UNIVERSIDAD PRIVADA FRANZ TAMAYO

FACULTAD DE INGENIERÍA

INGENIERÍA DE SISTEMAS

Redes Neuronales

MINERÍA DE DATOS

Docente: **Ing. Miguel Angel Paco**

Estudiantes: **María Bernarda Serrano Larrea**

Dámaris Raquel Terrazas Terán

Esteban Daniel Pereira Nuñez

Oscar Enrique Ibáñez Siñani

Cochabamba-Bolivia

2019

Introducción

Un CAPTCHA (test de Turing público y automático para distinguir a los ordenadores de los humanos, del inglés "Completely Automated Public Turing test to tell Computers and Humans Apart") es un tipo de medida de seguridad conocido como autenticación pregunta-respuesta. Un CAPTCHA te ayuda a protegerte del spam y del descifrado de contraseñas pidiéndote que completes una simple prueba que demuestre que eres humano y no un ordenador que intenta acceder a una cuenta protegida con contraseña.

La prueba de un CAPTCHA consta de dos partes simples: una secuencia de letras o de números generada aleatoriamente que aparece como una imagen distorsionada y un cuadro de texto. Para superar la prueba y probar que eres un ser humano, simplemente tienes que escribir los caracteres que veas en la imagen del cuadro de texto.

Mediante la siguiente actividad nos proponemos el desarrollo de un captcha en python, esto es posible hoy en día gracias a la librería Pybrain la cual es una librería muy poderosa de redes neuronales, su objetivo es ofrecer algoritmos flexibles y fáciles para el aprendizaje automático

Objetivos

General.

- Incrementar nuestros conocimientos respecto al uso de redes neuronales a través de la práctica.

Específicos.

- Seguir la guia del capítulo 8 de Learning Data Mining with Python de Robert Layson para crear un script que haga uso de redes neuronales.
- Probar el correcto funcionamiento del script creado.
- Analizar en detalle el funcionamiento de cada sección del script.

Desarrollo

```
# Drawing basic CAPTCHAs

import numpy as np
from PIL import Image, ImageDraw, ImageFont
from skimage import transform as tf
```

El primer paso es importar las librerías *numpy* (manejo de Arrays), *PIL* (*Pillow*, generación de imágenes de texto) e *skimage* (*scikit-image*, manejo de imágenes).

```
#def create_captcha(text, shear=0, size=(100, 24)):
def create_captcha(text, shear=0, size=(100, 30)):
    im = Image.new("L", size, "black")
    draw = ImageDraw.Draw(im)

    font = ImageFont.truetype(r"FiraCode-Regular.otf", 22)
    draw.text((2, 2), text, fill=1, font=font)

    image = np.array(im)

    affine_tf = tf.AffineTransform(shear=shear)
    image = tf.warp(image, affine_tf)
    return image / image.max()
```

Se define la función *'create_captcha'* para generar las imágenes CAPTCHA a partir de un string. Primero se define el tipo y tamaño de fuente que se usará. Para luego usar las funciones de *skimage* para distorsionar el texto y dificultar su lectura sistemática.

```
%matplotlib inline
from matplotlib import pyplot as plt

image = create_captcha("GENE", shear=0.5)
plt.imshow(image, cmap='Greys')
```

Se importa la librería *matplotlib* para posibilitar la visualización de las imágenes creadas. Para luego ejecutar *create_captcha* y poner el resultado (*image*) en pantalla.

```
# Splitting the image into individual letters

from skimage.measure import label, regionprops

def segment_image(image):
    labeled_image = label(image > 0)
    subimages = []
    for region in regionprops(labeled_image):
        start_x, start_y, end_x, end_y = region.bbox
        subimages.append(image[start_x:end_x,start_y:end_y])
    if len(subimages) == 0:
        return [image,]
    return subimages
```

Se define la función que *segment_image*, que tiene como objetivo separar la imagen que se le entregue como parámetro en un array de imágenes, cada una conteniendo una de las letras que contenga el texto original.

```
subimages = segment_image(image)

f, axes = plt.subplots(1, len(subimages), figsize=(10, 3))
for i in range(len(subimages)):
    axes[i].imshow(subimages[i], cmap="gray")
```

Se aplica la función anteriormente creada y se guarda el resultado en *subimages*. Cada una de las imágenes en *subimages* es entonces graficada con matplotlib.

```
# Creating a training dataset

from sklearn.utils import check_random_state
random_state = check_random_state(14)
letters = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
shear_values = np.arange(0, 0.5, 0.05)
```

El siguiente paso es la creación de un set de datos para permitir el aprendizaje de la red neuronal. Para esto se define un grupo de caracteres que luego servirán para generar CAPTCHAs aleatorias y sus equivalentes en texto.

```
def generate_sample(random_state=None):
    random_state = check_random_state(random_state)
    letter = random_state.choice(letters)
    shear = random_state.choice(shear_values)
    return create_captcha(letter, shear=shear, size=(30, 30)),letters.index(letter)
```

Se crea una función generate_sample, que usará los caracteres anteriormente establecidos y la funcion create_captcha para generar una imagen captcha.

```
image, target = generate_sample(random_state)
plt.imshow(image, cmap="Greys")
print("The target for this image is: {0}".format(target))

dataset, targets = zip(*(generate_sample(random_state) for i in range(3000)))
dataset = np.array(dataset, dtype='float')
targets = np.array(targets)

from sklearn.preprocessing import OneHotEncoder
onehot = OneHotEncoder()
y = onehot.fit_transform(targets.reshape(targets.shape[0],1))

y = y.todense()
```

Se usa *generate_sample* una vez para generar y visualizar una muestra. Y luego se usa la misma función para generar 3000 pares de CAPTCHAs y significados de estas para entrenar. Finalmente los datos de salida se ordenan con un formato apropiado para la librería de redes neuronales.

```
# Adjusting our training dataset to our methodology

from skimage.transform import resize
dataset = np.array([resize(segment_image(sample)[0], (20, 20)) for sample in dataset])
X = dataset.reshape((dataset.shape[0], dataset.shape[1] * dataset.shape[2]))

#from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.9)
```

De forma parecida a la anterior, se les da un formato apropiado a los datos de entrada. Se separa una parte de los datos no para entrenamiento, sino para pruebas.

```
# Training and classifying

from pybrain.datasets import SupervisedDataSet

training = SupervisedDataSet(X.shape[1], y.shape[1])
for i in range(X_train.shape[0]):
    training.addSample(X_train[i], y_train[i])

testing = SupervisedDataSet(X.shape[1], y.shape[1])
for i in range(X_test.shape[0]):
    testing.addSample(X_test[i], y_test[i])

from pybrain.tools.shortcuts import buildNetwork
net = buildNetwork(X.shape[1], 100, y.shape[1], bias=True)
```

Se importa los modelos de set de datos de Pybrain y con ellos se construyen los sets de datos de entrada y de salida para entrenamiento y testing.

Se inicializa la red neuronal *net* con las formas de los sets de datos.

```
# Back propagation

from pybrain.supervised.trainers import BackpropTrainer
trainer = BackpropTrainer(net, training, learningrate=0.01,
weightdecay=0.01)

trainer.trainEpochs(epochs=20)

predictions = trainer.testOnClassData(dataset=testing)

from sklearn.metrics import f1_score
print("F-score: {0:.2f}".format(f1_score(predictions, y_test.argmax(axis=1),average='macro' )))
```

Se entrena el modelo *net* usando el set de datos *training*. Se le ordena pasar por 20 iteraciones.

```
# Predicting words

def predict_captcha(captcha_image, neural_network):
    subimages = segment_image(captcha_image)
    predicted_word = ""
    for subimage in subimages:
        subimage = resize(subimage, (20, 20))
        outputs = net.activate(subimage.flatten())
        prediction = np.argmax(outputs)
        predicted_word += letters[prediction]
    return predicted_word
```

Se define una funcion que recibirá una imagen captcha y una red neuronal, para intentar predecir la palabra en la imagen. Pasa por el proceso de separar la palabra en letras individuales y predecir cada letra individualmente, antes de unir las letras y retornarlas en *predicted_word*.

```
word = "GENE"
captcha = create_captcha(word, shear=0.2)
print(predict_captcha(captcha, net))
```

Se pone a prueba el nuevo método con la palabra “GENE”. Creando una imagen CAPTCHA y haciendo su predicción.

```
def test_prediction(word, net, shear=0.2):
    captcha = create_captcha(word, shear=shear)
    prediction = predict_captcha(captcha, net)
    prediction = prediction[:4]
    return word == prediction, word, prediction
```

Se define una función para probar la exactitud de la predicción de una palabra.

```
from nltk.corpus import words
valid_words = [word.upper() for word in words.words() if len(word) == 4]

num_correct = 0
num_incorrect = 0
for word in valid_words:
    correct, word, prediction = test_prediction(word, net, shear=0.2)
    if correct:
        num_correct += 1
    else:
        num_incorrect += 1
print("Number correct is {0}".format(num_correct))
print("Number incorrect is {0}".format(num_incorrect))
```

Con la ayuda de NLTK (*Natural Language Toolkit*) se arma una lista de palabras válidas (*valid_words*). Luego procede a usar *test_prediction* con cada una de estas, contando si la predicción fue correcta o incorrecta.

```
# Ranking mechanisms for words

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(np.argmax(y_test, axis=1), predictions)

plt.figure(figsize=(10, 10))
plt.imshow(cm)

tick_marks = np.arange(len(letters))
plt.xticks(tick_marks, letters)
plt.yticks(tick_marks, letters)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

Se grafican los resultados de las predicciones, comparando las letras reales con las predecidas.

```
from nltk.metrics import edit_distance
steps = edit_distance("STEP", "STOP")
print("The number of steps needed is: {0}".format(steps))
```

La función *edit_distance* muestra el numero de caracteres de diferencia entre 2 strings.

```
def compute_distance(prediction, word):
    return len(prediction) - sum(prediction[i] == word[i] for i in range(len(prediction)))
```

Se define la función *compute_distance*, que cuenta la cantidad de caracteres que son diferentes entre 2 palabras del mismo tamaño.

```
# Putting it all together

from operator import itemgetter
def improved_prediction(word, net, dictionary, shear=0.2):
    captcha = create_captcha(word, shear=shear)
    prediction = predict_captcha(captcha, net)
    prediction = prediction[:4]

    if prediction not in dictionary:
        distances = sorted([(word, compute_distance(prediction, word)) for word in dictionary], key=itemgetter(1))
        best_word = distances[0]
        prediction = best_word[0]

    return word == prediction, word, prediction
```

Se define la función *improved_prediction*, que además de crear y predecir el significado de un CAPTCHA, busca en el diccionario la palabra más cercana y la entrega como respuesta.

```
num_correct = 0
num_incorrect = 0
for word in valid_words:
    correct, word, prediction = improved_prediction(word, net, valid_words, shear=0.2)
    if correct:
        num_correct += 1
    else:
        num_incorrect += 1
print("Number correct is {0}".format(num_correct))
print("Number incorrect is {0}".format(num_incorrect))
```

Para terminar, se llevan a cabo las predicciones una vez más, pero aplicando *improved_prediction* para mejorar la precisión.

Conclusiones

Una vez terminada la actividad podemos determinar una vez más que python es el lenguaje predilecto para las inteligencias artificiales ya que nos brinda una infinidad de librerías las cuales podemos usar libremente, incluso viendo el código base al ser open source, como este fue el caso, que se tuvo en esta actividad donde hubo un problema de importación de la librería pybrain a la cual no podiamos acceder a uno de sus apartados, de tal forma que tuvimos que importar la librería desde github para poder correr a 100%.