Recurrent_Neural_Network_(RNN)

May 3, 2022

1 Tarea 2

Ver el siguiente video y programar la red neuronal recurrente.

https://www.youtube.com/watch?v=BSpXCRTOLJA

El código se entrega en un archivo word o pdf.

La fecha de entrega es el miércoles 4 de mayo

[2]:
```
# importación de librerias y dependencias
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from tensorflow.compat.v1.keras.layers import CuDNNLSTM
```

[3]:
```
# Carga de datos
mnist = tf.keras.datasets.mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

[4]:
```
print(X_train.shape)
print(X_train[0].shape)
# Se tienen 60000 imagenes de entrenamiento de tamaño 28x28
```

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

[5]:
```
X_train = X_train / 255.
X_test = X_test / 255.
```

[6]:
```
model = Sequential() # Iniciamos el modelo

model.add(CuDNNLSTM(128, input_shape = (X_train.shape[1:]),  return_sequences =␣
 ↪True)) # La primera capa es LSTM
model.add(Dropout(0.2)) # Desactivamos el 20% de las neuronas
```

```
model.add(CuDNNLSTM(182)) # Capa LSTM
model.add(Dropout(0.2)) # Desactivamos el 20% de las neuronas

model.add(Dense(32, activation = 'relu')) # Capa densa
model.add(Dropout(0.2)) #Desactivamos el 20% de las neuronas

model.add(Dense(10, activation = 'softmax')) # Capa de salida con función␣
↪softmax
```

[7]:
```
opt = tf.keras.optimizers.Adam(lr = 1e-3, decay = 1e-5) # Definimos el␣
↪optimizador a usar
model.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = opt,
              metrics = ['accuracy']) #Compilamos el modelo
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

[8]:
```
# Entrenamiento del modelo
model.fit(X_train, y_train, epochs = 3, validation_data = (X_test, y_test))
```

```
Epoch 1/3
1875/1875 [==============================] - 54s 26ms/step - loss: 0.3862 -
accuracy: 0.8813 - val_loss: 0.1139 - val_accuracy: 0.9654
Epoch 2/3
1875/1875 [==============================] - 48s 26ms/step - loss: 0.1204 -
accuracy: 0.9671 - val_loss: 0.0728 - val_accuracy: 0.9788
Epoch 3/3
1875/1875 [==============================] - 48s 25ms/step - loss: 0.0785 -
accuracy: 0.9796 - val_loss: 0.0619 - val_accuracy: 0.9816
```

[8]:
```
<keras.callbacks.History at 0x7f86104ad710>
```