Deep Learning Tutorial Report

2nd year of engineering in advanced technology

Produced by: Khlifi Oussema 2TA3

Deep learning tutorial 1: MNIST handwritten digits Classification with MLPs.

I- Introduction:

The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9.

The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9.

1) Packages used in this tutorial:

Keras:

Keras is high-level API wrapper for the low-level API, capable of running on top of TensorFlow. Keras High-Level API handles the way we make models, defining layers, or set up multiple input-output models. In this level, Keras also compiles our model with loss and optimizer functions, training process with fit function.

Statement to import the package:

```
import keras as k
```

Tensorflow:

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of Tensorflow.

PIL (Python imaging library):

The Python Imaging Library adds image processing capabilities to your Python interpreter.

This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

2) Code implementation:

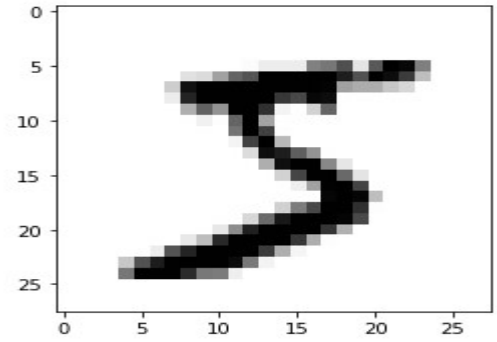
Package importation:

```
import keras as k #import la bibliotheque Keras
import tensorflow as tf #importaion de la bibliotheque tensorflow w
hich is a Python library for fast numerical computing created and r
eleased by Google.It is a foundation library that can be used to cr
eate Deep Learning models directly or by using wrapper libraries th
at simplify the process built on top of TensorFlow.
 #it's a linear algebra package, we need it because we play with im
ages which they are matrices
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from __future__ import print_function
from keras.datasets import mnist
from keras.models import *
from keras.layers import *
from keras.utils import *
```

Loading the data:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train.shape
#print(x_train)
x_train[0]
img = Image.fromarray(x_train[0])
plt.imshow(x_train[0], cmap=plt.cm.binary)
img = Image.fromarray(x_train[0])
img.show()
print(y_train.shape)
y_train[1]
```

```
(60000,)
0
```



```
img = Image.fromarray(x_train[0])
img.show()
y_train.shape
y_train[1]
```

Output :

0

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
print(x_train)
```

Output :

```
[[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]


 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
```

[0. 0. 0. ... 0. 0. 0.]

...

[0. 0. 0. ... 0. 0. 0.]

[0. 0. 0. ... 0. 0. 0.]

[0. 0. 0. ... 0. 0. 0.]]]

```
#x_train = x_train.reshape((x_train.shape[0],x_train.shape[1]*x_train.shape[-1]))
print(x_train.shape,x_train.shape[1]*x_train.shape[-1])
x_test = x_test.reshape((x_test.shape[0],x_test.shape[1]*x_test.shape[-1]))
Output :
(60000, 28, 28) 784
labels = [0,1,2,3,4,5,6,7,8,9]
labels.index(y_train[1])
Output :
0
num_classes = 10
num_features = x_train.shape[1]
print("the number of the features are {}".format(num_features))
n_samples = x_train.shape[0]
print("the number of the samples are {}".format(n_samples))
train_labels = tf.keras.utils.to_categorical(y_train, num_classes=None ,dtype='float32')
test_labels = tf.keras.utils.to_categorical(y_test, num_classes=None ,dtype='float32')
print(y_train.shape)
 Output :
the number of the features are 28
the number of the samples are 60000
(60000,)

T_train = np.zeros(shape = (n_samples,num_classes))
for i in range(n_samples):
   ind = labels.index(y_train[i])
   T_train[i,ind] = 1
T_test = np.zeros(shape = (y_test.shape[0],num_classes))
for i in range(y_test.shape[0]):
   ind = labels.index(y_test[i])
   T_test[i,ind] = 1
print(T_test.shape)
Output:
```

```
(10000, 10)
def logistic_regression(num_classes,num_features):
   model = Sequential()
   model.add(Dense(units = 10,input_dim = num_features, activation = 'soft
max'))
   return model
def lr(num_classes,num_features):
    Inp = Input(shape = (num_features,))
    x = Dense(units = 10,activation = 'softmax')(Inp)
    model = Model(inputs= [Inp],outputs = [x])
    return model
def mlp(num_classes,num_features):
    Inp = Input(shape = (num_features,))
    x1 = Dense(units = 10,activation = 'relu')(Inp)
    x = Dense(units = 10,activation = 'softmax')(x1)
    model = Model(inputs= [Inp],outputs = [x])
    return model
model = mlp(num_classes,num_features)
#categorical_crossentropy
model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['
accuracy'])
history = model.fit(x_train, T_train, validation_data = (x_test,T_test), b
atch_size= 100, epochs=30)


x_train[10].shape
#pred = model.predict(x_train[10].reshape(1,784))
pred = model.predict(x_test)
pred[10]
np.argmax(pred[10])
pred_classes = [np.argmax(x) for x in pred]
orig = [np.argmax(x) for x in T_test]
print("Predicted values : ",pred_classes[:20])
print("Original values : ",orig[:20])
```

Output:


```
x_train[10].shape
#pred = model.predict(x_train[10].reshape(1,784))
pred = model.predict(x_test)
pred[10]
np.argmax(pred[10])
pred_classes = [np.argmax(x) for x in pred]
orig = [np.argmax(x) for x in T_test]
print("Predicted values : ",pred_classes[:20])
print("Original values : ",orig[:20])
```

```
ERR = []
for i in range(T_test.shape[0]):
    if pred_classes[i] != orig[i]:
        ERR.append(i)
        len(ERR)
print(ERR)
```

Output:

liste=[x for x in range(10000)]

Deep learning tutorial 2: Larger Convolution Neural Network for MNIST

I-Introduction:

Now that we have seen how to create a simple CNN, let us look at a model capable of close to State-of-the-art results.

We import classes and function then load and prepare the data the same as in the previous CNN example.

Code:

```
import keras as k #import the keras packge which has the models , layers and datasets.


import tensorflow as tf #import of the tensorflow package to create deep learnong models


import numpy as np # we need numpy package because we will deal with matrices and arrays.



from PIL import Image
#PIL or Python images library. It's a library made of class Image and which allows us to deal with images.


import matplotlib.pyplot as plt
#matplotlib.pyplot is a package that contains functions  for plotting the data (represent graphically the data)
```

```
from keras.datasets import mnist
from keras.models import *
from keras.layers import *
from keras.utils import *
#Here we are importing the datasets , models, and layers needed for training and testing purposes.
```

```
img_rows, img_cols = 28, 28 # these are the dimensions of the images.
(x_train, y_train), (x_test, y_test) = mnist.load_data() # the data, split between train and test sets
#it returns the default image data format convention. it Returns a string, either 'channels_first' or 'channels_last.
if k.backend.image_data_format() == "channels_first" :
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols) #here we are reshape it to be aduate for the test.
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    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)
print(x_train.shape)
print(y_train.shape)
#mnist.load_data() returns: uint8 NumPy array of grayscale image data. Pixel values range from 0 to 255.
#x_train: uint8 NumPy array of grayscale image data with shapes (60000, 28, 28), containing the training data. Pixel values range from 0 to 255.
#y_train: uint8 NumPy array of digit labels (integers in range 0-9) with shape (60000,) for the training data.
#x_test: uint8 NumPy array of grayscale image data with shapes (10000, 28, 28), containing the test data. Pixel values range from 0 to 255.
#y_test: uint8 NumPy array of digit labels (integers in range 0-9) with shape (10000,) for the test data.
```

```
(60000, 28, 28, 1)
```

```
(60000,)
```

```
num_classes = 10 #we're fixed the number of classes.
num_features = x_train.shape[1]
print("the number of the features are {}".format(num_features))
n_samples = x_train.shape[0]
print("the number of the samples are {}".format(n_samples))
train_labels = tf.keras.utils.to_categorical(y_train, num_classes=None ,dtype='float32')
test_labels = tf.keras.utils.to_categorical(y_test, num_classes=None ,dtype='float32')
print(y_train.shape)
```

```
the number of the features are 28
the number of the samples are 60000
(60000,)
```

```
labels = [0,1,2,3,4,5,6,7,8,9] # we have 10 labels in the output file.
labels.index(y_train[1])
```

```
0
```

```
T_train = np.zeros(shape = (n_samples,num_classes))
for i in range(n_samples):
    ind = labels.index(y_train[i])
    T_train[i,ind] = 1
T_test = np.zeros(shape = (y_test.shape[0],num_classes))
for i in range(y_test.shape[0]):
    ind = labels.index(y_test[i])
    T_test[i,ind] = 1
print(T_test.shape)
```

```
(10000, 10)
```

```
def logistic_regression(num_classes,num_features):
    model = Sequential()
    model.add(Dense(units = 10,input_dim = num_features, activation = 'softmax'))
    return model
```

```
def lr(num_classes,num_features):
    Inp = Input(shape = (num_features,))
    x = Dense(units = 10,activation = 'softmax')(Inp)
    model = Model(inputs= [Inp],outputs = [x])
    return model
```

```
def mlp(num_classes,num_features):
    Inp = Input(shape = (num_features,))
    x1 = Dense(units = 10,activation = 'relu')(Inp)
    x = Dense(units = 10,activation = 'softmax')(x1)
    model = Model(inputs= [Inp],outputs = [x])
    return model
```

```
Mod1 = mlp(num_classes,num_features)
```

```python
#building the appropriate model for the training and testing. we transforme the images into vectors with this whole of process.
Mod1 = Sequential()
Mod1.add(Conv2D(32, kernel_size=(3, 3),
activation='relu',input_shape=input_shape))
Mod1.add(MaxPooling2D(pool_size=(2, 2)))
Mod1.add(Conv2D(64, (3, 3), activation='relu'))
Mod1.add(MaxPooling2D(pool_size=(2, 2)))
#Mod1.add(Dropout(0.25))
Mod1.add(Flatten())
Mod1.add(Dense(128, activation='relu'))
Mod1.add(Dropout(0.5))
Mod1.add(Dense(128, activation='softmax'))
```

Pyth

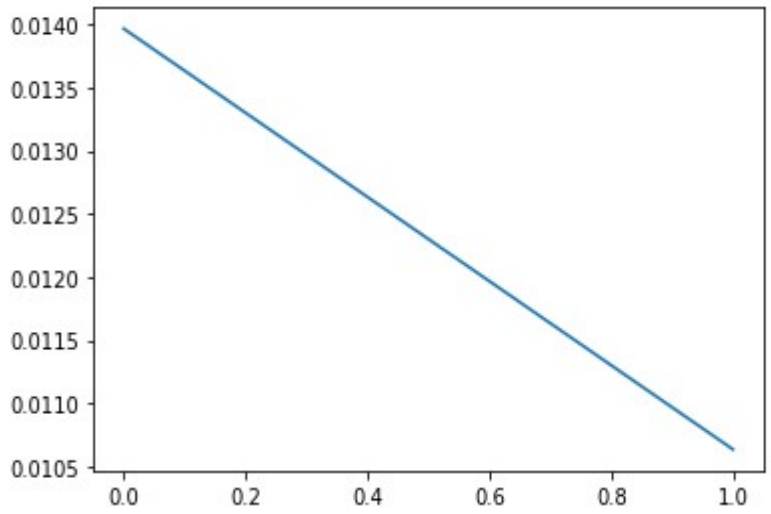
```python
Mod1.compile(loss='mean_squared_error',
optimizer='sgd',
metrics=['accuracy']) #Computes the mean squared error between x_test and y_test.
history = Mod1.fit(x_train,y_train,validation_data = (x_test,y_test),batch_size=60,epochs = 2) #training the model with the history of the training data.
```

Pyth

```
Epoch 1/2
1000/1000 [==============================] - 66s 66ms/step - loss: 28.1227 - accuracy: 0.0150 - val_loss: 28.0658 - val_accuracy: 0.0315
Epoch 2/2
1000/1000 [==============================] - 49s 49ms/step - loss: 28.1225 - accuracy: 0.0137 - val_loss: 28.0654 - val_accuracy: 0.0338
```

```python
plt.plot(history.history['accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7fe0f8152610>]
```



Deep learning tutorial 3: Text Classification using Neural Networks

I-introduction:

By understanding how chat bots work is important. A fundamental piece of machinery inside a chat bot is the text classifier. Let us look at the inner workings of an artificial neural network (ANN) for text classification

II. Steps of the program

Let us examine our text classifier one section at a time. We will take the following steps:

1. Refer to libraries we need
2. Provide training data
3. Organize our data
4. Iterate: code + test the algorithms needed

III- Code implantation:

```
#we are importing the main packages to work on this project which is to
identify the type of the text file .
#is it about business , sports, politics or other.
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import string
from keras.models import Model, Input
from keras.layers import *
import text_cleaners as txt
#we are uploading text_cleaners file which is already scripted.
# Import some text featrues predefined in sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
#Tfidfvectorizer Converts a collection of raw documents to a matrix of TF-
IDF features.
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split #splits the training
and the testing data.
from sklearn.feature_extraction.text import CountVectorizer
#CountVectorizer is used  to convert a collection of text documents to a
matrix of token counts.
from sklearn.linear_model import LogisticRegression
#Logistic regression is another powerful supervised ML algorithm used for
binary
# classification problems (when target is categorical). The best way to
think about
# logistic regression is that it is a linear regression but for
classification problems.
 #%run text_cleaners.py

data_path = 'data/'
```

```
dframe = pd.read_csv(data_path + "news.csv", sep=',', encoding = "ISO-
8859-1",
 error_bad_lines = False )
len(dframe)
dframe[:20]
#the 20 first columns are talking about business
```

	news	type
0	China had role in Yukos split-up\n \n China le...	business
1	Oil rebounds from weather effect\n \n Oil pric...	business
2	Indonesia 'declines debt freeze'\n \n Indonesi...	business
3	$1m payoff for former Shell boss\n \n Shell is...	business
4	US bank in $515m SEC settlement\n \n Five Bank...	business
5	Verizon 'seals takeover of MCI'\n \n Verizon h...	business
6	Parmalat boasts doubled profits\n \n Parmalat,...	business
7	US seeks new $280bn smoker ruling\n \n The US ...	business
8	Steel firm 'to cut' 45,000 jobs\n \n Mittal St...	business
9	Cars pull down US retail figures\n \n US retai...	business
10	Singapore growth at 8.1% in 2004\n \n Singapor...	business
11	UK bank seals South Korean deal\n \n UK-based ...	business
12	ECB holds rates amid growth fears\n \n The Eur...	business
13	Rank 'set to sell off film unit'\n \n Leisure ...	business
14	US adds more jobs than expected\n \n The US ec...	business
15	House prices show slight increase\n \n Prices ...	business
16	Pension hitch for long-living men\n \n Male li...	business
17	Asian quake hits European shares\n \n Shares i...	business
18	Honda wins China copyright ruling\n \n Japan's...	business
19	Bank set to leave rates on hold\n \n UK intere...	business

```
#list_of_words = dframe['Word'].values.tolist() ;
print(list_of_words[:10])
types = dframe['type'].values.tolist()
news = dframe['news'].values.tolist()
print(news[1])
print(set(types))
#from nltk.corpus import stopwords
#stopW = set(stopwords.words('english'))
from nltk import sent_tokenize
```

```
Oil rebounds from weather effect

 Oil prices recovered in Asian trade on Tuesday, after falling in New York on milder winter weather across the US.

 With winter temperatures staying relatively high in the northern US, a barrel of light crude ended Monday down $1.33 to $42.12. However crude prices have rebounded in Asia, rising to $42.30 a barrel for February delivery. In London, trading of Brent crude was suspended for a public holiday, but the price fell to $39.20 in the Far East.

 With milder temperatures expected to continue in the northern parts of the US over the next few days at least, analysts have said the price of oil may fall further - even if the decline was only temporary. "Weather has been the Achilles' heel of this market," said ABN AMRO analyst John Brady. "But it is winter in the northeast. Eventually we'll get another cold blast." Despite a fall of more than $12 a barrel from the record highs reached in late October, the price of crude oil remains almost 30% higher than year-ago levels. Prices rose last week after militant attacks in Riyadh, the capital of Saudi Arabia, briefly renewed fears that the supply chain might be broken in the world's leading crude exporter. "The market was panicked but fears essentially evaporated... since there was no follow-up," said Deborah White, senior economist for energy at SG Securities in Paris.

{'politics', 'tech', 'business', 'sport', 'entertainment'}
```

```
custom_stop_words =
['a','an','the','this','that','am','is','are','has','had','was','were']
cleaned_sent = txt.remove_stop(news,custom_stop_words)
cleaned_sent[5]
news[5]
# list of ponctuation that will be removed from texts
print(list(string.punctuation))
pnt = list(string.punctuation)
pnt.remove(':') # we kept the ':' in this example
Sent_without_pont = txt.removePonct(cleaned_sent,pnt = pnt)
# The list of categories
TYPES = list(set(types))
print(TYPES)
```

output:

```
['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.',
'/', ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`',
'{', '|', '}', '~'] ['politics', 'tech', 'business', 'sport',
'entertainment']
```

```
y = []
for i in range(len(types)):
 if(types[i] == TYPES[0]):
  y.append([1,0,0,0,0])
 elif(types[i] == TYPES[1]):
  y.append([0,1,0,0,0])
 elif(types[i] == TYPES[2]):
   y.append([0,0,1,0,0])
 elif(types[i] == TYPES[3]):
   y.append([0,0,0,1,0])
 elif(types[i] == TYPES[4]):
  y.append([0,0,0,0,1])
  Y = np.asarray(y)
print(Y.shape)
#Y is matrix with 896 lines and 5 columns.
Output:
(896, 5)
```

```
vectorizer = TfidfVectorizer(min_df=1, max_df=0.9)
from sklearn.model_selection import train_test_split
sent_train, sent_test, y_train, y_test =
train_test_split(Sent_without_pont, y, test_size=0.3,random_state=42)
vectorizer = CountVectorizer()
vectorizer.fit(sent_train)
X_train = vectorizer.fit_transform(sent_train).toarray() # from documenets
to tf-idf representation for training.
X_test = vectorizer.transform(sent_test).toarray()
X = news
print(X_train.shape)
num_labels = len(TYPES)
vocab_size = X_train.shape[1]
Output:
(1557, 29075)
```

```
# we are preparing for the model and the data to train and test.
inp1 = Input(shape = (vocab_size,))
x = Dense(num_labels,activation = 'softmax')(inp1)
M_lr = Model(inp1,x)
print(X_test.shape)

y_train=np.array(y_train)
y_test=np.array(y_test)
output:
(668, 29075)
```

```
Inp = Input(shape = (vocab_size,))
x1 = Dense(units = 100,activation = 'relu')(Inp)
x = Dense(num_labels,activation = 'softmax')(x1)
M_mlp = Model(Inp, x)
M_mlp.compile(optimizer="adamax", loss="mean_squared_error",
metrics=["accuracy"])
history = M_mlp.fit(X_train, y_train, validation_data
=(X_test,y_test),batch_size=50, epochs=2 ,verbose=1)
#we are training the data
```

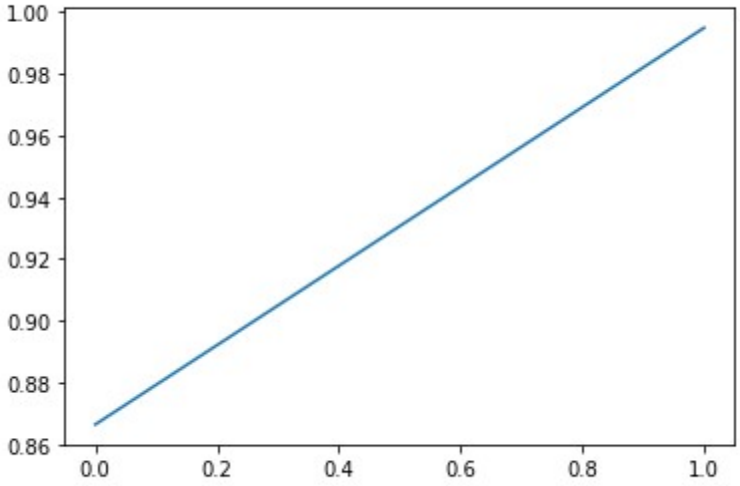
```
Epoch 1/2
32/32 [==============================] - 3s 70ms/step - loss: 0.0576 - accuracy: 0.8664 - val_loss: 0.0199 - val_accuracy: 0.9626
Epoch 2/2
32/32 [==============================] - 2s 57ms/step - loss: 0.0057 - accuracy: 0.9949 - val_loss: 0.0144 - val_accuracy: 0.9611
```

```
pred = M_mlp.predict(X_test)
np.argmax(pred[4])
pred_classes = [TYPES[np.argmax(x)] for x in pred]
orig = [TYPES[np.argmax(x)] for x in y_test]
print("Predicted values : ",pred_classes[0:100])
print("Original values : ",orig[0:100])
#we predict the original values and we obtain 96.26% of confidence which
is a good result.
```

```
Predicted values :  ['business', 'business', 'sport', 'business', 'politics', 'sport', 'sport', 'politics', 'entertainment', 'politics', 'business',
'entertainment', 'sport', 'business', 'politics', 'business', 'tech', 'entertainment', 'business', 'entertainment', 'business', 'sport', 'business',
'entertainment', 'entertainment', 'sport', 'entertainment', 'sport', 'politics', 'sport', 'sport', 'politics', 'tech', 'entertainment', 'business', 'business',
'business', 'sport', 'business', 'sport', 'tech', 'politics', 'business', 'sport', 'tech', 'tech', 'tech', 'business', 'business', 'tech', 'entertainment',
'politics', 'politics', 'business', 'entertainment', 'politics', 'politics', 'politics', 'business', 'business', 'entertainment', 'entertainment',
'entertainment', 'business', 'sport', 'entertainment', 'business', 'business', 'politics', 'sport', 'politics', 'politics', 'tech', 'sport', 'business',
'sport', 'sport', 'entertainment', 'business', 'business', 'sport', 'sport', 'business', 'entertainment', 'entertainment', 'sport', 'sport', 'sport',
'entertainment', 'sport', 'sport', 'business', 'politics', 'business', 'sport', 'business', 'entertainment', 'entertainment', 'business', 'politics']
Original values :  ['business', 'business', 'sport', 'business', 'politics', 'sport', 'sport', 'politics', 'entertainment', 'politics', 'business',
'entertainment', 'sport', 'business', 'entertainment', 'business', 'tech', 'entertainment', 'business', 'entertainment', 'business', 'sport', 'business',
'entertainment', 'business', 'sport', 'entertainment', 'sport', 'politics', 'sport', 'sport', 'politics', 'tech', 'entertainment', 'business', 'business',
'business', 'sport', 'tech', 'sport', 'tech', 'politics', 'business', 'sport', 'tech', 'tech', 'tech', 'business', 'business', 'tech', 'entertainment',
'business', 'politics', 'business', 'entertainment', 'politics', 'politics', 'politics', 'business', 'business', 'business', 'entertainment', 'entertainment',
'business', 'sport', 'entertainment', 'business', 'business', 'politics', 'sport', 'politics', 'politics', 'tech', 'sport', 'business', 'sport', 'sport',
'entertainment', 'business', 'business', 'sport', 'sport', 'business', 'entertainment', 'entertainment', 'sport', 'sport', 'sport', 'entertainment', 'sport',
'sport', 'business', 'politics', 'business', 'sport', 'business', 'entertainment', 'entertainment', 'business', 'politics']
```

```
plt.plot(history.history['accuracy']) # plot accuracy histogram of the
prediction.
```

```
[<matplotlib.lines.Line2D at 0x7f0574b74150>]
```



Deep learning tutorial 4: Cifar Object Recognition in Images using MLP and CNN.

I- Introduction:

CIFAR is an acronym that stands for the Canadian Institute For Advanced Research and the CIFAR-10 dataset was developed along with the CIFAR-100 dataset by researchers at the CIFAR institute.

The dataset is comprised of 60,000 32×32 pixel color photographs of objects from 10 classes, such as frogs, birds, cats, ships, etc. The class labels and their standard associated integer values are listed below.

0: airplane

1: automobile

2: bird

3: cat

4: deer

5: dog

6: frog

7: horse

8: ship

9: truck

These are very small images, much smaller than a typical photograph, and the dataset was intended for computer vision research.

CIFAR-10 is a well-understood dataset and widely used for benchmarking computer vision algorithms in the field of machine learning. The problem is "solved." It is relatively straightforward to achieve 80% classification accuracy. Top performance on the problem is achieved by deep learning convolution neural networks with a classification accuracy above 90% on the test dataset.

The example below loads the CIFAR-10 dataset using the Keras API and creates a plot of the first nine images in the training dataset.

II. Steps of the program

Let us examine our recognition in images at a time. We will take the following steps:

1. Download and process your dataset

2. Bring your data to a format acceptable by the CNN

3. Build the model

4. Train and test the model

III. Code implementation:

```
import numpy as np

from keras.datasets import cifar10 #It allows us to load the dataset and split it to data to train and other to test.

from keras.utils.np_utils import to_categorical

from keras.layers import Dense, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras.models import Sequential

from keras.layers import Activation, Dropout, Flatten

from keras.optimizers import gradient_descent_v2
```

```
from tensorflow.keras.optimizers import SGD

import matplotlib.pyplot as plt

(X_train, y_train), (X_test, y_test) = cifar10.load_data() #we are loading the data.
```

Output:

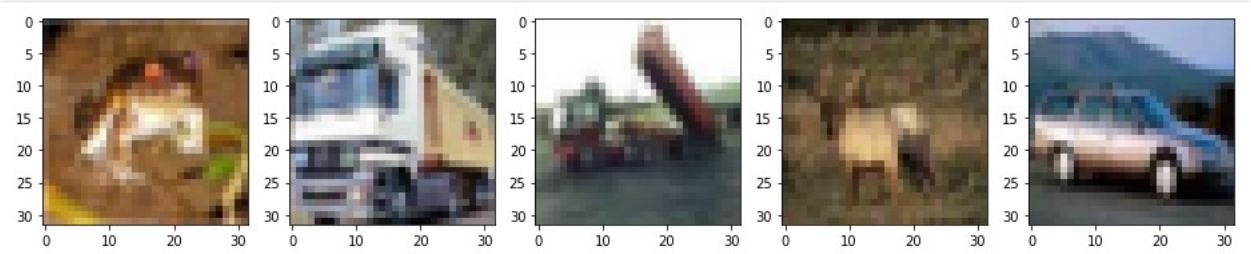
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

170500096/170498071 [==============================] - 2s 0us/step

170508288/170498071 [==============================] - 2s 0us/step
```

```
#to know the shape of the arrays of the training and the testing data.

print("Shape of training data:")

print(X_train.shape)

print(y_train.shape)

print("Shape of test data:")

print(X_test.shape)

print(y_test.shape)
```

output:

```
Shape of training data:

(50000, 32, 32, 3)

(50000, 1)

Shape of test data:

(10000, 32, 32, 3)

(10000, 1)
```

```
cifar_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

f, axarr = plt.subplots(1, 5)

f.set_size_inches(16, 6)

for i in range(5):

    img = X_train[i]

    axarr[i].imshow(img)

plt.show()
```



```
y_train = to_categorical(y_train, num_classes=10)

y_test = to_categorical(y_test, num_classes=10)

# Transform images from (32,32,3) to 3072-dimensional vectors (32*32*3)

X_train = np.reshape(X_train,(50000,3072))

X_test = np.reshape(X_test,(10000,3072))

X_train = X_train.astype('float32')

X_test = X_test.astype('float32')

# Normalization of pixel values (to [0-1] range)

X_train /= 255
```

```
X_test /= 255
```

```
model = Sequential()
model.add(Dense(256, activation='relu', input_dim=3072))
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy'])
```

output:

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:102:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

```
history = model.fit(X_train,y_train, epochs=15, batch_size=32, validation_split=0.2)
```

Output exceeds the size limit. Open the full output data in a text editor

```
Epoch 1/15
1250/1250 [==============================] - 12s 9ms/step - loss: 1.8296 - accuracy:
0.3360 - val_loss: 1.7223 - val_accuracy: 0.3806
Epoch 2/15
1250/1250 [==============================] - 12s 9ms/step - loss: 1.6697 - accuracy:
0.4019 - val_loss: 1.6542 - val_accuracy: 0.4156
```

Epoch 3/15

1250/1250 [==========] - 11s 9ms/step - loss: 1.5956 - accuracy: 0.4271 - val_loss: 1.6188 - val_accuracy: 0.4244

Epoch 4/15

1250/1250 [==========] - 11s 9ms/step - loss: 1.5486 - accuracy: 0.4446 - val_loss: 1.5686 - val_accuracy: 0.4437

Epoch 5/15

1250/1250 [==========] - 12s 10ms/step - loss: 1.5106 - accuracy: 0.4609 - val_loss: 1.5647 - val_accuracy: 0.4470

Epoch 6/15

1250/1250 [==========] - 12s 10ms/step - loss: 1.4805 - accuracy: 0.4691 - val_loss: 1.5527 - val_accuracy: 0.4431

Epoch 7/15

1250/1250 [==========] - 13s 10ms/step - loss: 1.4558 - accuracy: 0.4757 - val_loss: 1.5213 - val_accuracy: 0.4618

Epoch 8/15

1250/1250 [==========] - 11s 9ms/step - loss: 1.4344 - accuracy: 0.4845 - val_loss: 1.6001 - val_accuracy: 0.4505

Epoch 9/15

1250/1250 [==========] - 11s 9ms/step - loss: 1.4145 - accuracy: 0.4951 - val_loss: 1.4949 - val_accuracy: 0.4709

Epoch 10/15

1250/1250 [==========] - 11s 9ms/step - loss: 1.3910 - accuracy: 0.5005 - val_loss: 1.5265 - val_accuracy: 0.4577

Epoch 11/15

1250/1250 [==========] - 11s 9ms/step - loss: 1.3757 - accuracy: 0.5073 - val_loss: 1.4938 - val_accuracy: 0.4768

Epoch 12/15

1250/1250 [==========] - 11s 9ms/step - loss: 1.3614 - accuracy: 0.5114 - val_loss: 1.5468 - val_accuracy: 0.4594

Epoch 13/15

1250/1250 [==========] - 11s 9ms/step - loss: 1.3458 - accuracy: 0.5162 - val_loss: 1.5197 - val_accuracy: 0.4724

Epoch 14/15

1250/1250 [==========] - 11s 9ms/step - loss: 1.3251 - accuracy: 0.5238 - val_loss: 1.5624 - val_accuracy: 0.4679

Epoch 15/15

1250/1250 [==========] - 11s 8ms/step - loss: 1.3145 - accuracy: 0.5277 - val_loss: 1.4842 - val_accuracy: 0.4790

```
pred = model.predict(X_test)

pred_classes = [np.argmax(x) for x in pred]

orig = [np.argmax(x) for x in y_test]

print("Predicted values : ",pred_classes[:20])

print("Original values : ",orig[:20])

# captures indexes of misclassified numbers

ERR = []

for i in range(X_test.shape[0]):

    if(pred_classes[i] != orig[i]):

        ERR.append(i)
```

Total number of misclassified

len(ERR)

10-first indexes of misclassified images

ERR[:10]

Output:

Predicted values : [3, 8, 8, 0, 6, 6, 5, 6, 7, 1, 0, 9, 6, 7, 9, 8, 5, 9, 8, 6]

Original values : [3, 8, 8, 0, 6, 6, 1, 6, 3, 1, 0, 9, 5, 7, 9, 8, 5, 7, 8, 6]

[6, 8, 12, 17, 20, 21, 22, 23, 24, 25]

Deep learning tutorial 6: Sequence Classification with LSTM Recurrent Neural Networks

I. Introduction

In this project, we will discover how we can develop the LSTM recurrent neural network models for sequence classification problems [3].

II. Problem Description

The problem that we will use in this project is the IMDB movie review for the sentiment classification .

III. Code using LSTM code:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import string
from keras.models import Model, Input
from keras.layers import *
from keras.datasets import imdb #https://www.imdb.com/ dataset of movies
top_words = 5000
from keras.models import *
from keras.layers import *
# Import some text featrues predefined in sklearn
```

from sklearn.feature_extraction.text import TfidfVectorizer

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words) # get top (top_words)

Output:

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz 17465344/17464789 [==============================] - 0s 0us/step 17473536/17464789 [==============================] - 0s 0us/step

---

word_to_id = imdb.get_word_index()

word_to_id

INDEX_FROM =3

word_to_id["<PAD>"] = 0

word_to_id["<START>"] = 1

word_to_id["<UNK>"] = 2

word_to_id = {k:(v+INDEX_FROM) for k,v in word_to_id.items()} # add the three words to the dictionary with their ids

print(word_to_id)

Output:

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json 1646592/1641221 [==============================] - 0s 0us/step 1654784/1641221 [==============================] - 0s 0us/step {'fawn': 34704, 'tsukino': 52009, 'nunnery': 52010, 'sonja': 16819, 'vani': 63954, 'woods': 1411, 'spiders': 16118

…….

'gales': 41262, 'chip': 8175, "f'": 76386, 'chit': 53434, 'significances': 41263, "bird's": 30748, 'chih': 53435, 'chin': 12013, 'chio': 53436, 'chil': 53437, 'chim': 30750, 'chic': 14379, 'chia': 34957, 'lacks': 1503, "hartmen's": 53438, 'espescially

---

id_to_word = {value:key for key,value in word_to_id.items()}

id_to_word

print(X_train[0])

len(X_train[0])

max_len = np.argmax(np.asarray([len(s) for s in X_train])) ; print(max_len)

Output:

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 2, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 2, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 2, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 2, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 2, 19, 178, 32]

max_len=17934

---

from keras.preprocessing import sequence

max_review_length = 500

X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)

X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

---

embedding_vector_length = 100

model = Sequential()

model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))

model.add(LSTM(100))

model.add(Dense(1, activation='sigmoid')) # if p > 0.5 => 1 , et p < 0.5 => 0

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

model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3, batch_size=512)

Output:

Epoch 1/3 49/49 [==============================] - 268s 5s/step - loss: 0.6229 - accuracy: 0.6752 - val_loss: 0.5385 - val_accuracy: 0.7672 Epoch 2/3 49/49 [==============================] - 261s 5s/step - loss: 0.3596 - accuracy: 0.8484 - val_loss: 0.3214 - val_accuracy: 0.8666 Epoch 3/3 49/49 [==============================] - 307s 6s/step - loss: 0.2428 - accuracy: 0.9034 - val_loss: 0.3213 - val_accuracy: 0.8625

<keras.callbacks.History at 0x7f49fbad4a50>

```
pred = model.predict(X_test)
n_test_samples = pred.shape[0] ; print(n_test_samples)
pred_classes = []
for i in range(n_test_samples):
    if(pred[i] >= 0.5):
      pred_classes.append(1)
    else:
      pred_classes.append(0)
      ind = 24000
pred_classes[ind]
y_test[ind]
```

Output:

25000

0

```
err = []
for i in range(n_test_samples):
    if(pred_classes[i] != y_test[i]):
      err.append(i)
len(err)
```

len(err)=3438

IV- Code LSTM for Sequence Classification with Dropout:

```
import numpy as np # linear algebra
import numpy as np
import tensorflow as tf
from tensorflow import keras
import string
from keras.models import Model, Input
from keras.layers import *
from keras.datasets import imdb #https://www.imdb.com/ dataset of movies
top_words = 5000
from keras.models import *
from keras.layers import *
from keras.layers.embeddings import Embedding
```

```
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words) # get top ( top_words)

----------

word_to_id = imdb.get_word_index()

word_to_id

INDEX_FROM =3

word_to_id["<PAD>"] = 0

word_to_id["<START>"] = 1

word_to_id["<UNK>"] = 2

word_to_id = {k:(v+INDEX_FROM) for k,v in word_to_id.items()} # add the three words to the dictionary with their ids

print(word_to_id)

----------

id_to_word = {value:key for key,value in word_to_id.items()}

id_to_word

print(X_train[0])

len(X_train[0])

max_len = np.argmax(np.asarray([len(s) for s in X_train])) ; print(max_len)
```

Output:

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4,
```

```
22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106,
5, 4, 2223, 2, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25,
124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14,
407, 16, 82, 2, 8, 4, 107, 117, 2, 15, 256, 4, 2, 7, 3766, 5, 723, 36,
71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4,
381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 2, 18, 4, 226, 22,
21, 134, 476, 26, 480, 5, 144, 30, 2, 18, 51, 36, 28, 224, 92, 25,
104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103,
32, 15, 16, 2, 19, 178, 32]
```

```
17934
```

```
from keras.preprocessing import sequence

max_review_length = 500

X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)

X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

embedding_vector_length = 100


model = Sequential()

model.add(Embedding(100,embedding_vector_length, input_length=max_review_length))

#model.add(Dropout(0.2))

model.add(LSTM(100))

model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

print(X_test)
```

[[0 0 0 ... 14 6 717] [0 0 0 ... 125 4 3077] [33 6 58 ... 9 57 975] ... [0 0 0 ... 21 846 2] [0 0 0 ... 2302 7 470] [0 0 0 ... 34 2005 2643]]

```
pred = model.predict(X_test)
n_test_samples = pred.shape[0] ; print(n_test_samples)
pred_classes = []
for i in range(n_test_samples):
   if(pred[i] >= 0.5):
     pred_classes.append(1)
   else:
     pred_classes.append(0)
     ind = 24000
     pred_classes[ind]
     y_test[ind]
```

```
err = []
for i in range(n_test_samples):
    if(pred_classes[i] != y_test[i]):
      err.append(i)
len(err)
len(err)= 3438
```