ANN-Intro-Parte-1

November 8, 2019

1

Por: V. Robles B.

2 Redes Neuronales con Scikit-Learn: una introducción - Parte 1

En este cuaderno se presenta una breve introducción de los principales aspectos para crear, entrenar y validar redes neuronales artificiales en Python con la librería scikit-learn. A lo largo del cuaderno se hará especial énfasis en el **Perceptrón multicapa** como herramienta para realizar tareas de clasificación.

Como primer punto, es importante verificar que tengamos todos las librerías instaladas.

2.1 Prerrequisitos:

A fin de poder ejecutar las instrucciones de este cuaderno, debemos verificar que tengamos instaladas las siguientes librerías:

- Python (versiones >=2.7 o >=3.3)
- Numpy >= 1.8.2
- SciPy >= 0.13.3

2.2 Instalación:

La instalación de **scikit-learn** se puede realizar de manera sencilla a través del siguiente comando:

```
pip install -U scikit-learn
```

Donde la opción **-U** indica que si existe el paquete, deberá actualizarse a la última versión estable existente.

De igual forma, si se desean mayores detalles, es factible consultar el siguiente enlace.

2.3 Ejemplo básico 1: compuerta XOR

Como se conoce, el perceptrón simple (de una sola neurona y una sola capa) no es capaz de resolver problemas que no sean separables linealmente.

Por ello, en esta sección aprenderemos cómo resolver un el sencillo problema de la compuerta XOR (que no es separable linealmente). Comencemos!

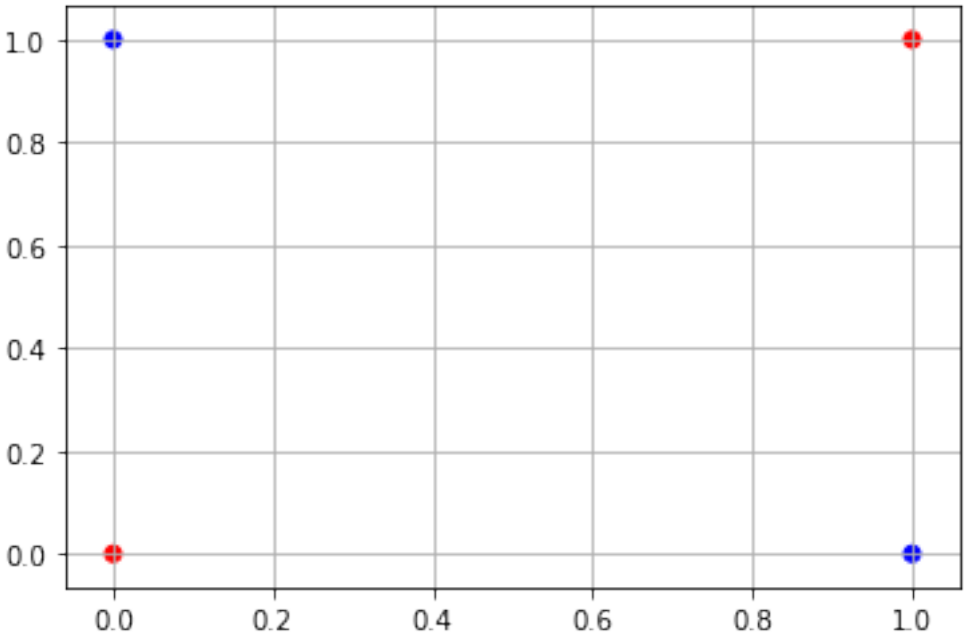
2.3.1 Corpus

Para entrenar la red, debemos tener claro en primer lugar, cuáles son las entradas y salidas que nuestra red neuronal deberá aprender. En la siguiente tabla se puede apreciar los patrones de entrada (donde cada patrón está conformado por dos entradas x_1 y x_2), y las correspondientes etiquetas o salidas (δ).

x_1	x_2	δ
0	0	0
0	1	1
1	0	1
1	1	0

A continuación visualizamos los patrones que se desea que aprenda nuestra red:

[1]:
```
import matplotlib.pyplot as pp
import numpy as np

%matplotlib inline

x=np.array([[0,0],[0,1],[1,0],[1,1]])
d=np.array([0,1,1,0])

pp.scatter(x[:,0],x[:,1],color=['blue' if i==1 else 'red' for i in d])

pp.grid(True)
pp.show()
```



Como se puede apreciar, no es posible separar con una sola línea los puntos azules (que indican que la red debería devolver un valor de 1 en esos casos) de los puntos rojos.

Por ello, ahora vamos a crear una red multicapa que tendrá la siguiente estructura:

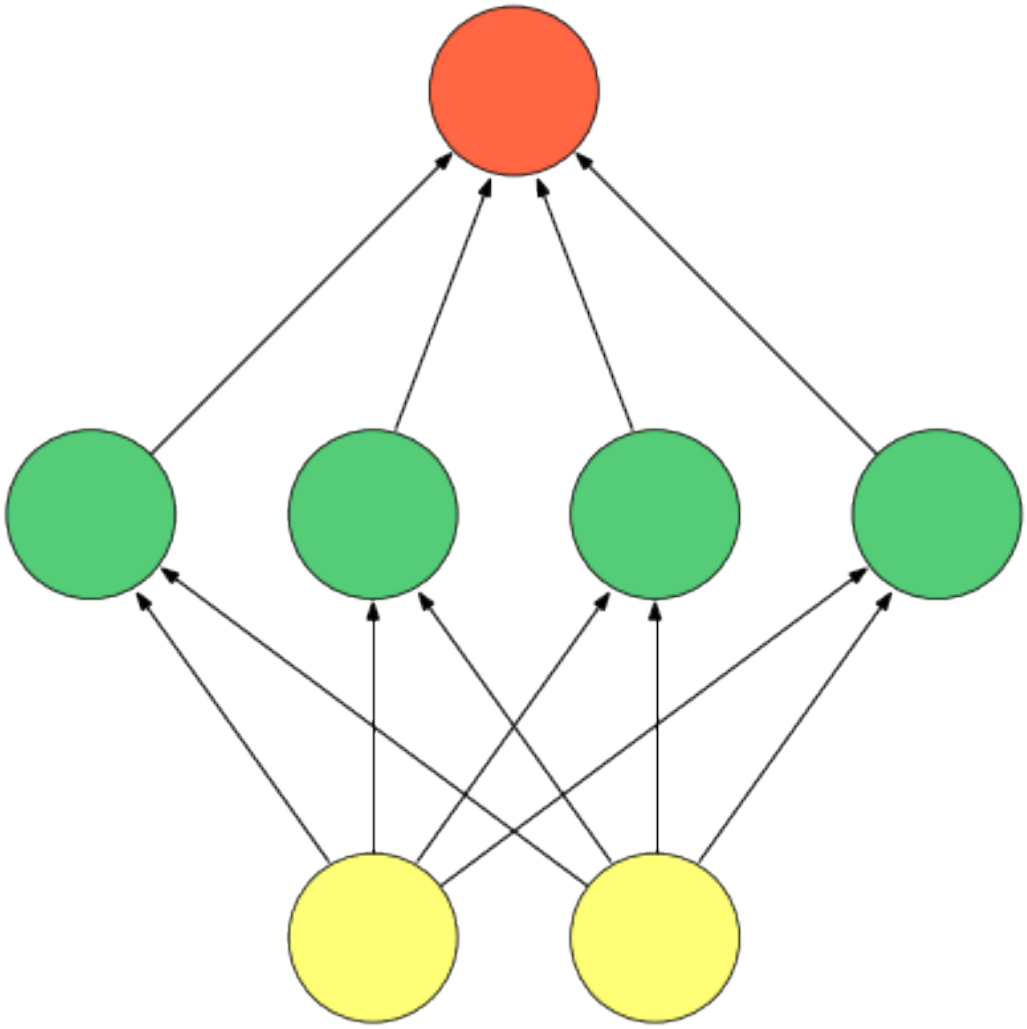
- Número de entradas = 2
- Total de capas = 2
- Neuronas en la capa oculta = 4
- Salidas = 1

Si deseamos visualizar la **estructura** de nuestra red neuronal, podemos emplear el paquete viznet. Es importante observar que en **THE ASIMOV INSTITUTE** existe una completa descripción gráfica de la estructura de los diferentes tipos de redes neuronales: The Neural Network Zoo [3].

[2]:
```
from viznet import connecta2a, node_sequence, NodeBrush, EdgeBrush, DynamicShow

# Creamos variables con los parametros que tendra la red
entradas = 2
neuronas_capa_oculta = 4
neuronas_capa_salida = 1


def dibujar_red_neuronal(ax, num_node_list):

    num_hidden_layer = len(num_node_list) - 2
    token_list = ['\sigma^z'] + \
        ['y^{(%s)}' % (i + 1) for i in range(num_hidden_layer)] + ['\psi']
```

```
    kind_list = ['nn.input'] + ['nn.hidden'] * num_hidden_layer + ['nn.output']
    radius_list = [0.3] + [0.2] * num_hidden_layer + [0.3]
    y_list = 1.5 * np.arange(len(num_node_list))

    seq_list = []
    for n, kind, radius, y in zip(num_node_list, kind_list, radius_list,␣
 ↪y_list):
        b = NodeBrush(kind, ax)
        seq_list.append(node_sequence(b, n, center=(0, y)))

    eb = EdgeBrush('-->', ax)
    for st, et in zip(seq_list[:-1], seq_list[1:]):
        connecta2a(st, et, eb)


def real_bp():
    with DynamicShow((6, 6), '_feed_forward.png') as d:
        dibujar_red_neuronal(d.ax, num_node_list=[entradas,␣
 ↪neuronas_capa_oculta, neuronas_capa_salida])


real_bp()
```

```
Press `c` to save figure to "_feed_forward.png", `Ctrl+d` to break >>
> /root/anaconda2/envs/ia2/lib/python3.7/site-
packages/viznet/context.py(61)__exit__()
-> plt.savefig(self.filename, dpi=300, transparent=True)
(Pdb) c
```



A continuación emplearemos **sckikit learn** para crear, entrenar y probar la red neuronal MLP-Classifier que se especificó con anterioridad. Los parámetros que se usarán son los siguientes:

- Algoritmo para la reducción del error en el entrenamiento: **lbfgs** optimizador basado en métodos cuasi-Newtonianos. Mayor información en este link.
- Función de activación de las neuronas: **logística** (*logistic*)
- Máximo número de iteraciones (*max_iter*): 10000

[3]:
```
# Importamos el Perceptron Multicapa para Clasificacion
from sklearn.neural_network import MLPClassifier

# Creamos la red neuronal
mlp=MLPClassifier(solver = 'lbfgs', activation='logistic', verbose=True,␣
 ↪alpha=1e-4, tol=1e-15, max_iter=10000, \
```

```
                    hidden_layer_sizes=(neuronas_capa_oculta,␣
 ↪neuronas_capa_salida))

print(mlp)
# Realizamos el proceso de entrenamiento
mlp.fit(x,d)

# Mostramos los pesos entre la entrada y la capa oculta
print('Pesos W^(0): \n:',mlp.coefs_[0])

# Mostramos los pesos entre la capa oculta y la capa de salida
print('\nPesos W^(1): \n:',mlp.coefs_[1])

# Probamos si la red devuelve valores apropiados de acuerdo a las entradas␣
 ↪(test):
for entrada in x:
    print('\nPrueba con {','|'.join([str(i) for i in entrada]),'} => ',mlp.
 ↪predict(entrada.reshape(1,-1)))
```

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(4, 1), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='lbfgs', tol=1e-15,
              validation_fraction=0.1, verbose=True, warm_start=False)
Pesos W^(0):
: [[-1.40533430e-02 -2.23279792e-02 -2.25502888e-03 -3.84970283e-03]
 [-2.33452747e-02 -9.36290419e-03  8.47461104e-06  4.83697348e-03]]

Pesos W^(1):
: [[-0.00288992]
 [-0.00257242]
 [-0.01855455]
 [ 0.02772381]]

Prueba con { 0|0 } =>  [1]

Prueba con { 0|1 } =>  [1]

Prueba con { 1|0 } =>  [1]

Prueba con { 1|1 } =>  [1]
```

2.3.2 Práctica ANN-0:

2.3.3 Desarrollado por: Jorge Sanisaca

Modifique el código anterior, a fin de usar Hot Encoding y contar con 2 salidas en lugar de 1.

[4]:
```
# Importamos el Perceptron Multicapa para Clasificacion
from sklearn.neural_network import MLPClassifier

# TODO:
# Modificar las salidas deseadas para representarlas con 2 valores binarios


# Modoficar los parametros de la Red Neuronal. Sustituir None por el valor␣
 ↪correspondiente
mlp=MLPClassifier(solver = 'lbfgs', activation='logistic', alpha=1e-4,␣
 ↪tol=1e-15, max_iter=10000, \
                  hidden_layer_sizes=(neuronas_capa_oculta, 2))

print(mlp)
# Realizamos el proceso de entrenamiento con la nueva representacion de la␣
 ↪salida.
# Sustituir None por el valor correspondiente:

# TODO:

mlp.fit(x,d)

# Mostramos los pesos entre la entrada y la capa oculta
print('Pesos W^(0): \n:',mlp.coefs_[0])

# Mostramos los pesos entre la capa oculta y la capa de salida
print('\nPesos W^(1): \n:',mlp.coefs_[1])

# Probamos si la red devuelve valores apropiados de acuerdo a las entradas␣
 ↪(test):
for entrada in x:
    print('\nPrueba con {','|'.join([str(i) for i in entrada]),'} => ',mlp.
 ↪predict(entrada.reshape(1,-1)))
```

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(4, 2), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='lbfgs', tol=1e-15,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

```
Pesos W^(0):
: [[-2.86086672  2.82005671 -2.8371253  -2.84449472]
 [ 2.85877029 -2.82327593  2.83573742  2.84256589]]

Pesos W^(1):
: [[-1.9528605   5.12583438]
 [ 2.0556224  -5.05034787]
 [-5.07550605  2.00422812]
 [-5.08668856  1.98143225]]

Prueba con { 0|0 } =>  [0]

Prueba con { 0|1 } =>  [1]

Prueba con { 1|0 } =>  [1]

Prueba con { 1|1 } =>  [0]
```

2.3.4 Práctica ANN-1:

Genere 1000 puntos aleatorios con coordenadas (x_1, x_2). Con estos puntos, deberá realizar las siguientes tareas:

- Seleccionar de forma aleatoria 80% de los puntos para entrenar la red y el restante 20% se empleará para probar la red.
- Entrenar la red hasta lograr un error mínimo.
- Probar la red y presentar la matriz de confusión.
- Indicar el nivel de precisión (muestras correctamente clasificadas frente al total de muestras):

$precision = \frac{muestras \;\; correctamente \;\; clasificadas}{total \;\; de \;\; muestras}$

[5]:
```
#IMportamos el modulo clasificasion_report para mostrar las principales
 ↪metricas de clasificacion
#Importamos el modulo confusion_matrix para obtener la matriz de confusion
from sklearn.metrics import classification_report, confusion_matrix
#Importamos el modulo train_test_split para realizar la separación respectiva
from sklearn.model_selection import train_test_split
#Importamos el perceptron multicapa para la clasificasion
from sklearn.neural_network import MLPClassifier


#Importamos el modolo random para generar los 1000 numeros aleatorios
import random

#Total de numeros aleatotios a generar
n = 1000
#Coordenadas(x1, x2)
coordenadas = []
```

```
delta = [1 if i%2 == 0 else 0 for i in range(0, n)]
#Generamos las coordenadas randomicas
for i in range(0, n):
    coordenadas.append([random.random() * 100, random.random() * 100])

datos = np.array(coordenadas)
#realizamos la separacion respectiva de datos de entrenamiento y de prueba
x_train, x_test, d_train, d_test = train_test_split(datos, delta, test_size=0.
 ↪80, random_state=42)

#Creamos la red reuronal
mlp = MLPClassifier(solver = 'lbfgs', activation='logistic', verbose=True,␣
 ↪alpha=1e-4, tol=1e-15, max_iter=10000, \
                 hidden_layer_sizes=(150, 2))

#Realizamos en proceso de entrenamiento
mlp.fit(datos, delta)

#Realizamos la prediccion con los datos de prueba
prediccion = mlp.predict(x_test)
#Obtenemos la matriz de confusion
print('Matriz de Confusion\n')
matriz = confusion_matrix(d_test, prediccion)
print(matriz)
print('\n')
#Imprimimos el reporte de clasificacion
print(classification_report(d_test, prediccion))
```

```
Matriz de Confusion

[[149 259]
 [ 34 358]]


              precision    recall  f1-score   support

           0       0.81      0.37      0.50       408
           1       0.58      0.91      0.71       392

    accuracy                           0.63       800
   macro avg       0.70      0.64      0.61       800
weighted avg       0.70      0.63      0.60       800
```

2.4 Referencias

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

[2] Portilla, J. (2017). A Beginner's Guide to Neural Networks in Python and SciKit Learn 0.18. Retrieved from https://www.springboard.com/blog/beginners-guide-neural-network-in-python-scikit-learn-0-18/.

[3] The Asimov Institute. (2018). THE NEURAL NETWORK ZOO. Retrived from: http://www.asimovinstitute.org/neural-network-zoo/