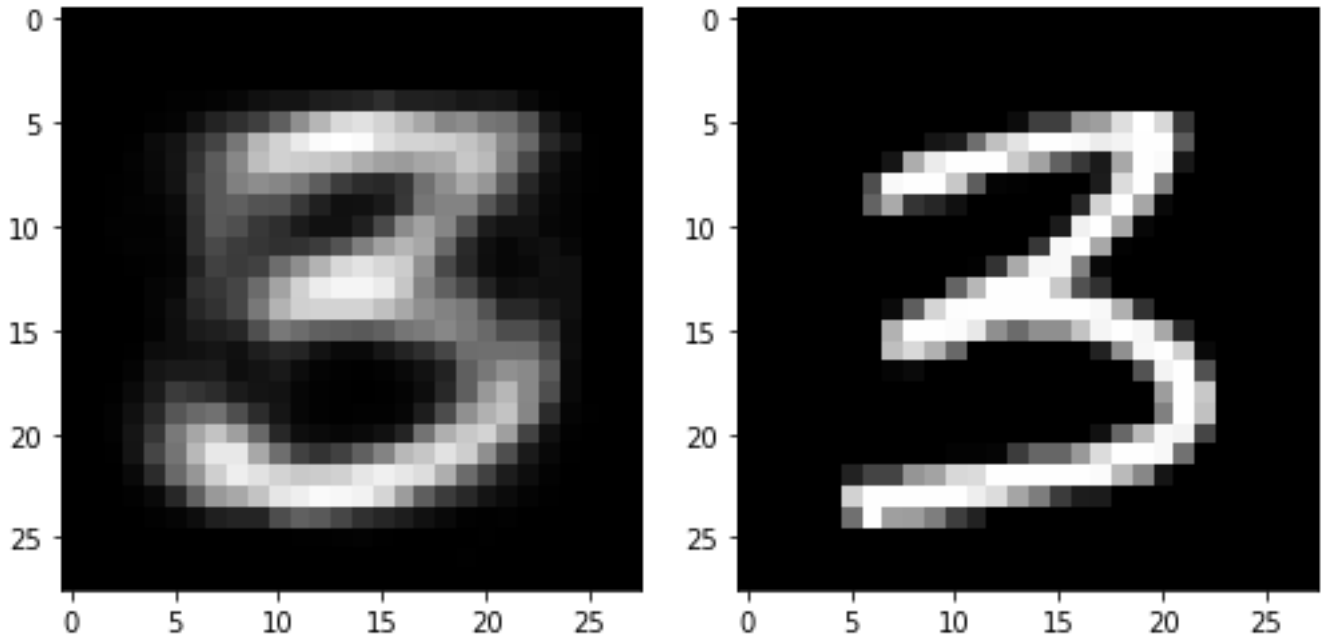
```
unreshaped = tf.reshape(dec, [-1, 28*28])
img_loss = tf.reduce_sum(tf.squared_difference(unreshaped, Y_flat), 1)
latent_loss = -
0.5 * tf.reduce_sum(1.0 + 2.0 * sd - tf.square(mn) - tf.exp(2.0 * sd), 1)
loss = tf.reduce_mean(img_loss + latent_loss)
optimizer = tf.train.AdamOptimizer(0.0005).minimize(loss)
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

--

```
for i in range(30000):
    batch = [np.reshape(b, [28, 28]) for b in mnist.train.next_batch(batch_size=batch
_size)[0]]
    sess.run(optimizer, feed_dict = {X_in: batch, Y: batch, keep_prob: 0.8})

    if not i % 200:
        ls, d, i_ls, d_ls, mu, sigm = sess.run([loss, dec, img_loss, latent_loss, mn,
 sd], feed_dict = {X_in: batch, Y: batch, keep_prob: 1.0})
        plt.imshow(np.reshape(batch[0], [28, 28]), cmap='gray')
        plt.show()
        plt.imshow(d[0], cmap='gray')
        plt.show()
        print(i, ls, np.mean(i_ls), np.mean(d_ls))
```

--

```
1200 31.73598 22.617443 9.118536
```



RBM's

```
import struct
from pylab import *
from array import array
import numpy as np
import os
def load_mnist(n_examples=0, training = True):
  if training:
    values = 'data/train-images’
    labels = 'data/train-labels'
  else:
    values = 'data/t10k-images'
    labels = 'data/t10k-labels'

  with open(values, "rb") as f:
    magic_number,n_images,n_rows,n_columns = struct.unpack('>iiii', f.read(16))

    if (n_examples == 0 or n_examples > n_images):
      n_examples = n_images # load all examples

    raw = array("B", f.read(int(n_rows * n_columns * n_examples)))
    images = np.zeros((n_examples, int(n_rows * n_columns)), dtype=np.uint8)

    for i in range(n_examples):
      start = int(i * n_rows * n_columns)
      end = int((i+1) * n_rows * n_columns)
      images[i] = np.array(raw[start : end])

    images = np.true_divide(images, 255) # all features between 0 and 1

  with open(labels, "rb") as f:
    magic_number,n_labels = struct.unpack('>ii', f.read(8))
```

```
    raw = array("B", f.read(int(n_examples)))
    labels = np.array(raw, dtype=np.uint8)
  return images, labels


def filter_by_digit_mnist(digit, images, labels):
  indexes = [i for i in range(len(labels)) if labels[i] == digit]
  return images[indexes]


def save_mnist_image(image, directory, filename):
  if not os.path.exists(directory):
    os.makedirs(directory)
  imshow(image.reshape((28,28)), cmap=cm.gray)
  axis('off')
  savefig(directory + os.sep + filename)
if __name__ == '__main__':
  images, labels = load_mnist(n_examples=10, training=True)
  filtered = filter_by_digit_mnist(3, images, labels)
```

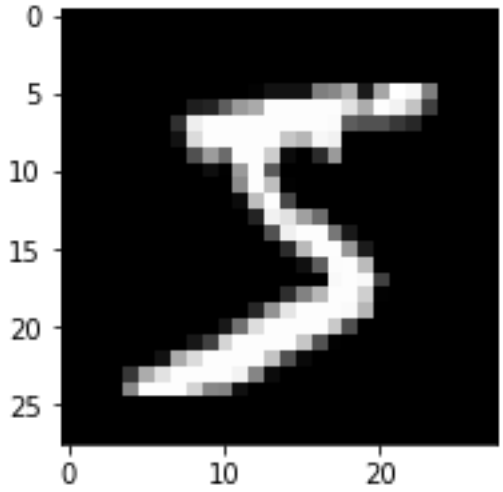
GAN’s

```
from torchvision import datasets
import torchvision.transforms as transforms
import torch
num_workers = 0
batch_size = 64
transform = transforms.ToTensor()
train_data = datasets.MNIST(root='data', train=True,
 download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
 num_workers=num_workers)
```

--

Deep Learning Lab

```
import numpy as np
dataiter = iter(train_loader)
images, labels = dataiter.next()
images = images.numpy()
img = np.squeeze(images[0])
fig = plt.figure(figsize = (3,3))
ax = fig.add_subplot(111)
ax.imshow(img, cmap='gray')
```




```
import torch.nn as nn
import torch.nn.functional as F
class Discriminator(nn.Module):

  def __init__(self, input_size, hidden_dim, output_size):

    super(Discriminator, self).__init__()
    self.fc1 = nn.Linear(input_size, hidden_dim*4)
    self.fc2 = nn.Linear(hidden_dim*4, hidden_dim*2)
    self.fc3 = nn.Linear(hidden_dim*2, hidden_dim)
    self.fc4 = nn.Linear(hidden_dim, output_size)
    self.dropout = nn.Dropout(0.3)


  def forward(self, x):
    x = x.view(-1, 28*28)
    x = F.leaky_relu(self.fc1(x), 0.2) # (input, negative_slope=0.2)
    x = self.dropout(x)
    x = F.leaky_relu(self.fc2(x), 0.2)
    x = self.dropout(x)
    x = F.leaky_relu(self.fc3(x), 0.2)
```

```
    x = self.dropout(x)
    out = self.fc4(x)
    return out
```

--

```
class Generator(nn.Module):
  def __init__(self, input_size, hidden_dim, output_size):
    super(Generator, self).__init__()
    self.fc1 = nn.Linear(input_size, hidden_dim)
    self.fc2 = nn.Linear(hidden_dim, hidden_dim*2)
    self.fc3 = nn.Linear(hidden_dim*2, hidden_dim*4)
    self.fc4 = nn.Linear(hidden_dim*4, output_size)
    self.dropout = nn.Dropout(0.3)
  def forward(self, x):
    x = F.leaky_relu(self.fc1(x), 0.2) # (input, negative_slope=0.2)
    x = self.dropout(x)
    x = F.leaky_relu(self.fc2(x), 0.2)
    x = self.dropout(x)
    x = F.leaky_relu(self.fc3(x), 0.2)
    x = self.dropout(x)
    out = F.tanh(self.fc4(x))
    return out
```

--

```
input_size = 784
d_output_size = 1
d_hidden_size = 32
z_size = 100
g_output_size = 784
g_hidden_size = 32
```

--

```
D = Discriminator(input_size, d_hidden_size, d_output_size)
G = Generator(z_size, g_hidden_size, g_output_size)
```

--

```
def real_loss(D_out, smooth=False):
  batch_size = D_out.size(0)
  if smooth:
   labels = torch.ones(batch_size)*0.9
  else:
   labels = torch.ones(batch_size) # real labels = 1
  criterion = nn.BCEWithLogitsLoss()
  loss = criterion(D_out.squeeze(), labels)
  return loss
def fake_loss(D_out):
 batch_size = D_out.size(0)
 labels = torch.zeros(batch_size) # fake labels = 0
 criterion = nn.BCEWithLogitsLoss()
 # calculate loss
 loss = criterion(D_out.squeeze(), labels)
 return loss
```

--

```
import torch.optim as optim
lr = 0.002
d_optimizer = optim.Adam(D.parameters(), lr)
g_optimizer = optim.Adam(G.parameters(), lr)
```

--

```
import pickle as pkl
num_epochs = 100
samples = []
losses = []
print_every = 400
sample_size=16
fixed_z = np.random.uniform(-1, 1, size=(sample_size, z_size))
fixed_z = torch.from_numpy(fixed_z).float()
D.train()
G.train()
for epoch in range(num_epochs):
    for batch_i, (real_images, _) in enumerate(train_loader):
        batch_size = real_images.size(0)
        real_images = real_images*2 - 1  # rescale input images from [0,1) to [-1, 1)
        d_optimizer.zero_grad()
        D_real = D(real_images)
```

```
        d_real_loss = real_loss(D_real, smooth=True)
        z = np.random.uniform(-1, 1, size=(batch_size, z_size))
        z = torch.from_numpy(z).float()
        fake_images = G(z)
        D_fake = D(fake_images)
        d_fake_loss = fake_loss(D_fake)
        d_loss = d_real_loss + d_fake_loss
        d_loss.backward()
        d_optimizer.step()
        g_optimizer.zero_grad()
        z = np.random.uniform(-1, 1, size=(batch_size, z_size))
        z = torch.from_numpy(z).float()
        fake_images = G(z)
        D_fake = D(fake_images)
        g_loss = real_loss(D_fake) # use real loss to flip labels
        g_loss.backward()
        g_optimizer.step()
        if batch_i % print_every == 0:
            print('Epoch [{:5d}/{:5d}] | d_loss: {:6.4f} | g_loss: {:6.4f}'.format(
                    epoch+1, num_epochs, d_loss.item(), g_loss.item()))
    losses.append((d_loss.item(), g_loss.item()))
    G.eval()
    samples_z = G(fixed_z)
    samples.append(samples_z)
    G.train() # back to train mode
with open('train_samples.pkl', 'wb') as f:
    pkl.dump(samples, f)
```

--

```
# helper function for viewing a list of passed in sample images
def view_samples(epoch, samples):
    fig, axes = plt.subplots(figsize=(7,7), nrows=4, ncols=4, sharey=True, sharex=Tru
e)
    for ax, img in zip(axes.flatten(), samples[epoch]):
        img = img.detach()
        ax.xaxis.set_visible(False)
        ax.yaxis.set_visible(False)
        im = ax.imshow(img.reshape((28,28)), cmap='Greys_r')
```

--

```
# Load samples from generator, taken while training
with open('train_samples.pkl', 'rb') as f:
    samples = pkl.load(f)
```

Deep Learning Lab

--

```
rows = 10 # split epochs into 10, so 100/10 = every 10 epochs
cols = 6
fig, axes = plt.subplots(figsize=(7,12), nrows=rows, ncols=cols, sharex=True, sharey=
True)

for sample, ax_row in zip(samples[::int(len(samples)/rows)], axes):
    for img, ax in zip(sample[::int(len(sample)/cols)], ax_row):
        img = img.detach()
        ax.imshow(img.reshape((28,28)), cmap='Greys_r')
        ax.xaxis.set_visible(False)
        ax.yaxis.set_visible(False)
```

Deep Learning Lab