ST: BIG DATA ANALYTICS

(CS 696-16) (FA18)

Project 1

“Facial Expression Recognition using 3-layered Neural Network”

Submitted By

Ashok Kumar Shrestha

#A25265227

Contents

S.N.	Topics	Page No.
1.	Introduction	1
2.	Methods	1
3.	Results and Discussion	1
4.	Pros and Cons of PCA	3
5.	Conclusion	4
6.	References	5
7.	Appendix	6

Introduction:

For the project 1, I have chosen to work on three datasets: MNIST[1], CIFAR-10[2] and Facial Expression datasets[3] and focusing mainly on the last one. The facial expression datasets consists of 28,709 labeled samples in csv file. The file consists of two columns: emotion and pixels. Each sample consist of string of 48x48 image in grayscale format in "pixel" column and its corresponding label represented by integer in "emotion" column. The image consists of the various facial expressions which are labeled with one of seven predefined expressions. They are (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

The MNIST datasets consist of 28x28 grayscale image of handwritten digits from 0-9. It has 60,000 training datasets and 10,000 testing datasets. Similarly, The CIFAR-10 datasets consist of 32x32 RGB, 3-channel colored image of 10 classes. It has over 50,000 training datasets and 10,000 testing datasets.

For this project, the three layer neural network is designed for the classification purpose. The details of the network is mentioned in the methods section. All the codings are done in Python on Jupyter Notebook with Keras library support[4].

Methods:

The architecture of this project consists of 3 layered neural network that can be extended or reduced the number of layers as desired. The layers are:

1. First layer is fully connected layer with Relu activation function.
2. Second layer is also fully connected layer with Relu activation function.
3. Third layer is also the fully connected layer with Softmax as activation function.

In the model, adam is used as the optimizer with categorical_crossentropy as the loss function and batch size of 512. StandardScaler is used for pre-processing the data sets. The network was tested with various value for PCA. The results are compared and contrasted to reflect the effect of using PCA on the network. The model was tested on three datasets mentioned previously on introduction section.

Results and Discussion:

The following results were obtained without using PCA in the network.

S.N.	Datasets	Samples	Epoch	Train Accuracy	Test Accuracy
1.	MNIST	60,000	200	100.0 %	98.08 %
2.	CIFAR-10	10,000	200	63.10 %	49.62 %
3.	Facial Expression	28,709	200	67.24 %	44.68 %

Table 1: Without using PCA on the model

The following results were obtained with PCA(0.95) in the network.

S.N.	Datasets	Samples	Epoch	Train Accuracy	Test Accuracy
1.	MNIST	60,000	200	99.81 %	97.26 %
2.	CIFAR-10	10,000	200	91.44 %	43.36 %
3.	Facial Expression	28,709	200	73.61 %	43.90 %

Table 2: Using PCA on the model

The following results were obtained with various values of PCA in the network for Facial expression datasets. The training sample datasets reduced to 10,000 for each.

S.N.	PCA	No. of Components	Test Accuracy
1.	1.0	2303	43.03 %
2.	0.99	843	43.88 %
3.	0.95	269	43.90 %
4.	0.90	113	45.77 %
5.	0.85	59	44.37 %

Table 3: Various value of PCA on the model

Graph and plots:

Datasets: Facial Expression datasets
Epochs: 200
Optimizer: Adam
Loss: categorical_crossentropy
PCA: 0.95
No. of components: 269
No. of samples: 28,709

Image size: 48 x 48
Train accuracy: 73.61 %
Test accuracy: 43.90 %
Test Error: 1.62
Layer 1: Relu, 128 Neurons
Layer 2: Relu, 64 Neurons
Layer 3: Softmax, 10 Neurons

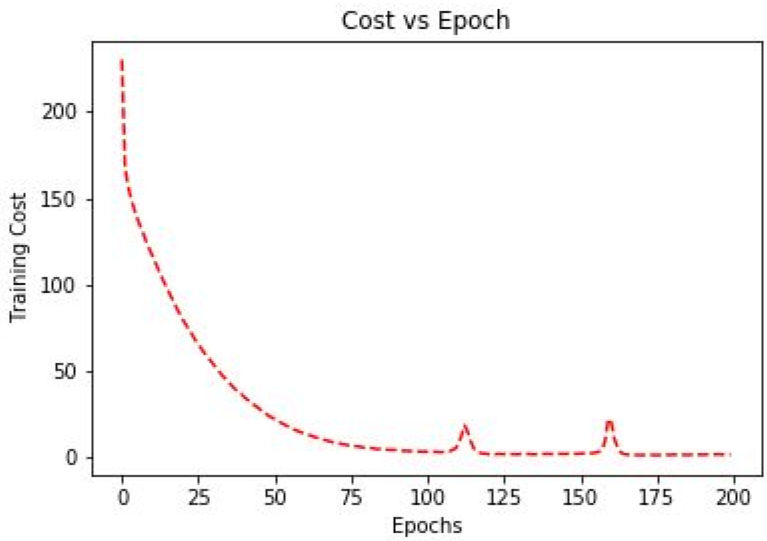

Fig-1: Cost vs Epoch for facial expression datasets

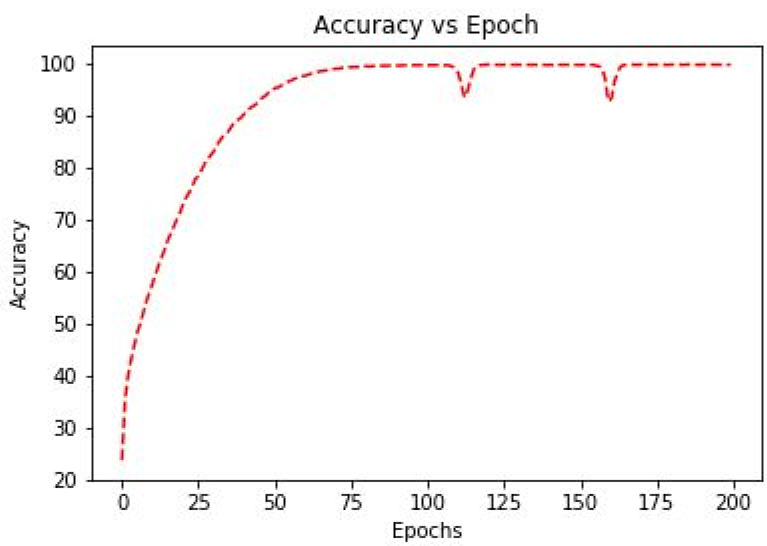

Fig-2: Accuracy vs Epoch for Facial expression datasets

Pros and Cons of PCA:

PCA is used for dimension reduction of the datasets for this projects. Some of its pros and cons relevant to the project are highlighted below:

- **Pros:**
    - **Dimensionality reduction:** It reduces high dimension data samples to low dimensions preserving the important features and removing the unwanted (less important) features. This reduces the data size and leads to faster and efficient computations for small datasets as shown in table 3.

    - **Denoising:** As PCA discards unwanted or less important features, it denoises the sample data.

    - **Better Data separability:** It reduces complexity in images grouping. So, the data is better separated and it leads to better classification model.

- **Cons:**
    - **Computation Complexity:** The complexity of the PCA is $O(\min(p^3, n^3))$ where p is number of features and n is number of data points. So, for the large datasets, the complexity increases and becomes computationally expensive. Though the classification time has decreased for the model designed, preprocessing the samples including PCA takes considerable amount of time because of the relatively large datasets.

    - **Non Linearity:** PCA does not do well in finding nonlinear principal components.

Conclusion:

The 3 layer neural network model performs reasonably well after preprocessing the data and using PCA for small datasets as shown in table 3. However, the performance decreases as the size of the sample datasets increases.

References:

[1] MNIST database of handwritten digits datasets. https://keras.io/datasets/
[2] CIFAR10 small image classification datasets. https://keras.io/datasets/
[3] Facial Expression Recognition Challenge datasets.
https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data
[4] Keras. https://keras.io/

Images:

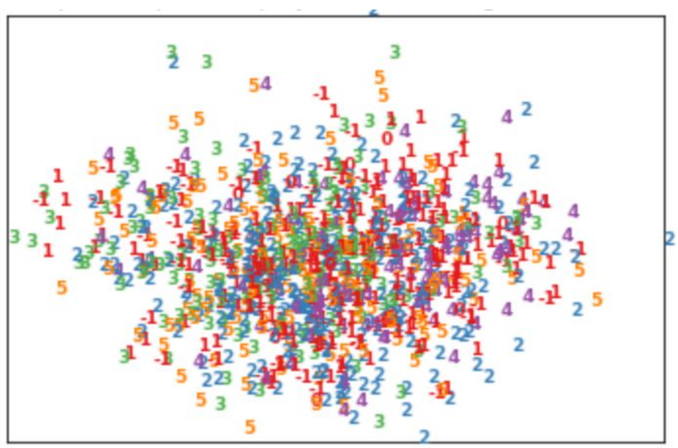

Fig-3: Data visualization for face data sets (1000 samples), PCA(0.95)

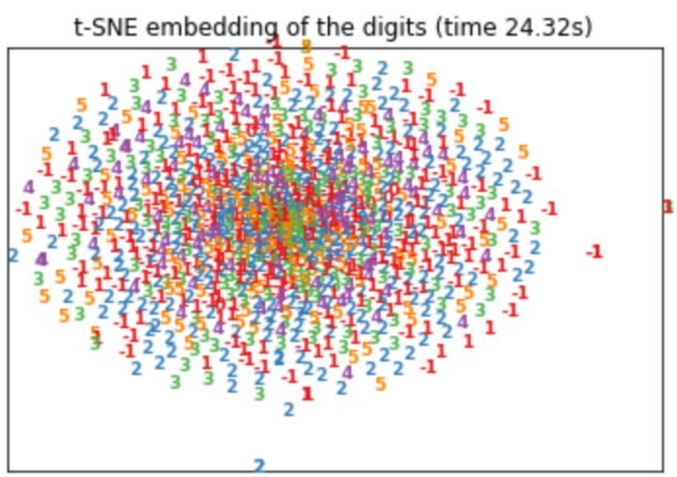

Fig-4: Data visualization for face data sets (1000 samples), t-SNE

Appendix:

Source Code
Datasets from Kaggle
Challenges in Representation Learning: Facial Expression Recognition Challenge
source: [Kaggle Challenge]

```
import csv, sys
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

def load_facedata():
	filename = '../fer2013.csv'
	data = []
	label = []
	w, h = 48, 48
	image = np.zeros((h, w), dtype=np.uint8)
	cnt = 0
	with open(filename, 'r') as f:
	reader = csv.reader(f)
	try:
	for row in reader:
		cnt += 1
		if cnt>1:
		emotion = int(row[0]) - 1
		pixels = list(map(int, row[1].split()))
		pixels_array = np.asarray(pixels)

		data.append(pixels_array)
		label.append(emotion)

	except csv.Error as e:
	sys.exit('file %s, line %d: %s' % (filename, reader.line_num, e))

	return data, label


def face_data():
	data, label = load_facedata()

	X = np.array(data)[:]
	y = np.asarray(label)[:]

	X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
    return (X_train, Y_train), (X_test, Y_test)

import numpy as np
from keras.models import Sequential
from keras.layers.core import Activation
from keras.layers.core import Dense
from keras.utils import np_utils
from keras.datasets import mnist
from keras.datasets import cifar10
from keras import backend as K
K.set_image_dim_ordering('th')
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

def show_graph(epoch, history, title="Cost vs Epoch",xlabel="Epochs",ylabel="Training
Cost",image_name="cost_vs_epochs.png"):
    epochs = range(epoch)

    epochs = np.reshape(epochs,(-1,1))
    history = np.reshape(history,(-1,1)) *100

    plt.plot(epochs, history,"r--")
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.savefig(image_name)
    plt.show()

def read_data(file="mnist", n_classes = 10):
    if file=="mnist":
    (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
    c = 1
    w,h = X_train.shape[1:]
    print("MNIST data loaded.")
    elif file=="cifar10":
    (X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
    c,w,h = X_train.shape[1:]
    print("CIFAR data loaded.")
    elif file=="face_data":
    (X_train, Y_train), (X_test, Y_test) = face_data()
    c,w,h = 1,48,48
    print("Face data loaded.")

    X_train = X_train.reshape(X_train.shape[0], c*w*h)
```

```
    X_test = X_test.reshape(X_test.shape[0], c*w*h)
    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')
    X_train /= 255
    X_test /= 255

    Y_train = np_utils.to_categorical(Y_train, n_classes)
    Y_test = np_utils.to_categorical(Y_test, n_classes)

    return (X_train, Y_train), (X_test, Y_test)

def main(file="mnist", batch_size = 512, epochs = 10):
    print("Starting Network...")
    print("--------------------------------------------------------")
    print("Reading Data sets...")

    num_classes = 7
    (X_train, Y_train), (X_test, Y_test) = read_data(file, n_classes=num_classes)

    #input_shape = X_train.shape[1:]

    X_train = X_train[:10000]
    Y_train = Y_train[:10000]
    X_test = X_test[:]
    Y_test = Y_test[:]

    #print(X_train.shape, X_test.shape)
    scaler = StandardScaler()

    # Fit on training set only.
    scaler.fit(X_train)

    # Apply transform to both the training set and the test set.
    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)

    pca = PCA(0.95)
    pca.fit(X_train)
    print("PCA Components: {0}".format(pca.n_components_))
    input_shape = (pca.n_components_,)
    print("Input shape: ", input_shape)
    X_train = pca.transform(X_train)
    X_test = pca.transform(X_test)

    print("--------------------------------------------------------")
    print("Begin Training...")
```

```
	model = Sequential()
	model.add(Dense(128, activation='relu',input_shape=input_shape))
	model.add(Dense(64, activation='relu'))
	model.add(Dense(num_classes, activation='softmax'))

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

	history = model.fit(X_train, Y_train,
	batch_size=batch_size,
	epochs=epochs,
	verbose=1)

	print("End Training.")
	print("-----------------------------------------------------")
	print("Begin Testing...")

	score = model.evaluate(X_test, Y_test, verbose=0)
	test_loss = score[0]
	test_accuracy = score[1]*100

	print("End Testing.")
	print("-----------------------------------------------------")

	print('Test Error: {0}'.format(test_loss))
	print('Test Accuracy: {0:0.2f} %'.format(test_accuracy))

	show_graph(epochs, history.history['loss'],
	title="Cost vs Epoch",
	xlabel="Epochs",
	ylabel="Training Cost",
	image_name="cost_vs_epochs.png")

	show_graph(epochs, history.history['acc'],
	title="Accuracy vs Epoch",
	xlabel="Epochs",
	ylabel="Accuracy",
	image_name="accuracy_vs_epochs.png")

if __name__=="__main__":
	#main(file="mnist", batch_size = 512, epochs = 50)
	#main(file="cifar10", batch_size = 512, epochs = 20)
	main(file="face_data", batch_size = 512, epochs = 100)
```