COLOMBIA POTENCIA DE LA VIDA

TIC

Ejemplo de red neuronal aplicada en clasificación

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

UTP Universidad Tecnológica de Pereira

Ejemplo de red neuronal aplicada en clasificación

Inicialmente se tiene un código para crear datos artificiales para clasificación en forma de dos anillos concéntricos y luego visualiza estos datos junto con un mapa de predicción.

1. Importar bibliotecas:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
```

numpy (np): Se utiliza para manipulación numérica y cálculos matriciales.

matplotlib.pyplot (plt): Se utiliza para la visualización de datos.

make_circles de sklearn.datasets: Se utiliza para generar datos artificiales en forma de dos anillos concéntricos.

2. Generación de datos:

```
# Creamos nuestros datos artificiales, donde buscaremos
clasificar dos anillos concéntricos de datos.

X, Y = make_circles(n_samples=500, factor=0.5, noise=0.05)
```

Se utilizan los datos generados artificialmente de dos anillos concéntricos mediante la función **make_circles**. Los parámetros **n_samples** controlan el número de puntos de datos, **factor** controla la distancia entre los dos anillos y **noise** controla la cantidad de ruido agregado a los datos.

3. Definición del mapa de predicción:

```
# Resolución del mapa de predicción.
res = 100
# Coordendadas del mapa de predicción.
_x0 = np.linspace(-1.5, 1.5, res)
_x1 = np.linspace(-1.5, 1.5, res)
```

Se define la resolución del mapa de predicción (**res**) y se generan las coordenadas **_x0** y **_x1** para el mapa de predicción. Luego, se crea una cuadrícula de coordenadas **_pX** que cubre todo el espacio del mapa de predicción. Además, se inicializa una matriz **_pY** con valores de 0.5 en todas las posiciones del mapa de predicción.

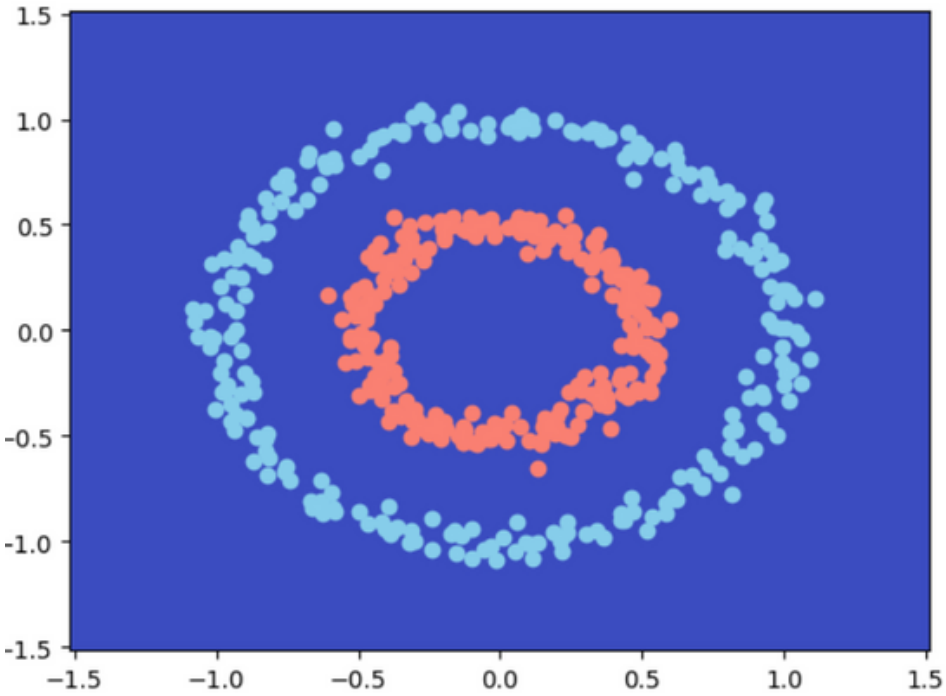
4. Visualización del mapa de predicción y los datos:

```
# Visualización del mapa de predicción.
plt.figure()
plt.pcolormesh(_x0, _x1, _pY, cmap="coolwarm")
# Visualización de la nube de datos.
plt.scatter(X[Y == 0,0], X[Y == 0,1], c="skyblue")
plt.scatter(X[Y == 1,0], X[Y == 1,1], c="salmon")
```

Se traza el mapa de predicción utilizando **plt.pcolormesh** para visualizar la región donde se realizarán las predicciones.

Se trazan los puntos de datos generados en forma de dos anillos concéntricos usando **plt.scatter**. Los puntos pertenecientes a la primera clase (anillo interno) se muestran en azul claro (**"skyblue"**) y los puntos de la segunda clase (anillo externo) se muestran en rojo claro (**"salmon"**).

Hasta este punto se genera el siguiente gráfico:



Después, continúa con un código que utiliza Keras para construir, compilar y entrenar una red neuronal secuencial para clasificar los datos generados artificialmente anteriormente.

1 Importar Keras:

```
import keras
```

2 Definición de hiperparámetros:

```
# learning rate

lr = 0.01
```

lr: learning rate (tasa de aprendizaje) establecido en 0.01. Este valor determina el tamaño de los pasos que el optimizador da en la dirección de minimizar la función de pérdida durante el entrenamiento.

3 Creación del modelo:

```
# Creamos el objeto que contendrá a nuestra red neuronal,
como secuencia de capas.

model = keras.Sequential()
```

Se crea un modelo de red neuronal secuencial utilizando **keras.Sequential()**. Este modelo se compone de capas que se apilan secuencialmente una encima de la otra.

4 Añadir capas:

```
# Añadimos la capa 1

l1 = model.add(keras.layers.Dense(32, activation='relu'))

# Añadimos la capa 2

l2 = model.add(keras.layers.Dense(8, activation='relu'))

# Añadimos la capa 3

l3 = model.add(keras.layers.Dense(1, activation='sigmoid'))
```

Se añaden tres capas densamente conectadas a la red neuronal utilizando el método **add()** del modelo.

- La primera capa (**l1**) tiene 32 neuronas con una función de activación ReLU (Rectified Linear Unit), que es comúnmente utilizada en redes neuronales para introducir no linealidades en el modelo.
- La segunda capa (**l2**) tiene 8 neuronas con una función de activación ReLU.
- La tercera capa (**l3**) tiene 1 neurona con una función de activación sigmoide, que es adecuada para problemas de clasificación binaria ya que produce una salida entre 0 y 1 que puede interpretarse como la probabilidad de pertenecer a una clase.

5 Compilación del modelo:

```
# Compilamos el modelo, definiendo la función de coste y el
optimizador.

model.compile(loss='mse',
optimizer=keras.optimizers.SGD(lr=0.05), metrics=['acc'])
```

Se compila el modelo utilizando el método **compile()**. Aquí, se especifica la función de pérdida (**loss**), que en este caso es la pérdida cuadrática media (**mse**, mean squared error), y el optimizador (**optimizer**), que es un SGD (Stochastic Gradient Descent) con una tasa de aprendizaje de 0.05. También se especifica una métrica adicional para evaluar el rendimiento del modelo durante el entrenamiento, que es la precisión (**acc**, accuracy).

6 Entrenamiento del modelo:

```
# Y entrenamos al modelo. Los callbacks

model.fit(X, Y, epochs=100)
```

El modelo se entrena utilizando el método **fit()**, pasando los datos de entrada (**X**) y las etiquetas de salida (**Y**). Se especifica el número de épocas (**epochs**) para el entrenamiento, que en este caso es 100. El proceso de entrenamiento muestra lo siguiente en pantalla:

```
Epoch 95/100
16/16 [==============================] - 0s 2ms/step - loss: 0.2141 - acc: 0.9640
Epoch 96/100
16/16 [==============================] - 0s 2ms/step - loss: 0.2136 - acc: 0.9660
Epoch 97/100
16/16 [==============================] - 0s 2ms/step - loss: 0.2130 - acc: 0.9680
Epoch 98/100
16/16 [==============================] - 0s 2ms/step - loss: 0.2123 - acc: 0.9740
Epoch 99/100
16/16 [==============================] - 0s 2ms/step - loss: 0.2118 - acc: 0.9820
Epoch 100/100
16/16 [==============================] - 0s 2ms/step - loss: 0.2111 - acc: 0.9840
```

Finalmente se ejecuta un código que completa la visualización del mapa de predicción junto con los datos generados y clasificados por el modelo de red neuronal previamente definido.

1. Resolución del mapa de predicción:

```
# Resolución del mapa de predicción.
res = 100
# Coordendadas del mapa de predicción.
_x0 = np.linspace(-1.5, 1.5, res)
_x1 = np.linspace(-1.5, 1.5, res)
```

Se define la resolución del mapa de predicción (**res**) como 100, lo que significa que el mapa se dividirá en una cuadrícula de 100x100 puntos. Se generan arreglos **_x0** y **_x1** que contienen valores espaciados uniformemente en el rango de -1.5 a 1.5 utilizando **np.linspace()**. Estos arreglos representan las coordenadas del mapa de predicción a lo largo de los ejes x e y.

2. Generación de las coordenadas de entrada del mapa de predicción:

```
# Input con cada combo de coordenadas del mapa de predicción.

_pX = np.array(np.meshgrid(_x0, _x1)).T.reshape(-1, 2)
```

Se crea una cuadrícula de coordenadas **_pX** utilizando **np.meshgrid()** para combinar todas las posibles combinaciones de coordenadas **_x0** y **_x1**. Luego, se reformatea esta matriz en una matriz bidimensional de tamaño **(res*res, 2)** para que pueda ser alimentada al modelo de red neuronal.

3. Realización de predicciones:

```
# Realiza predicciones.

_pY = model.predict(_pX)
```

Se utilizan las coordenadas del mapa de predicción **_pX** como entrada para el modelo de red neuronal (**model.predict()**), lo que resulta en las predicciones **_pY**.

4. Visualización del mapa de predicción:

```
# Visualización del mapa de predicción.
plt.figure()
plt.pcolormesh(_x0, _x1, _pY.reshape(res, res),
cmap="coolwarm")
```

- Se crea una figura utilizando **plt.figure()** para la visualización.
- Se utiliza **plt.pcolormesh()** para trazar el mapa de predicción utilizando los arreglos **_x0** y **_x1**, y los valores de predicción **_pY**. Se aplica un mapa de colores "coolwarm" para resaltar las diferentes regiones del mapa de predicción.

5. Visualización de los datos generados:

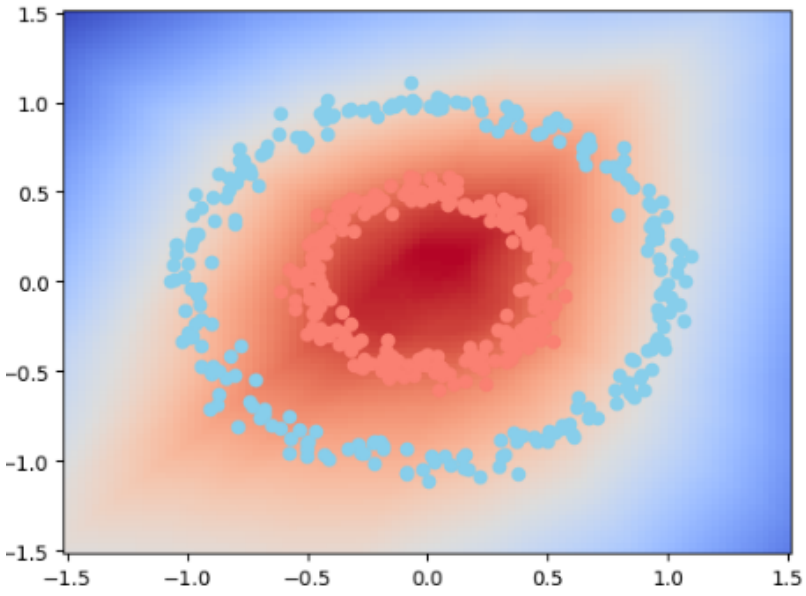
```
# Visualización de la nube de datos.

plt.scatter(X[Y == 0,0], X[Y == 0,1], c="skyblue")

plt.scatter(X[Y == 1,0], X[Y == 1,1], c="salmon")
```

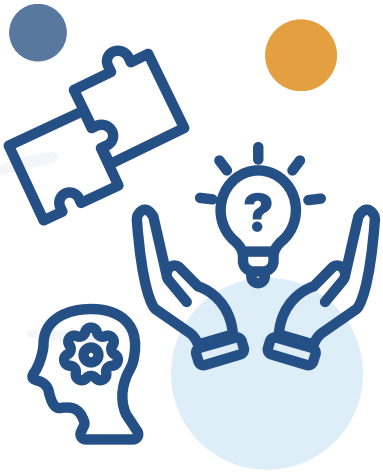
Se trazan los datos generados artificialmente utilizando **plt.scatter()**. Los puntos de la clase 0 se muestran en azul claro (**"skyblue"**) y los puntos de la clase 1 se muestran en rojo claro (**"salmon"**).

Ejecutando este último bloque de código se observa la siguiente gráfica:



Preguntas de comprensión

1. ¿Qué tipo de problema se aborda en este código?
2. ¿Qué función de activación se utiliza en las capas ocultas de la red neuronal?
3. ¿Cuál es el propósito de la función de pérdida "mse" en la compilación del modelo?
4. ¿Qué método de optimización se utiliza para entrenar el modelo?
5. ¿Cuántas épocas se especifican para el entrenamiento del modelo?



Ejercicios de exploración

1. Modifica el número de neuronas en las capas ocultas de la red neuronal y observa cómo afecta al rendimiento del modelo.
2. Experimenta con diferentes funciones de activación en las capas ocultas, como 'tanh' o 'sigmoid', y compara los resultados con la función de activación 'relu'.
3. Prueba diferentes tasas de aprendizaje (lr) en el optimizador SGD y observa cómo afecta al proceso de entrenamiento.
4. Cambia la función de pérdida utilizada en la compilación del modelo a 'binary_crossentropy' y observa cómo afecta al rendimiento del modelo.
5. Explora cómo varía el rendimiento del modelo al ajustar el número de épocas durante el entrenamiento. ¿Se produce sobreajuste o subajuste con un número de épocas demasiado bajo o demasiado alto?

Estas preguntas y ejercicios ayudarán a profundizar tu comprensión del código y a explorar diferentes aspectos del entrenamiento de redes neuronales.