Curso de IA

Capítulo 8. Cuestionario

Para estudiantes

1. ¿Cuál es el problema que se plantea cuando el número de nodos es pequeño al analizar redes neuronales artificiales?

   1 Si el número de nodos es pequeño, el número de operaciones a procesar es pequeño, por lo que se realiza rápidamente.

   2 No se pueden crear límites de decisión complejos al realizar un modelo de análisis.

   3 No se puede realizar el algoritmo de retropropagación que ajusta pesos y umbrales.

   4 El número de nodos no tiene nada que ver con el modelo de análisis.

2. ¿Cuál es el concepto que se describe a continuación?

   Matriz multidimensional con 3 componentes: Rango, Forma, Tipo

Se esta haciendo referencia al concepto de **Tensor.**

3. ¿Qué ocurre si el error de verificación aumenta constantemente cuando graficamos el error de verificación para cada época utilizando el método de descenso de pendiente por lotes? Además, ¿cómo se puede resolver este problema?

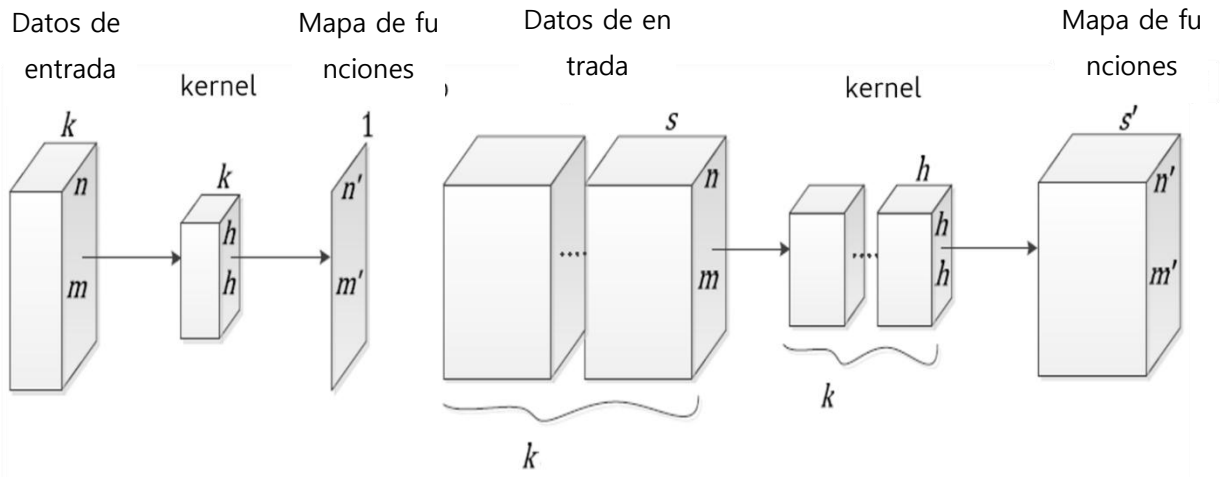
   Cuando hay un aumento constante del error de verificación, esto indica overfitting en el modelo, lo que significa que el modelo aprende demasiado bien los datos de entrenamiento, pero este aprendizaje no le sirve para datos futuros o nunca antes vistos lo que lo vuelve poco confiable.

   Este problema puede resolverse aplicando técnicas como la regularización, el uso de dropout, detener el entrenamiento a tiempo para evitar un sobreajuste aún mayor, ampliar el conjunto de datos o, en su caso, reducir la complejidad de la red para mejorar su capacidad de generalización ante datos futuros.

4. La figura de la página siguiente son datos con una estructura tridimensional.

   (1) Presente las ecuaciones de convolución para (a).

   (2) Presente las ecuaciones de convolución para (b).



Datos multicanal (eje. imagen a color RGB) (b) Datos 3D (eje. video, imá genes del cerebro MRI)

5. Aplicar la función softmax cuando la salida de la red neuronal es $(0.4,2.0,0.001,0.32)^T$ y escribir el resultado.

Aplicación de la función softmax

Se solicita aplicar la función **softmax** a la salida de la red neuronal representada por el vector:

[(0.4, 2.0, 0.001, 0.32)^T]

```
import numpy as np
```

```
#? Vector de salida de la red neuronal
vectorSalida = np.array([0.4, 2.0, 0.001, 0.32])
```

```
#? Aplicacion de la funcion softmax
exponencial = np.exp(vectorSalida)
valor_softmax = exponencial / np.sum(exponencial)
```

```
#? Mostrar resultado
print(f"Vector original de salida : {vectorSalida}")
print(f"Vector aplicando softmax : {valor_softmax}")
```

```
Vector original de salida : [4.0e-01 2.0e+00 1.0e-03 3.2e-01]
Vector aplicando softmax : [0.13250053 0.65627943 0.08890663 0.12231341]
```

6. Forme un modelo de predicción de series temporales basado en RNN con referencia al siguiente código.

```
importar pandas como pd
importar numpy como np
desde sklearn.model_selection importar train_test_split
desde sklearn.preprocessing importar MinMaxScaler
importar tensorflow como tf
importar matplotlib.pyplot como plt

# read data# Leer datos
df = pd.read_csv('data_boston.csv', header='infer', encoding='latin1')
df = df[['PRICE']]

# scale input & X, y
scaler = MinMaxScaler()
ts_scaled = scaler.fit_transform(df)

# scale
ts_scaled_2 = ts_scaled.reshape(1, -1, 1)

# training parameters
batch_size = 1
n_epochs  = 1000
learn_rate = 0.0001

# model
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(None, 1)))

# completar el modelo basado en la rnn y la capacitación
```

EL CODIGO COMPLETO LO MANDE EN PDF EN FORMATO JUPYTER NOTEBOOK

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
import matplotlib.pyplot as plt

#? Lectura de datos
df = pd.read_csv(r'./data_boston.csv',
                 header='infer',
                 encoding='latin1')
df = df[['PRICE']]
```

```
#? Escalar los datos
scaler = MinMaxScaler()
scaleddata = scaler.fit_transform(df)

#? Crear datos de entrada para RNN
def crear_secuencias(data, lensec=5):
    """Creacion de datos de entrada
    data = datos a introducir
    lensec = secuencia de datos por defecto 5"""
    X, y = [], []
    for i in range (len(data) - lensec):
        X.append(data[i:i+lensec])
        y.append(data[i+lensec])
    return np.array(X), np.array(y)
lensecuencias = 5
X, y = crear_secuencias(scaleddata)

#? Dividir datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2
,
                                                    shuffle=False
)

#? Parámetros de entrenamiento
batch_size = 1
n_epochs = 1_000
learn_rate = 0.0001

#? Definicion del modelo RNN
model = tf.keras.Sequential()
model.add(tf.keras.layers.SimpleRNN(50, activation='tanh',
return_sequences=False, input_shape=(lensecuencias, 1)))
model.add(tf.keras.layers.Dense(1))

#? Compilar modelo
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=le
arn_rate),
              loss='mse')

#? Entrenar modelo
history = model.fit(X_train, y_train,
                    epochs=n_epochs,
                    batch_size=batch_size,
                    validation_data=(X_test, y_test),
                    verbose=1)

#? Graficar pérdida
plt.plot(history.history['loss'], label='Pérdida entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida validación')
plt.legend()
plt.show()

# Predicción final para verificar funcionamiento
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)
```