DL Assignment No. 01

In [1]:
```
# Tensorflow Test program
import tensorflow as tf
print(tf.__version__)
```

```
2.3.0
```

In [2]:
```
print(tf.reduce_sum(tf.random.normal([1000, 1000])))
```

```
tf.Tensor(17.24144, shape=(), dtype=float32)
```

In [3]:
```
# Keras Test Program
from tensorflow import keras
from keras import datasets
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
train_images.shape, test_images.shape
```

Out[3]:
```
((60000, 28, 28), (10000, 28, 28))
```

In [6]:
```
# Theano test program
import numpy
import theano.tensor as T
from theano import function
x = T.dscalar('x')
y = T.dscalar('y')
z = x + y
f = function([x, y], z)
print(f(5, 7))
```

```
12.0
```

In [5]:
```
# Test program for PyTorch
import torch
print(torch.__version__)
```

```
1.12.1+cpu
```

DL Assignment No. 2

In [1]:

```
# a. Import the necessary packages
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
```

In [2]:

```
# b. Load the training and testing data
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Feature Scaling
x_train = x_train/255
x_test = x_test/255
```

In [3]:

```
# c. Define the network architecture using Keras
model = tf.keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

In [4]:

```
# d. Train the model using SGD
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, validation_data=(x_test, y_test),
                    epochs=10, verbose=3)
```

```
Epoch 1/10
Epoch 2/10
Epoch 3/10
Epoch 4/10
Epoch 5/10
Epoch 6/10
Epoch 7/10
Epoch 8/10
Epoch 9/10
Epoch 10/10
```

In [5]:

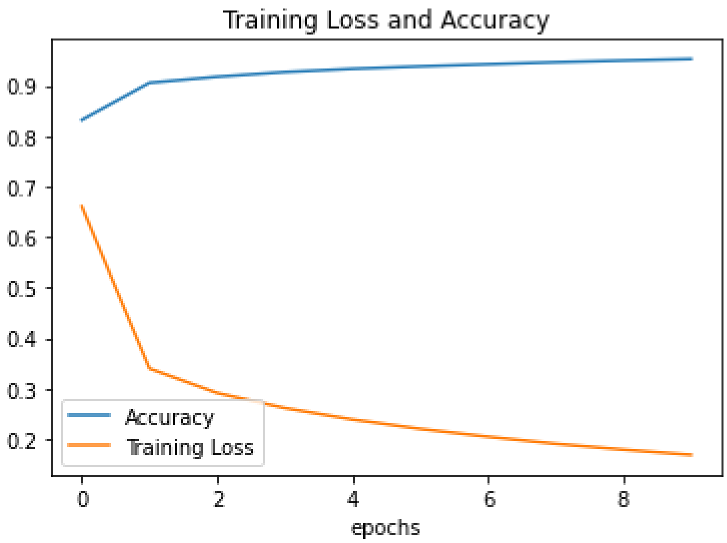
```
# e. Evaluate the network
test_loss, test_acc=model.evaluate(x_test, y_test, verbose=0)
print("Loss =", test_loss)
print("Accuracy =", test_acc)
```

```
Loss = 0.16409531235694885
Accuracy = 0.9509999752044678
```

In [6]:

```
# f. Plot the training loss and accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['loss'])
plt.title('Training Loss and Accuracy')
plt.xlabel('epochs')
plt.legend(['Accuracy', 'Training Loss'], loc='lower left')
```

Out[6]: <matplotlib.legend.Legend at 0x1b270ac7ca0>



DL Assignment No. 03

In [1]:

```
import tensorflow as tf
from tensorflow import keras
```

In [2]:

```
# a. Loading and preprocessing the image data
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Feature Scaling
x_train = x_train/255
x_test = x_test/255
```

In [3]:

```
# b. Defining the model's architecture
model = tf.keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

In [4]:

```
# c. Training the model
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, validation_data=(x_test, y_test),
                    epochs=10, verbose=3)
```

```
Epoch 1/10
Epoch 2/10
Epoch 3/10
Epoch 4/10
Epoch 5/10
Epoch 6/10
Epoch 7/10
Epoch 8/10
Epoch 9/10
Epoch 10/10
```

In [5]:

```
# d. Estimating the model's performance
test_loss, test_acc=model.evaluate(x_test, y_test, verbose=0)
print("Loss =", test_loss)
print("Accuracy =", test_acc)
```

```
Loss = 0.16344955563545227
Accuracy = 0.9534000158309937
```

DL Assignment No. 04

In [1]:
```
# a. Import required libraries
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_s
RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal", "Fraud"]
```

In [2]:
```
# b. Upload / access the dataset
dataset = pd.read_csv("creditcard.csv")
```

In [3]:
```
sc=StandardScaler()
dataset['Time'] = sc.fit_transform(dataset['Time'].values.reshape(-1, 1))
dataset['Amount'] = sc.fit_transform(dataset['Amount'].values.reshape(-1, 1))
```

In [4]:
```
raw_data = dataset.values
# The last element contains if the transaction is normal which is represented by a 0 an
labels = raw_data[:, -1]
# The other data points are the electrocadriogram data
data = raw_data[:, 0:-1]
train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=2021)
```

In [5]:
```
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)
train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)
train_data = tf.cast(train_data, tf.float32)
test_data = tf.cast(test_data, tf.float32)
```

In [6]:
```
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
#creating normal and fraud datasets
normal_train_data = train_data[~train_labels]
normal_test_data = test_data[~test_labels]
fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
```

In [7]:
```
nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1]
encoding_dim = 14
```

```
hidden_dim_1 = int(encoding_dim / 2)
hidden_dim_2=4
learning_rate = 1e-7
```

In [8]:

```
#input Layer
input_layer = tf.keras.layers.Input(shape=(input_dim, ))
# c. Encoder that converts it into latent representation
encoder = tf.keras.layers.Dense(encoding_dim, activation="tanh",
                                activity_regularizer=tf.keras.regularizers.l2(learning_
encoder = tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(encoder)
encoder = tf.keras.layers.Dense(hidden_dim_2,
                                activation=tf.nn.leaky_relu)(encoder)
# d. Decoder networks that convert it back to the original input
decoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(encoder)
decoder = tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encoding_dim, activation='relu')(decoder)
decoder = tf.keras.layers.Dense(input_dim, activation='tanh')(decoder)
# Autoencoder
autoencoder = tf.keras.Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 30)]              0         
_________________________________________________________________
dense (Dense)                (None, 14)                434       
_________________________________________________________________
dropout (Dropout)            (None, 14)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 105       
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 32        
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 35        
_________________________________________________________________
dropout_1 (Dropout)          (None, 7)                 0         
_________________________________________________________________
dense_4 (Dense)              (None, 14)                112       
_________________________________________________________________
dense_5 (Dense)              (None, 30)                450       
=================================================================
Total params: 1,168
Trainable params: 1,168
Non-trainable params: 0
_________________________________________________________________
```

In [9]:

```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",
                                        mode='min', monitor='val_loss',
                                        verbose=2, save_best_only=True)
# define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True)
```

In [10]:

```
# e. Compile the models with Optimizer, Loss, and Evaluation Metrics
autoencoder.compile(metrics=['accuracy'],
                    loss='mean_squared_error',
                    optimizer='adam')
```

In [11]:

```
history = autoencoder.fit(normal_train_data, normal_train_data,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(test_data, test_data),
                    verbose=1,
                    callbacks=[cp, early_stop]
                    ).history
```

```
Epoch 1/50
3530/3554 [============================>.] - ETA: 0s - loss: 0.0046 - accuracy: 0.0549
Epoch 00001: val_loss improved from inf to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 0.0045 - accuracy: 0.05
50 - val_loss: 2.0208e-05 - val_accuracy: 0.0110
Epoch 2/50
3517/3554 [============================>.] - ETA: 0s - loss: 1.9522e-05 - accuracy: 0.06
68
Epoch 00002: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_frau
d.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9519e-05 - accuracy:
0.0666 - val_loss: 2.0128e-05 - val_accuracy: 0.0596
Epoch 3/50
3550/3554 [============================>.] - ETA: 0s - loss: 1.9525e-05 - accuracy: 0.06
39
Epoch 00003: val_loss did not improve from 0.00002
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9523e-05 - accuracy:
0.0640 - val_loss: 2.0212e-05 - val_accuracy: 0.0371
Epoch 4/50
3527/3554 [============================>.] - ETA: 0s - loss: 1.9498e-05 - accuracy: 0.06
14
Epoch 00004: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_frau
d.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9492e-05 - accuracy:
0.0613 - val_loss: 1.9898e-05 - val_accuracy: 0.1301
Epoch 5/50
3505/3554 [============================>.] - ETA: 0s - loss: 1.8720e-05 - accuracy: 0.12
59
Epoch 00005: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_frau
d.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.8719e-05 - accuracy:
0.1267 - val_loss: 1.8257e-05 - val_accuracy: 0.2170
Epoch 6/50
3544/3554 [============================>.] - ETA: 0s - loss: 1.8231e-05 - accuracy: 0.16
95
Epoch 00006: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_frau
d.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.8225e-05 - accuracy:
0.1695 - val_loss: 1.7669e-05 - val_accuracy: 0.2684
Epoch 7/50
3517/3554 [============================>.] - ETA: 0s - loss: 1.7663e-05 - accuracy: 0.18
53
Epoch 00007: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_frau
d.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7654e-05 - accuracy:
0.1855 - val_loss: 1.7358e-05 - val_accuracy: 0.2877
```

```
Epoch 8/50
3537/3554 [============================>.] - ETA: 0s - loss: 1.7321e-05 - accuracy: 0.21
27
Epoch 00008: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_frau
d.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7314e-05 - accuracy:
0.2128 - val_loss: 1.7163e-05 - val_accuracy: 0.2535
Epoch 9/50
3527/3554 [============================>.] - ETA: 0s - loss: 1.7156e-05 - accuracy: 0.22
30
Epoch 00009: val_loss did not improve from 0.00002
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7161e-05 - accuracy:
0.2229 - val_loss: 1.7260e-05 - val_accuracy: 0.2524
Epoch 10/50
3504/3554 [============================>.] - ETA: 0s - loss: 1.7026e-05 - accuracy: 0.23
59
Epoch 00010: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_frau
d.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7026e-05 - accuracy:
0.2362 - val_loss: 1.7045e-05 - val_accuracy: 0.3440
Epoch 11/50
3519/3554 [============================>.] - ETA: 0s - loss: 1.6972e-05 - accuracy: 0.24
43
Epoch 00011: val_loss did not improve from 0.00002
Restoring model weights from the end of the best epoch.
3554/3554 [==============================] - 4s 1ms/step - loss: 1.6964e-05 - accuracy:
0.2443 - val_loss: 1.7348e-05 - val_accuracy: 0.2599
Epoch 00011: early stopping
```

DL Assignment No. 05

In [1]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import matplotlib.pylab as pylab
import numpy as np
%matplotlib inline
```

In [2]:

```
import re
sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules called a program.
People create programs to direct processes.
In effect, we conjure the spirits of the computer with our spells."""
```

In [3]:

```
# remove special characters
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)
# remove 1 letter words
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()
# lower all characters
sentences = sentences.lower()
```

In [4]:

```
words = sentences.split()
vocab = set(words)
vocab_size = len(vocab)
embed_dim = 10
context_size = 2
```

In [5]:

```
word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}
```

In [6]:

```
data = []
for i in range(2, len(words) - 2):
    context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
    target = words[i]
    data.append((context, target))
print(data[:5])
```

```
[(['we', 'are', 'to', 'study'], 'about'), (['are', 'about', 'study', 'the'], 'to'), (['a
bout', 'to', 'the', 'idea'], 'study'), (['to', 'study', 'idea', 'of'], 'the'), (['stud
y', 'the', 'of', 'computational'], 'idea')]
```

In [7]:

```
embeddings = np.random.random_sample((vocab_size, embed_dim))
```

In [8]:

```
def linear(m, theta):
    w = theta
    return m.dot(w)
```

In [9]:

```
def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum()/len(out)
def log_softmax_crossentropy_with_logits(logits,target):
    out = np.zeros_like(logits)
    out[np.arange(len(logits)),target] = 1
    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)
    return (- out + softmax) / logits.shape[0]
```

In [10]:

```
def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)
    return m, n, o
```

In [11]:

```
def backward(preds, theta, target_idxs):
    m, n, o = preds
    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)
    return dw
```

In [12]:

```
def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta
```

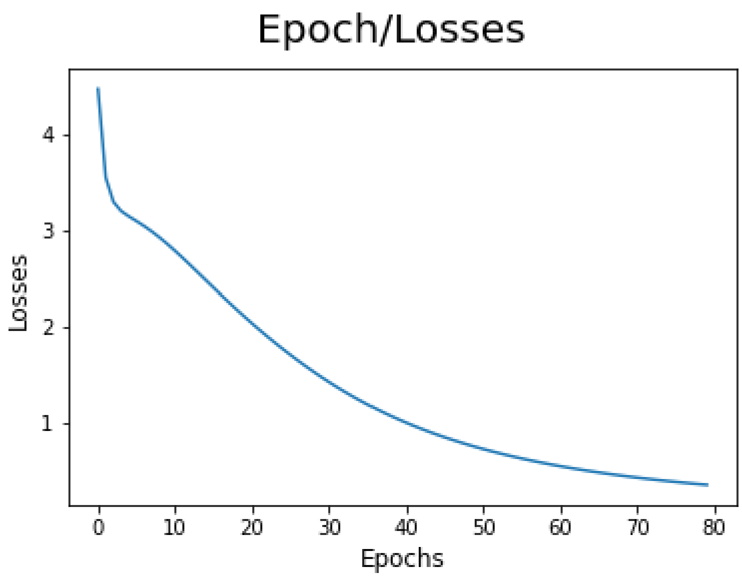
In [13]:

```
theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim,
vocab_size))
epoch_losses = {}
for epoch in range(80):
    losses = []
    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)
        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)
        losses.append(loss)
        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)
    epoch_losses[epoch] = losses
```

In [14]:

```
ix = np.arange(0,80)
fig = plt.figure()
fig.suptitle('Epoch/Losses', fontsize=20)
plt.plot(ix,[epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Losses', fontsize=12)
```

Out[14]: Text(0, 0.5, 'Losses')



In [15]:
```
def predict(words):
    context_idxs = np.array([word_to_ix[w] for w in words])
    preds = forward(context_idxs, theta)
    word = ix_to_word[np.argmax(preds[-1])]
    return word
# (['we', 'are', 'to', 'study'], 'about')
predict(['we', 'are', 'to', 'study'])
```

Out[15]: 'about'

In [16]:
```
def accuracy():
    wrong = 0
    for context, target in data:
        if(predict(context) != target):
            wrong += 1
    return (1 - (wrong / len(data)))
accuracy()
```

Out[16]: 1.0

In [17]:
```
predict(['processes', 'manipulate', 'things', 'study'])
```

Out[17]: 'the'

Input Image:

DL Assignment No. 06

In [4]:

```
# example of using a pre-trained model as a classifier
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16

# load an image from file
image = load_img('goldenretriever1.jpg', target_size=(224, 224))

# convert the image pixels to a numpy array
image = img_to_array(image)

# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1],
image.shape[2]))

# prepare the image for the VGG model
image = preprocess_input(image)

# load the model
model = VGG16()

# predict the probability across all output classes
yhat = model.predict(image)

# convert the probabilities to class labels
label = decode_predictions(yhat)

# retrieve the most likely result, e.g. highest probability
label = label[0][0]

# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
golden_retriever (97.08%)
```

LIST OF LAB EXPERIMENTS

ACADEMIC YEAR: 2022 - 23

DEPARTMENT:INFORMATION TECHNOLOGY

CLASS:B.E. **SEMESTER:I**

SUBJECT: 414447: Lab Practice IV

LAB EXPT.NO	PROBLEMSTATEMENT
1.	**Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch. Document the distinct features and functionality of the packages.**
2.	**Implementing Feedforward neural networks with Keras and TensorFlow a. Import the necessary packages b. Load the training and testing data (MNIST/CIFAR10) c. Define the network architecture using Keras d. Train the model using SGD e. Evaluate the network f. Plot the training loss and accuracy**
3.	**Build the Image classification model by dividing the model into following 4 stages: a. Loading and preprocessing the image data b. Defining the model's architecture c. Training the model d. Estimating the model's performance**
4.	**Use Autoencoder to implement anomaly detection. Build the model by using: a. Import required libraries b. Upload / access the dataset c. Encoder converts it into latent representation d. Decoder networks convert it back to the original input e. Compile the models with Optimizer, Loss, and Evaluation Metrics**
5.	**Implement the Continuous Bag of Words (CBOW) Model. Stages can be: a. Data preparation b. Generate training data c. Train model d. Output**
6.	**Object detection using Transfer Learning of CNN architectures a. Load in a pre-trained CNN model trained on a large dataset b. Freeze parameters (weights) in model's lower convolutional layers c. Add custom classifier with several layers of trainable parameters to model d. Train classifier layers on training data available for task e. Fine-tune hyper parameters and unfreeze more layers as needed**